The Unwritten Contract of Cloud-based Elastic Solid-State Drives

Yingjia Wang and Ming-Chang Yang
The Chinese University of Hong Kong

***Abstract*—Elastic block storage (EBS) with the storage-compute disaggregated architecture stands as a pivotal piece in today's cloud. EBS furnishes users with storage capabilities through the elastic solid-state drive (ESSD). Nevertheless, despite the widespread integration into cloud services, the absence of a thorough ESSD performance characterization raises critical doubt: when more and more services are shifted onto the cloud, can ESSD satisfactorily substitute the storage responsibilities of the local SSD and offer comparable performance?**

In this paper, we for the first time target this question by characterizing two ESSDs from Amazon AWS and Alibaba Cloud. We present an unwritten contract of cloud-based ESSDs, encapsulating four observations and five implications for cloud storage users. Specifically, the observations are counter-intuitive and contrary to the conventional perceptions of what one would expect from the local SSD. The implications we hope could guide users in revisiting the designs of their deployed cloud software, i.e., harnessing the distinct characteristics of ESSDs for better system performance.

I. Introduction

Elastic block storage (EBS) has become the cornerstone of today's cloud, underpinning the high-performance storage demand from diverse cloud services [1]–[5]. As shown in Figure 1, EBS typically adopts a storage-compute disaggregated architecture, with compute and storage clusters located separately and interconnected by the high-speed datacenter network. In EBS, the virtual machines (VM) for users are hosted in the compute cluster while the user data is persistently stored in the storage cluster. This architectural innovation not only enhances the efficiency and elasticity of the cloud infrastructure, but also improves the cost-effectiveness for users on a flexible and pay-as-you-go basis.

EBS provides storage resources for users in the form of *elastic solid-state drive*, which, for simplicity, is called *ESSD* in the following. ESSD is a virtualized storage device attached to the user VM, employing the block interface and supporting random access (e.g., read and write) in the storage space. From the perspective of the user, ESSD is similar to the well-known local SSD, on which the existing filesystems and applications can be directly deployed. But differently, the most noteworthy feature of ESSD is that it breaks the physical performance and capacity boundaries of the local SSD. This makes it possible to elastically guarantee performance level (e.g., in throughput and IOPS) and allocate storage capacity, to accommodate the ever-changing demands of users.

This work is supported in part by The Research Grants Council of Hong Kong SAR (Project No. CUHK14218522).

The rapid development of cloud computing has fostered the widespread adoption of ESSDs in numerous cloud services. *However, to the best of our knowledge, there has been no comprehensive performance characterization of cloud-based ESSDs.* As the majority of host software now is designed and optimized for the local SSD, this research gap raises concerns in the following question: *when more and more services are shifted onto the cloud, can ESSD satisfactorily substitute the storage responsibilities of the local SSD and offer comparable performance?*

To this end, we make the first endeavor to answer this question by characterizing two ESSDs from two leading cloud storage providers in the world: Amazon AWS [6] and Alibaba Cloud [7]. Based on extensive experiments, we surprisingly find that ESSD exhibits several counter-intuitive performance features, contrary to the common beliefs of the local SSD.

The Unwritten Contract of Cloud-based ESSDs

Observations

1) *The latency of ESSDs is tens to a hundred times higher than that of SSD when I/Os are not well scaled up (i.e., I/O size is small and/or I/O queue depth is low).*
2) *The performance impact of GC appears much later or even disappears.*
3) *The throughput of random writes outperforms that of sequential writes, reaching a maximum of 1.52× and 2.79× in two ESSDs, respectively.*
4) *The maximum bandwidth is deterministic and no longer sensitive to the access pattern.*

Implications

1) *Scale the I/O sizes and I/O queue depths up as much as possible.*
2) *Reconsider if and how existing GC-mitigated techniques based on local SSDs should be correspondingly adapted for ESSDs.*
3) *Rethink if it is still worthwhile to convert random writes in random-write-based software into sequential writes and if it is beneficial to proactively trigger random writes in sequential-write-based software.*
4) *Smooth the read/write I/Os to be evenly distributed across the timeline and below the guaranteed throughput budget.*
5) *Re-evaluate I/O reduction techniques (e.g., compression, deduplication) that were previously considered to impair performance.*

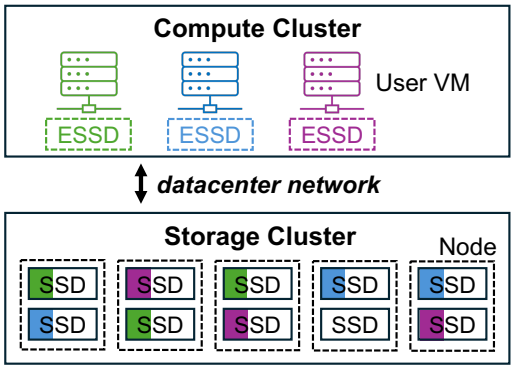


Fig. 1. **The storage-compute disaggregated architecture of elastic block storage (EBS).** ESSD is attached to the user virtual machine (VM) in the compute cluster. The physical storage space of an ESSD is distributed and replicated across different nodes and SSDs in the storage cluster.

We present an unwritten contract of cloud-based ESSDs, organized into four observations and five implications, as shown above. Over the past years, cloud storage users have been requesting ever-higher service-level agreements (SLA). This places higher expectations not only on cloud storage providers but also on users themselves in terms of their deployed software. The unwritten contract we offer in this paper could benefit cloud storage users to revisit their cloud software designs, specifically, to unleash the better potential of ESSDs by leveraging unique performance characteristics. We open-source the evaluation framework to stimulate the following studies[1].

In the following of this paper, we first provide the background of (local) SSD, EBS, and ESSD in §II. We present the unwritten contract in detail in §III. The related work and conclusion are introduced in §IV and §V, respectively.

II. Background

A. Flash-based Solid-State Drive (SSD)

Flash-based SSD has achieved a high and continuously increasing share in the storage market due to the benefits in performance, energy efficiency, and impact resistance. According to IDC, the byte shipment share of SSDs is expected to reach about 40% of the total in 2025 [8]. Flash-based SSD typically adopts a multi-level architecture, including channel, die, plane, block, and page. The flash die is the minimum unit of parallel operations while the flash page is the minimum unit of data storage. To improve SSD firmware management efficiency, flash blocks are typically grouped into superblocks, each of which comprises flash blocks across many flash dies to fully leverage flash parallelism.

To communicate with the host, the SSD longstandingly employs the *block interface*, which abstracts the storage space into an array of logical blocks (e.g., 4KB) that can be randomly read or written. While this simple abstraction has successfully contributed to the broad adoption of SSDs, a complex *flash translation layer (FTL)* is required in the SSD to bridge the gap with the underlying flash characteristics (e.g., erase-before-write). The two main duties of FTL are address mapping and garbage collection (GC). The address mapping converts logical addresses from the host to the physical addresses in the SSD by keeping track of a fine-grained (e.g., page-level) mapping table. The GC is carried out periodically to reclaim invalid space in the granularity of flash blocks, when the valid pages in some blocks are relocated and these blocks can be erased. FTL is also responsible for other tasks such as wear-leveling [9], error correction [10], and bad block management [11].

B. Elastic Block Storage (EBS)

As shown in Figure 1, *elastic block storage (EBS)* typically employs a storage-compute disaggregated architecture with independent compute and storage clusters, and the two clusters are not physically co-located but interconnected with the high-speed datacenter network. The *compute cluster* hosts virtual machines (VM) for users and forwards user requests to the backend storage cluster, while the *storage cluster* persists or fetches requested data in a distributed filesystem.

The disaggregation architecture of storage and compute brings out several benefits [1], [3], [5]. First, the storage and compute resources can be elastically paid and allocated according to the user target workloads. This allows for higher resource utilization while enhancing the cost-effectiveness for users, compared to offering general-purpose servers. Second, the resources in storage and compute clusters can be maintained and scaled independently, thereby realizing higher maintainability and scalability. Third, it is simpler and faster to migrate application services across compute servers, given that their states can be persistently stored in storage servers.

C. Elastic Solid-State Drive (ESSD)

EBS provides storage resources for users in the form of *elastic solid-state drive (ESSD)*, which is a virtualized storage device attached to user VM. ESSD can be created, attached, unattached, and destroyed without disrupting the user VM. Figure 1 illustrates the mapping relationship between the virtualized ESSD and the (multiple) physical SSDs. That is, the physical storage space of an ESSD is typically distributed and replicated (e.g., three-way [3]) across different nodes and SSDs in the storage cluster, for the purposes of load balancing and data availability.

From the user's point of view, ESSD is similar to the well-known local SSD in that ESSD also employs the block interface and supports random access in the storage space. This ensures seamless compatibility with the existing software ecosystem, and facilitates the existing file systems and applications can be directly deployed on ESSDs.

Nonetheless, beyond the local SSD, ESSD delivers several additional benefits [3], [6], [7]. First, ESSD is virtualized, which breaks the physical performance and capacity boundaries in the local SSD. As a result, both the performance level (e.g., in throughput and IOPS) and the storage capacity can

[1]https://github.com/yingjia-wang/UC

TABLE I
THE CONFIGURATIONS OF TWO ESSDS AND SSD.

	Provider and Type	Max. BW (GB/s)	Max. IOPS	Cap. (TB)	VM Type	Region
ESSD-1	Amazon AWS io2 [12]	∼3.0	25.6K	2	m6in.xlarge	Tokyo
ESSD-2	Alibaba Cloud PL3 [7]	∼1.1	100K	2	ecs.g5.4xlarge	Hangzhou
SSD	Samsung 970 Pro [13]	Seq. R/W 3.5/2.7	Rand. R/W 500K/500K (4KB, QD32)	1	/	/

be flexibly requested in response to the ever-changing user requirements. Second, ESSD can ensure high data availability and reliability, thanks to the data replication across multiple nodes to prevent single-point failure. Third, ESSD also enables some advanced features such as encryption and snapshotting.

Cloud storage providers typically offer multiple ESSD types with varying performance levels. Different performance levels guarantee different budgets in maximum throughput and maximum IOPS (higher is usually more expensive). For reference, Amazon AWS [6] supports general-purposes types (i.e., gp2, gp3) and provisioned-IOPS types (i.e., io1, io2), where io2 is the most advanced one. Alibaba Cloud [7] supports four performance levels: PL0, PL1, PL2, and PL3 (the higher the number the better the performance).

III. UNWRITTEN CONTRACT OF CLOUD-BASED ESSDS

Despite the prevalent integration in diverse cloud services, to the best of our knowledge, the performance features of the ESSD have not been adequately investigated and revealed. In this section, we present an unwritten contract of cloud-based ESSDs. In the following, we begin with the experimental setups, and then introduce our unwritten contract in detail.

Note that this paper focuses on the device-level performance characterization as in prior work [14]–[16] without exploring specific pieces of software, in the hope of benefiting a wider range of cloud storage users to revisit their software designs.

A. Experimental Setups

To strengthen the generalizability of findings, we characterize two ESSDs from two leading cloud providers in the world (i.e., Amazon AWS [6] and Alibaba Cloud [7]), and the configurations of ESSDs are summarized in Table I. Specifically, both ESSDs have the highest-class types available from their providers, making their maximum bandwidth comparable to that of the local SSD.

To demonstrate the features of the local SSD for comparison, we also evaluate Samsung 970 Pro [13], a popular SSD in research, and its configurations are also listed in Table I. Experiments on Samsung 970 Pro are running on a workstation PC, equipped with Intel® Xeon® W-2245 Processor, where the operating system is 64-bit Ubuntu 20.04 LTS with Linux kernel version 5.15.0.

We employ the widely-used FIO benchmark tool [17] to generate workloads with diverse loads (i.e., I/O size and I/O queue depth) and access patterns (i.e., random write, sequential write, random read, and sequential read). The detailed workload settings are introduced in each subsection below.

B. Latency Performance

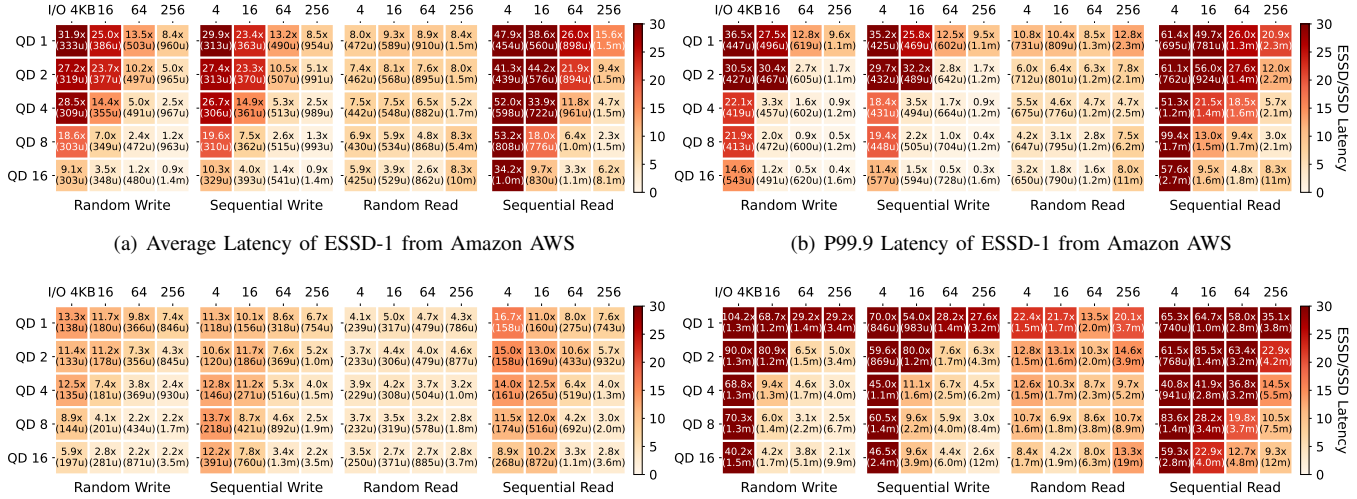
Observation #1: *The latency of ESSDs is tens to a hundred times higher than that of SSD when I/Os are not well scaled up (i.e., I/O size is small and/or I/O queue depth is low).*

ESSD can meet the user's best expectations if it can provide comparable performance (e.g., throughput and latency) to the local SSD. However, even though the maximum throughput that ESSD can provide reaches the level of GBs, we discover that the latency of ESSDs is extremely unsatisfactory when I/Os are not well scaled up. Here, we define *"not well scaled up"* as workloads dominated by small-size I/Os and/or low I/O queue depths. Figure 2 demonstrates the average and P99.9 latency of two ESSDs under different access patterns (i.e., random write, sequential write, random read, and sequential read) with I/O sizes varied from 4KB to 256KB and I/O queue depths varied from 1 to 16. Particularly, let us focus on the multiples the ESSD latency is divided by the SSD latency (i.e., the top of each pixel in Figure 2), with smaller numbers being better. For simplicity, we call this metric *latency gap* in the following of this subsection.

We can first observe that, the latency gap effectively decreases when I/Os are scaled in either sizes or queue depths. This finding is generally consistent among two metrics (i.e., average and P99.9 latency), four access patterns (i.e., random write, sequential write, random read, and sequential read), and two ESSD providers (i.e., Amazon AWS and Alibaba Cloud). Specifically, the average latency gaps of two ESSDs are up to 47.9× and 16.7×, respectively, while the P99.9 latency gaps are even much higher at up to 99.4× and 104.2×. In comparison, when the I/O size is scaled up to 256KB, the average latency gaps of two ESSDs are reduced to up to 15.6× and 7.6×, respectively, while the P99.9 latency gaps are mitigated to 20.9× and 35.1× at most. This trend also holds true when scaling up the I/O queue depths.

The primary cause of the high latency of ESSDs would be the network latency and software processing overhead within the cloud storage, which is still substantial compared to the local storage. Scaling the I/O sizes and I/O queue depths can effectively reduce the latency gap by leveraging parallelism in I/O processing and storage access distribution (see §II-C).

We can also notice that, the latency gaps in two ESSDs are significantly smaller in the random read workload but much higher in the other three workloads (i.e., random write, sequential write, and sequential read). Taking ESSD-1 as an example, the average and P99.9 latency gaps are up to 9.3× and 12.8× in the random read workload, but as high as 31.9×



(a) Average Latency of ESSD-1 from Amazon AWS

(b) P99.9 Latency of ESSD-1 from Amazon AWS

(c) Average Latency of ESSD-2 from Alibaba Cloud

(d) P99.9 Latency of ESSD-2 from Alibaba Cloud

Fig. 2. **Latency performance of two ESSDs under workloads with different access patterns (i.e., random write, sequential write, random read, sequential read), I/O sizes, and I/O queue depths (QD).** The top of each pixel indicates the multiples the ESSD latency is divided by the SSD latency, with smaller numbers being better. The bottom of each pixel indicates the value of average or P99.9 latency of the ESSD (u: μs, m: ms).

and 36.5×, 29.9× and 35.2×, 53.2× and 99.4× in the other three workloads.

This performance discrepancy is because modern SSD typically employs a DRAM-based write buffer to improve (both random/sequential) write performance and also uses prefetching to enhance sequential read performance [18]. Thus, in random/sequential write and sequential read workloads, there are much fewer flash accesses on the critical I/O path. However, the prefetching techniques work poorly for random reads, resulting in a larger number of flash accesses. This would make the inherent overhead in cloud storage (e.g., network latency, software processing) less prominent than in the other three workloads, thereby leading to a smaller latency gap.

The above results and analysis suggest that, to save the cloud software from tens to a hundred times latency penalty, cloud storage users should revisit their software designs to *scale the sizes and queue depths of I/Os up as much as possible* (**Implication #1**). Especially, when I/Os are well scaled up in random and sequential write workloads, ESSD-1 even achieves up to 3.3× better P99.9 latency compared to the local SSD (see Figure 2b).

C. Garbage Collection (GC)

> ***Observation #2:*** *The performance impact of GC appears much later or even disappears.*

Device-side garbage collection (GC) is widely recognized as one of the main culprits for SSD performance degradation and unpredictability [19]–[21]. The main reason behind this is that, to reclaim the space occupied by the invalid data, FTL periodically relocates the valid data in some flash blocks to others. The relocation of data results in a large number of extra writes, which severely compete with foreground I/Os and thus degrade the user-perceived performance. Nevertheless, in contrast to this common anticipation, we find that the performance impact of GC appears much later or even disappears in ESSDs. Figure 3 depicts the runtime throughput of two ESSDs and SSD under a random write workload, and the total write volume is configured to be 3× storage capacity of each device.

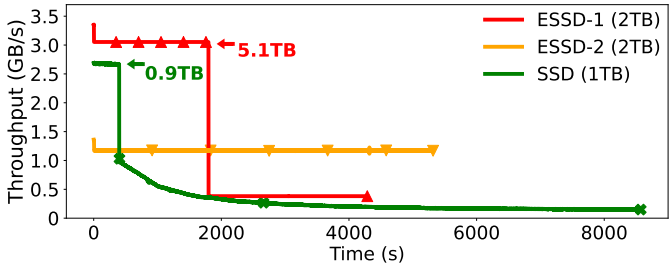


Fig. 3. **Runtime throughput of two ESSDs and SSD under a random write workload until writing 3× storage capacity.** Thus, the total write volumes towards ESSDs and SSD are 6TB and 3TB, respectively. The markers indicate the time when the total write volume reaches a multiple of 1TB.

We can first verify that, the SSD starts to experience a severe throughput drop when only writing 0.9TB (i.e., 90% storage capacity), and the throughput sharply decreases by 63% from 2.7GB/s to 1.0GB/s. After that, the throughput of SSD continues to decrease further as the write volume increases, down to a low of 149MB/s finally. The long-term low performance here demonstrates the sensitivity and powerlessness of the SSD performance to GC. Due to the high and constant write load, GC is unavoidably executed in the SSD almost all the time.

In contrast, ESSD can sustain high throughput over a much longer period even in this high-pressure workload, although the exact performance of an ESSD may depend on the provider. Specifically, ESSD-1 undergoes a sharp throughput

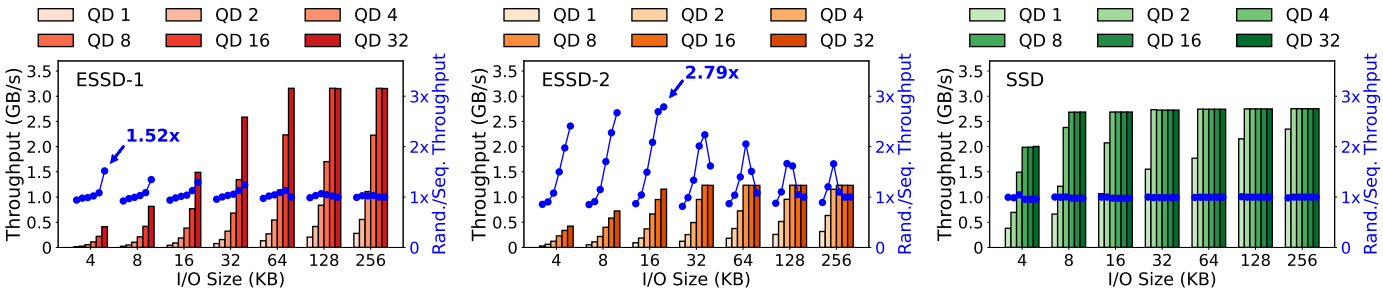


Fig. 4. **Throughput of ESSDs and SSD under random write workloads and throughput gain of ESSDs and SSD in random writes over sequential writes.** The I/O size varies from 4KB to 256KB and the I/O queue depth (QD) varies from 1 to 32.

decline when writing as large as 5.1TB (i.e., 2.55× storage capacity), after which the throughput is stabilized at about 305MB/s. On the contrary, ESSD-2 can maintain a high throughput consistently until writing 6TB (i.e., 3× storage capacity). We speculate on the rationale behind this as follows. On the one hand, cloud providers make efforts to utilize their plenty of storage resources to hide the GC impact transparently for better user experiences. On the other hand, cloud providers may trigger flow-limiting mechanisms when they can not hide the GC impact anymore.

Actually, it has been a long journey for researchers to study software techniques that can mitigate the performance impact of GC [22]–[24]. However, our results and analysis indicate that, cloud software should *reconsider if and how existing GC-mitigated techniques based on local SSDs should be correspondingly adapted for ESSDs* (**Implication #2**), due to the trade-offs they commonly entail.

D. Access Pattern

> ***Observation #3:*** *The throughput of random writes actually outperforms that of sequential writes, reaching a maximum of 1.52× and 2.79× in two ESSDs, respectively.*

Random writes have long been considered harmful to the SSD due to the exacerbated mix of valid/invalid data in the same flash blocks, which consequently results in higher valid data relocation overhead [25]–[27]. However, surprisingly, we discover that the throughput of the ESSD is generally better in the case of random rather than sequential writes. As shown in Figure 4, let us focus on the throughput gain of ESSDs and SSD in random writes over sequential writes (i.e., blue lines). For simplicity, we call this metric *throughput gain* in the following of this subsection. The I/O size varies from 4KB to 256KB and the I/O queue depth varies from 1 to 32.

We can first observe that, the throughput gains in two ESSDs are up to 1.52× and 2.79×, respectively. On the contrary, the throughput of the SSD does not show obvious differences in random or sequential writes (when GC does not occur). The main reason behind this would be the unique physical storage space mapping of ESSDs. As introduced in §II-C, the storage space of an ESSD is distributed and replicated across different nodes and SSDs in the storage cluster. Therefore, data from random writes is likely to embrace higher available SSD bandwidth, potentially leading to higher user-perceived throughput.

Additionally, we discover a notable variation in throughput gains amongst ESSD providers. Specifically, ESSD-1 has a smaller throughput gain, which is mainly concentrated on workloads with higher I/O queue depths and small-to-medium I/O sizes. For example, when the I/O size ranges from 4KB to 32KB and the I/O queue depth is 32, the throughput gain of ESSD-1 is 24.1%∼51.8%. In contrast, the throughput gain in ESSD-2 is much more significant, at up to 1.66× to 2.79× across a wide range of I/O sizes (i.e., 4KB to 256KB). The throughput gain in ESSD-2 also varies a lot in different workload scenarios. For small-size I/Os (e.g., 4KB to 16KB), the throughput gain generally increases as the queue depth increases. For medium-to-large I/Os (e.g., 32KB and larger), the throughput gain has a trend of growing and then declining as the queue depth increases; moreover, the peak of the throughput gain occurs earlier as the I/O size increases.

Converting random writes into sequential writes (e.g., log-structured [28] or copy-on-write [29]) has been widely recognized to reduce device-side GC overhead and improve SSD performance. However, given the ESSD performance is quite insensitive to GC (see §III-C) and the performance gain from random writes, it is now essential to *rethink if it is worthwhile to convert random writes in random-write-based software into sequential writes anymore and if it is beneficial to proactively trigger random writes in sequential-write-based software* (**Implication #3**).

E. Throughput Budget

> ***Observation #4:*** *The maximum bandwidth is now deterministic and no longer sensitive to the access pattern.*

The maximum throughput of the SSD largely depends on the access pattern, mainly due to the asymmetric latency of different flash operations. For example, since flash reads typically have shorter latency than flash writes, the maximum read bandwidth is usually higher than the maximum write bandwidth (e.g., 3.5GB/s vs. 2.7GB/s in Samsung 970 Pro, see Table I). However, we find that the maximum bandwidth of ESSDs is deterministic and no longer sensitive to the access pattern. Figure 5 shows the throughput of two ESSDs and

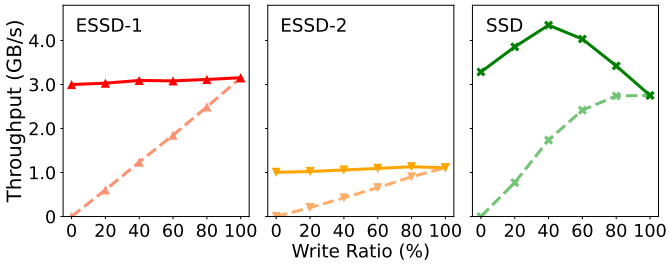


Fig. 5. **Throughput of two ESSDs and SSD under mixed read/write workloads with different write ratios.** The solid and dashed lines represent the total (i.e., read and write) throughput and the write throughput, respectively.

SSD under mixed read/write workloads. The workloads have different write ratios from 0 to 100, where 0 and 100 denote the random read and write workloads, respectively.

We can observe that, regardless of the specific write ratio, the throughput of two ESSDs is deterministically surrounded at the value guaranteed by the provider, i.e., 3.0GB/s and 1.1GB/s, respectively. In contrast, the throughput of the SSD is non-deterministic, varying between 2.5GB/s and 4.3GB/s in different write ratios. Note that this observation holds only for throughput and not for IOPS, as we find that the guaranteed IOPS in ESSDs is non-deterministic and is closely related to the I/O size.

The above results and analysis actually serve as a reminder to cloud storage users that, in contrast to the intricate performance engineering on the SSD, the throughput optimization target of the ESSD is straightforward and clear. For cloud storage users, this can deliver two implications.

First, since the throughput budget directly affects the system cost (see §II-C), to achieve higher cost-effectiveness through a smaller throughput budget, cloud software should *smooth the read/write I/Os to be evenly distributed across the timeline and below the guaranteed throughput budget* (**Implication #4**). This implication would be much more meaningful for software with frequent I/O bursts (i.e., I/O loads are very uneven across the timeline).

Second, to achieve higher cost-effectiveness, cloud software should also *re-evaluate I/O reduction techniques (e.g., compression [30], deduplication [31]) that were previously considered to impair performance* (**Implication #5**). The rationale behind this is that, for local SSDs with low latency, these I/O reduction techniques usually result in performance degradation due to high computational overhead [30], [31]. However, for ESSDs with network latency and software processing overhead that accompany cloud storage, the overhead of these techniques may not be the bottleneck, leading to not only cost reduction (i.e., accommodate workloads with a smaller throughput budget) but also performance improvement.

IV. Related Work

Depicting the performance characteristics of emerging storage devices is always a timeless topic. Prior works have investigated the performance features of HDD [32], flash-based SSD [27], Optane-based SSD [14], and flash-based SSD with new interfaces such as Key-Value (KV) [15] and Zoned Namespace (ZNS) [16]. Inspired by them, we first characterize the performance of the emerging cloud-based ESSD.

Some works introduce EBS's designs and evolution in storage, networking, and hardware acceleration [1], [3]–[5]. Li et al. characterize the I/O features of workloads on EBS from Alibaba Cloud and Tencent Cloud [2]. Cosine [33] searches for the best key-value storage engine with its cost models considering an input workload, a cloud budget, a target performance, and required cloud SLAs. Many works focus on data management between EBS and other faster/slower tiers (e.g., local SSD, object store such as Amazon S3 [34]). For example, Mutant [35] proposes to organize different SSTables in the key-value store into different storage tiers based on access frequencies. ROCKSMASH [36] proposes new cache and metadata structure designs to use local SSDs to store frequently accessed data and metadata while using EBS to hold the rest. MirrorKV [37] focuses on hybrid cloud storage consisting of EBS and object store, and proposes a series of designs to address the efficiency challenges of compaction and querying in this architecture.

Zhou et al. reveal the high latency variances in ESSDs when exceeding the paid IOPS budget and propose a range of I/O-regulated techniques in the key-value store to improve tail latency [38]. In contrast, we focus on the comprehensive performance characterization of ESSDs and deliver insights from many aspects (e.g., latency performance, GC, access pattern, throughput budget).

V. Conclusion and Future Work

This paper pioneers a thorough performance characterization of the cloud-based ESSD, and draws an unwritten contract that incorporates four counter-intuitive observations and five intriguing implications. We hope the unwritten contract could guide cloud storage users in revisiting their deployed cloud software designs, particularly, harnessing the distinct performance characteristics of ESSDs for better system performance.

Meanwhile, given the increasingly rapid growth of cloud computing and cloud storage, we believe this research direction deserves more investigation. We outline the future work as follows.

- **Comprehensiveness and generalizability.** We acknowledge that our current observations may not cover all the unique properties of ESSDs. In the future, we will continue to investigate, and also evaluate ESSDs from more cloud storage providers and validate our observations in this paper on these ESSDs.
- **Cloud-software-level exploration.** In the future, we will explore how cloud software can take advantage of the implications offered in this paper. In particular, we will first use RocksDB [39] (a popular persistent key-value store) as a case study. Although some works have explored building key-value stores on cloud storage (see §IV), these works may not fully exploit the potential of ESSDs.

REFERENCES

[1] R. Miao, L. Zhu, S. Ma, K. Qian, S. Zhuang, B. Li, S. Cheng, J. Gao, Y. Zhuang, P. Zhang *et al.*, "From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 753–766.

[2] J. Li, Q. Wang, P. P. Lee, and C. Shi, "An in-depth comparative analysis of cloud block storage workloads: Findings and implications," *ACM Transactions on Storage*, vol. 19, no. 2, pp. 1–32, 2023.

[3] W. Zhang, E. Xu, Q. Wang, X. Zhang, Y. Gu, Z. Lu, T. Ouyang, G. Dai, W. Peng, Z. Xu *et al.*, "What's the story in {EBS} glory: Evolutions and lessons in building cloud block store," in *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, 2024, pp. 277–291.

[4] Z. Wang, Y. Song, E. Xu, H. Wu, G. Tong, S. Sun, H. Li, J. Liu, L. Ding, R. Liu *et al.*, "Ransom access memories: Achieving practical ransomware protection in cloud with {DeftPunk}," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 687–702.

[5] J. Shu, K. Qian, E. Zhai, X. Liu, and X. Jin, "Burstable cloud block storage with data processing units," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 783–799.

[6] "Amazon AWS ESSD," https://docs.aws.amazon.com/ebs/latest/userguide/ebs-volumes.html, 2024.

[7] "Alibaba Cloud ESSD," https://www.alibabacloud.com/help/en/ecs/user-guide/essds, 2024.

[8] "IDC Data Age 2025," https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf, 2017.

[9] Z. Jiao, J. Bhimani, and B. S. Kim, "Wear leveling in ssds considered harmful," in *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, 2022, pp. 72–78.

[10] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "{LDPC-in-SSD}: Making advanced error correction codes work effectively in solid state drives," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 243–256.

[11] J.-N. Yen, Y.-C. Hsieh, C.-Y. Chen, T.-Y. Chen, C.-L. Yang, H.-Y. Cheng, and Y. Luo, "Efficient bad block management with cluster similarity," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 503–513.

[12] "Amazon EBS volume types," https://aws.amazon.com/ebs/volume-types/, 2024.

[13] "Samsung 970 Pro," https://semiconductor.samsung.com/consumer-storage/internal-ssd/970pro/, 2024.

[14] K. Wu, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Towards an unwritten contract of intel optane {SSD}," in *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, 2019.

[15] M. P. Saha, A. Maruf, B. S. Kim, and J. Bhimani, "Kv-ssd: What is it good for?" in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1105–1110.

[16] K. Doekemeijer, N. Tehrany, B. Chandrasekaran, M. Bjørling, and A. Trivedi, "Performance characterization of nvme flash devices with zoned namespaces (zns)," in *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2023, pp. 118–131.

[17] "FIO benchmark," https://fio.readthedocs.io/en/latest/, 2024.

[18] N. Li, M. Hao, H. Li, X. Lin, T. Emami, and H. S. Gunawi, "Fantastic ssd internals and how to learn and use them," in *Proceedings of the 15th ACM International Conference on Systems and Storage*, 2022, pp. 72–84.

[19] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien, and H. S. Gunawi, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in nand ssds," *ACM Transactions on Storage (TOS)*, vol. 13, no. 3, pp. 1–26, 2017.

[20] N. Elyasi, C. Choi, A. Sivasubramaniam, J. Yang, and V. Balakrishnan, "Trimming the tail for deterministic read performance in ssds," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2019, pp. 49–58.

[21] H. Li, M. L. Putra, R. Shi, X. Lin, G. R. Ganger, and H. S. Gunawi, "Ioda: A host/device co-design for strong predictability contract on modern flash storage," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 263–279.

[22] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt, "Flash on rails: Consistent flash performance through redundancy," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 463–474.

[23] J. Kim, K. Lim, Y. Jung, S. Lee, C. Min, and S. H. Noh, "Alleviating garbage collection interference through spatial separation in all flash arrays," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 799–812.

[24] T. Jiang, G. Zhang, Z. Huang, X. Ma, J. Wei, Z. Li, and W. Zheng, "{FusionRAID}: Achieving consistent low latency for commodity {SSD} arrays," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 355–370.

[25] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 181–192, 2009.

[26] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "Sfs: random write considered harmful in solid state drives." in *FAST*, vol. 12, 2012, pp. 1–16.

[27] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The unwritten contract of solid state drives," in *Proceedings of the twelfth European conference on computer systems*, 2017, pp. 127–144.

[28] C. Lee, D. Sim, J. Hwang, and S. Cho, "{F2FS}: A new file system for flash storage," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 273–286.

[29] O. Rodeh, J. Bacik, and C. Mason, "Btrfs: The linux b-tree filesystem," *ACM Transactions on Storage (TOS)*, vol. 9, no. 3, pp. 1–32, 2013.

[30] A. Zuck, S. Toledo, D. Sotnikov, and D. Harnik, "Compression and {SSDs}: Where and how?" in *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*, 2014.

[31] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[32] S. W. Schlosser and G. R. Ganger, "Mems-based storage devices and standard disk interfaces: A square peg in a round hole?" in *FAST*, 2004, pp. 87–100.

[33] S. Chatterjee, M. Jagadeesan, W. Qin, and S. Idreos, "Cosine: a cloud-cost optimized self-designing key-value storage engine," *Proceedings of the VLDB Endowment*, vol. 15, no. 1, pp. 112–126, 2021.

[34] "Amazon S3," https://aws.amazon.com/s3/, 2024.

[35] H. Yoon, J. Yang, S. F. Kristjansson, S. E. Sigurdarson, Y. Vigfusson, and A. Gavrilovska, "Mutant: Balancing storage cost and latency in lsm-tree data stores," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 162–173.

[36] P. Xu, N. Zhao, J. Wan, W. Liu, S. Chen, Y. Zhou, H. Albahar, H. Liu, L. Tang, and Z. Tan, "Building a fast and efficient lsm-tree store by integrating local storage with cloud storage," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 3, pp. 1–26, 2022.

[37] Z. Wang and Z. Shao, "Mirrorkv: An efficient key-value store on hybrid cloud storage with balanced performance of compaction and querying," *Proceedings of the ACM on Management of Data*, vol. 1, no. 4, pp. 1–27, 2023.

[38] Y. Zhou, J. Zhou, S. Chen, P. Xu, P. Wu, Y. Wang, X. Liu, L. Zhan, and J. Wan, "Calcspar: A {Contract-Aware}{LSM} store for cloud storage with low latency spikes," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 451–465.

[39] "RocksDB: A persistent key-value store for fast storage environments," http://rocksdb.org/, 2024.

AASD: Accelerate Inference by Aligning Speculative Decoding in Multimodal Large Language Models

Chaoqun Yang[1,2,5], *Ran Chen*[3], *Muyang Zhang*[4], *Weiguang Pang*[1,5], *Yuzhi Chen*[4]
Rongtao Xu[4,1], *Kexue Fu*[1,5], *Changwei Wang*[1,5,*], *Longxiang Gao*[1,5,*],
[1]Key Laboratory of Computing Power Network and Information Security, Ministry of Education,
Shandong Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences)
[2] Shenzhen International Graduate School, Tsinghua University
[3] Department of Information and Computational Sciences, School of Mathematical Sciences, Peking University
[4]School of Artificial Intelligence, University of Chinese Academy of Sciences
[5]Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing,
Shandong Fundamental Research Center for Computer Science

***Abstract*—Multimodal Large Language Models (MLLMs) have achieved notable success in visual instruction tuning, yet their inference is time-consuming due to the auto-regressive decoding of Large Language Model (LLM) backbone. Traditional methods for accelerating inference, including model compression and migration from language model acceleration, often compromise output quality or face challenges in effectively integrating multimodal features. To address these issues, we propose AASD, a novel framework for Accelerating inference with refined KV Cache and Aligning speculative decoding in MLLMs. Our approach leverages the target model's cached Key-Value (KV) pairs to extract vital information for generating draft tokens, enabling efficient speculative decoding. To reduce the computational burden associated with long multimodal token sequences, we introduce a KV Projector to compress the KV Cache while maintaining representational fidelity. Additionally, we design a Target-Draft Attention mechanism that optimizes the alignment between the draft model and the target model, achieving the benefits of real inference scenarios with minimal computational overhead. Extensive experiments on mainstream MLLMs demonstrate that our method achieves up to a *2*× inference speedup without sacrificing accuracy. This study not only provides an effective and lightweight solution for accelerating MLLM inference but also introduces a novel alignment strategy for speculative decoding in multimodal contexts, laying a strong foundation for future research in efficient MLLMs. Code is availiable at https://github.com/transcend-0/ASD.**

***Index Terms*—Multimodal Large Language Model, Inference Acceleration, Speculative Decoding**

I. Introduction

The rapid development of Multimodal Large Language Models (MLLMs) is exerting a profound impact on the whole world. These models have demonstrated remarkable capabilities in various domains, including visual understanding, question answering, logical reasoning *etc.* [26, 27], and they are expected to further drive artificial intelligence toward Artificial General Intelligence. However, despite the widespread recognition of MLLMs' accuracy and versatility, inference speed remains a significant challenge in practical applications. Current multimodal models generally rely on autoregressive decoding, which, while ensuring coherent outputs, greatly limits inference speed. As model size and complexity increase, so do the computational cost and resource demands associated with their inference [6]. So how to accelerate the inference of the MLLMs is an important problem, which has both academic value and practical significance.

At present, research focusing on accelerating the inference of MLLMs remains relatively scarce. Algorithmic research primarily falls into three categories:

- Traditional model compression techniques: such as distillation [23], quantization [11, 18], and pruning [25, 37] aim to reduce model size and computational burden but encounter issues with fidelity loss and implementation complexity.
- Compact architectures: leveraging lightweight models [22, 33] and efficient encoders such as vision token compression [28, 30]; however, they may sacrifice multimodal alignment and overall performance.
- Adaptation of LLM acceleration techniques to MLLMs: such as efficient attention [22] and MoE [14], offers potential solutions but faces challenges in integrating visual features primely.

Most of these methods tend to compromise output quality, which is unacceptable for many precision-demanding applications. Recently, Speculative Decoding (SD) has emerged as a promising approach for accelerating inference without compromising accuracy, and it has shown effective utility in (Natural Language Processing) NLP tasks [7, 29, 36]. Recent work has attempted to migrate this technique to multimodal scenarios. Gagrani et al. (2024) proposes the speculative decoding with a language-only draft model to improve inference efficiency. However, a language-only draft model struggles to align closely with MLLMs, leading to limited acceleration benefits that fail to meet practical demands. On the other hand, speculative decoding usually requires a small

This work was supported by National Science and Technology Major Project (No.2022ZD0116800), and in part by the Taishan Scholars Program (Nos. TSQN202211214 and TSQN202408245), Shandong Excellent Young Scientists Fund Program (Overseas) No.2023HWYQ-113, Shandong Provincial Natural Science Foundation for Young Scholars (Nos.ZR2024QF110 and ZR2024QF052), National Natural Science Foundation of China (No. U21A20515, 62376271, U22B2034, 62172416, 52175493, 62365014, and 62262043), Beijing Natural Science Foundation(No. L231013), and the Key Laboratory of Computing Power Network and Information Security, Ministry of Education under Grant No.2024PY021.

*Changwei Wang and Longxiang Gao are the corresponding authors (Email: chanweiwang@sdas.org; gaolx@sdas.org). This is work done by Chaoqun Yang and Ran Chen during their internship at the Shandong Computer Science Center.

model that has a high acceptance rate and low inference time to achieve an acceleration effect [29, 36]. This requirement is relatively easy to fulfill in NLP tasks but poses a significant challenge in multimodal contexts, where tokens are longer and encode more extensive information. Smaller models with fewer parameters struggle to capture these rich multimodal representations, hindering their ability to effectively align with the target model. Conversely, using a larger draft model to better capture this information adds computational overhead, thereby increasing inference latency.

To address these challenges, we proposed **AASD**: **A**ccelerate the inference by refined KV Cache and **A**lign **S**peculative **D**ecoding in Multimodal Large Language Models: **(1) Accelerate the inference by refined KV Cache.** Typical speculative decoding uses the *draft-then-verify* paradigm. During inference, the draft model first generates multiple draft tokens, which are then verified in parallel by the target model in a single call. We observed that during inference, the target model caches Key-Value (KV) pairs, which contain significant information about the target distribution. Leveraging this insight, we designed a speculating module that utilizes the last layer's KV Cache from the target model to extract pertinent information for generating draft tokens. Considering that the Key and Value of multimodal tokens are relatively long, which increases the inference costs and the training difficulty of the speculating module, we introduced the KV Projector. This projector is designed to compress information from the multimodal KV Cache and then calculate Cross Attention with the Query from the speculating module.

(2) Align speculative decoding in Multimodal Large Language Models. Utilizing the target model's KV Cache during inference allows the draft model to learn representations closer to those of the target model, but it poses a challenge for training. When generating the *k*-th draft token at the *i*-th step during inference, it requires access to the first *i-1* KV pairs from the target model and the *i*-th to *(i+k-1)*-th KV pairs from the draft model. Replicating this scenario exactly in training is complex and costly. A regular lower triangular causal mask cannot achieve this goal, and accurately mirroring inference conditions would incur $O(n^2)$ computational and memory costs, which is a tough problem. To overcome this, we developed the Target-Draft Attention (T-D Attn), an optimized mechanism that achieves the effects of real inference with minimal additional computation, thereby addressing the challenge of aligning training with the inference scenario in a computationally efficient manner.

Our extensive experiments on various mainstream MLLMs validate that the proposed method achieves an effective acceleration for inference (around $2\times$ speedup). Our work not only provides a lightweight acceleration solution for efficient MLLM inference, but also introduces a novel approach for aligning SD with multimodal contexts, establishing a solid foundation for future research in efficient MLLM inference. The main contributions of this study are summarized as follows: ***i)*** Lightweight inference acceleration method: A lightweight, efficient inference acceleration framework specifically designed for MLLMs, achieving significant speed improvements while maintaining model accuracy, providing a feasible solution for real-time MLLM applications. ***ii)*** Innovative multimodal alignment strategy: we proposed an innovative strategy to align SD with multimodal contexts, enabling the model to effectively leverage the synergy between visual and textual information in multimodal input scenarios, further enhancing decoding efficiency and information fusion. ***iii)*** Strong Empirical Evidence of Acceleration: Through extensive empirical validation, the proposed method achieved up to $2\times$ speedup in inference, demonstrating its high efficiency and robustness in practical applications.

II. Related Work

A. Accelerating Inference of Multimodal Large Language Models

Multimodal Large Language Models (MLLMs) are large language models (LLMs) that integrate multimodal capabilities, enabling them to receive, interpret, and generate outputs based on multimodal information [34]. These models are engineered to tackle complex tasks, including image description, visual question answering (VQA), multimodal translation, and instruction-following by jointly processing and generating natural language along with visual and other perceptual data.

A typical MLLM architecture comprises three core components: a multimodal encoder, a LLM backbone, and a multimodal connector to coordinate multiple modalities [5]. The input to an MLLM often includes raw multimodal data, such as images, audio, and text, which are first processed by modality-specific encoders and transformed into a standardized format suitable for the model's processing. The model's output depends on the task at hand, for instance, in VQA, the output may be a text answer to a visual query, while in image description, the output is a textual description of the image content.

MLLMs usually require significant computational resources for both training and inference. When handling high-resolution images or performing complex reasoning tasks, the inference latency can be considerable, which poses challenges for real-time applications [12]. As a result, accelerating MLLM inference has become an active research area, with key approaches including: **Vision Token Compression.** Techniques such as token pruning [35] and the design of novel vision-language bridging modules [4] aim to reduce the number of visual tokens, enhancing computational efficiency. **Bypassing Visual Tokens.** Speculative decoding methods enable selective bypassing of image tokens and associated processing components [3], which reduces computational load by withdrawing visual tokens at specific layers [16]. **Hardware Acceleration.** Solutions such as the development of elastic cache systems have been proposed to improve the inference efficiency of instruction-following multimodal models [19].

B. Speculative Decoding

Speculative Decoding uses the idea of *draft-then-verify* to fully leverage the parallel processing capabilities of GPU [13]. Specifically, this method initially utilizes a draft model, typically a small and rapidly executing model, to generate multiple draft tokens. Subsequently, the target model verifies these draft tokens in a single, parallel operation, reducing the number of auto-regressive decoding steps of the target model, thereby accelerating the overall inference process [29]. The speedup provided by speculative decoding mainly depends on the acceptance of the draft tokens [13, 31]. In other words, the alignment of the draft model and the target model largely determines the upper limit of acceleration.

III. Method

A. Preliminaries

Below we use lowercase letters for vectors and uppercase letters for matrices except $\boldsymbol{X}$, which represents model inputs. For ease of reading, all vectors default to row vectors for ease

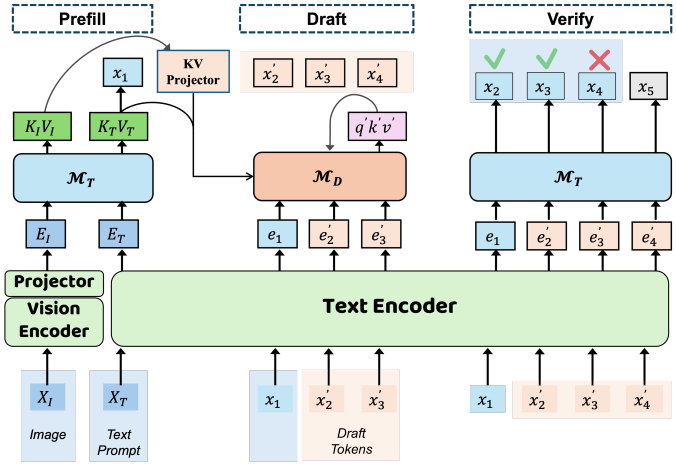


Fig. 1. The overview of the pipeline in the inference phase. We use refined KV cache from the target model (M_T) to enhance the draft model (M_D).

of reading and the representation of the partitioned matrix does not clearly distinguish the column and column directions, such as $K_n = (K_{1\sim i}, K_{i+1\sim n})$, where $K_n \in R^{n\times d}$, $K_{1\sim i} \in R^{i\times d}$ and $K_{i+1\sim n} \in R^{i\times d}$.

1) Multimodal Large Language Models: A typical Multimodal Large Language Model (MLLM) comprises 3 modules: 1) a modality encoder to encode visual inputs, 2) a connector to map the visual encodings to text embeddings, and 3) a LLM backbone to generate response. In inference, inputting an image $\boldsymbol{X}_I$ and a prompt text $\boldsymbol{X}_T$, the modality encoder converts the image $\boldsymbol{X}_I$ to visual encodings, which is then transformed into embeddings E_I in the text embedding space. Meanwhile, LLM's encoder converts the prompt text into text embeddings E_T. And then LLM backbone autoregressively generates tokens using these embeddings. The whole process can be expressed by the following formula:

$$x \sim p(x|\boldsymbol{X}_I, \boldsymbol{X}_T) = M(\boldsymbol{X}_I, \boldsymbol{X}_T) \tag{1}$$

where x denotes the token sampled from the probability space p generated by the model M conditioning on the input image $\boldsymbol{X}_I$ and prompt text $\boldsymbol{X}_T$.

2) Speculative Decoding: Speculative decoding is a lossless approach to accelerate the inference of autoregressive models, adopting a *draft-then-verify* paradigm. To expedite inference of a large model, commonly referred to as the *target model* (M_T), this method employs a small auxiliary model, referred to as the *draft model* (M_D), which is typically obtained through fine-tuning or distillation.

B. Accelerated Inference by Refined KV Cache

1) KV Cache: The core of LLM is the attention computing mechanism:

$$\boldsymbol{O} = \boldsymbol{A}\boldsymbol{V} = Softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})\boldsymbol{V} \tag{2}$$

where $\boldsymbol{O} \in R^{n\times d}$ is the attention output, $\boldsymbol{A} \in R^{n\times n}$ is the attention weight, $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in R^{n\times d}$ correspond to query states, key states, and value states respectively. n denotes the sequence length, and d denotes the hidden dimension. Typically, the computation of $\boldsymbol{A}$ incorporates a lower triangular matrix known as a causal mask to prevent subsequent tokens from attending to preceding tokens, thereby maintaining the autoregressive property of the model. For ease of reading, we use simplified symbols ($\boldsymbol{Q}\boldsymbol{K}^T$) to represent $Softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})$ below.

Due to the nature of autoregressive generation, where each computation step utilizes the $\boldsymbol{K}$ and $\boldsymbol{V}$ representations of preceding tokens, it is common practice to cache $\boldsymbol{K}$ and $\boldsymbol{V}$ at each inference step for subsequent use. This caching strategy is crucial for maintaining the efficiency of the generation process. The procedure in MLLMs can be detailed as follows:

(1) Prefilling stage. An input image $\boldsymbol{X}_I$ token sequence $\boldsymbol{X}_T$ is processed to compute the attention output. Concurrently, the key representations $\boldsymbol{K} = (\boldsymbol{K}_I, \boldsymbol{K}_T)$ and value representations $\boldsymbol{V} = (\boldsymbol{V}_I, \boldsymbol{V}_T)$ derived from this sequence are cached. This initial computation establishes the foundational context for the autoregressive generation process.

(2) Decoding stage. For each subsequent token input x, its corresponding query $\boldsymbol{q}$, key $\boldsymbol{k}$, and value $\boldsymbol{v}$ are computed. These are then concatenated with the previously cached $\boldsymbol{K}$ and $\boldsymbol{V}$ to calculate the attention outputs. That is

$$\boldsymbol{o} = (\boldsymbol{q}\tilde{\boldsymbol{K}})\tilde{\boldsymbol{V}} \tag{3}$$

where

$$\tilde{\boldsymbol{K}} = Concat(\boldsymbol{K}, \boldsymbol{k}), \tilde{\boldsymbol{V}} = Concat(\boldsymbol{V}, \boldsymbol{v}) \tag{4}$$

Our inference framework, shown in Figure 1, addresses the unique challenges of applying Speculative Decoding (SD) in multimodal tasks. Unlike NLP tasks, achieving satisfactory performance with a smaller draft model in multimodal settings is significantly more difficult. Current relatively compact models, such as KarmaVLM with a 0.4B vision encoder and a 0.5B LLM backbone [24], DeepSeek-VL with a 0.4B vision encoder and a 1.3B LLM backbone [20], and MiniCPM-V 2.0 with a 0.4B vision encoder and a 2.7B LLM backbone [33], still have a huge number of parameters. Moreover, it has been pointed out that a small MLLM as a draft model brings limited acceleration gain to SD, sometimes even worse than a draft model with language-only LLM [3]. This is most likely because learning directly from multimodal data is too difficult for small models. This motivated us to design **AASD**: An approach that accelerate the inference by effectively refining KV Cache generated by the target model to better align the draft model outputs with the target model. We structure our method in two key stages: Prefilling and Decoding.

(1) Prefilling stage. During the Prefilling stage, the target model processes the input image $\boldsymbol{X}_I$ and prompt text $\boldsymbol{X}_T$ to generate the initial KV Cache $\boldsymbol{K}_I, \boldsymbol{V}_I, \boldsymbol{K}_T, \boldsymbol{V}_T$ and the probability distribution over the next token. Although these KV Caches contain valuable information, the image KV Cache tends to be particularly large, creating both a learning barrier and a computational burden for the draft model. To address this, we introduce two feature projectors that compress the target model's KV Cache, enabling the draft model to better align with the target model during draft generation. Mathematically, this compression is represented as:

$$\boldsymbol{K}_I^* = \boldsymbol{W}_K \boldsymbol{K}_I, \boldsymbol{V}_I^* = \boldsymbol{W}_V \boldsymbol{V}_I \tag{5}$$

where

$$\boldsymbol{W}_K, \boldsymbol{W}_V \in R^{k\times n}, \boldsymbol{K}_I, \boldsymbol{V}_I \in R^{n\times d}, \boldsymbol{K}^*, \boldsymbol{V}^* \in R^{k\times d}$$

This operation compresses the lengthy sequence of visual key and value states into k tokens, here set to $k = 64$, effectively reducing approximately 90% of the redundant information. The

target model then passes the compressed KV pairs, $\boldsymbol{K}_I^*$, $\boldsymbol{V}_I^*$, $\boldsymbol{K}_T$, and $\boldsymbol{V}_T$, to the draft model to generate the draft tokens.

(2) Decoding stage.

(2.1) Draft process. First, the draft model builds upon foregoing KV Cache generated by the target model to produce a sequence of candidate tokens. Specially, the draft model M_D receives token $\boldsymbol{x}_i$ from the target model and computes corresponding $\boldsymbol{q}'_i, \boldsymbol{k}'_i, \boldsymbol{v}'_i$, which are then combined with the target model's key states $\boldsymbol{K}_{i-1} = \boldsymbol{k}_{\leq i-1}$ and value states $\boldsymbol{V}_{i-1} = \boldsymbol{v}_{\leq i-1}$ up to the previous token, namely

$$\hat{\boldsymbol{K}}_i = (\boldsymbol{K}_{i-1},\ \boldsymbol{k}'_i), \hat{\boldsymbol{V}}_i = (\boldsymbol{V}_{i-1}, \boldsymbol{v}'_i) \tag{6}$$

The draft model computes the attention using $\boldsymbol{q}'_i, \hat{\boldsymbol{K}}_i, \hat{\boldsymbol{V}}_i$, and generated next draft token $\boldsymbol{x}'_{i+1}$. Then draft model continues to generate subsequent tokens by incorporating the newly generated tokens into the KV Cache, namely

$$x'_{i+k+1} = M_D(\boldsymbol{q'}_{i+k}, \hat{\boldsymbol{K}}_{i+k}, \hat{\boldsymbol{V}}_{i+k}) \tag{7}$$

where

$$\hat{\boldsymbol{K}}_{i+k} = (\boldsymbol{K}_{i-1}, \boldsymbol{K}'_{i\sim i+k}) \tag{8}$$

$$\hat{\boldsymbol{V}}_{i+k} = (\boldsymbol{V}_{i-1}, \boldsymbol{V}'_{i\sim i+k}) \tag{9}$$

(2.2) Verify process. The Verify process involves the target model validating the draft tokens. This validation is done by once forwarding on draft tokens of the target model, which computes the true token probabilities.

The target model receives the sequence of draft tokens

$$\boldsymbol{X}'_{i\sim i+\gamma} = (x'_{i+1}, \cdots, x'_{i+\gamma}) \tag{10}$$

and computes the true token probabilities

$$\boldsymbol{P}_{i\sim i+\gamma} = (\boldsymbol{p}_{i+1}, \cdots, \boldsymbol{p}_{i+\gamma}) \tag{11}$$

where

$$\boldsymbol{p}_{i+k+1} = M_T(x_{i+k}, \boldsymbol{K}_{i+k-1}, \boldsymbol{V}_{i+k-1}) \tag{12}$$

Then get true tokens

$$\boldsymbol{X}_{i\sim\gamma} = (x_i, \cdots, x_{i+\lambda}) \tag{13}$$

and additional next token $x_{i+\gamma+1}$, by strictly match when doing greed decoding or speculative sampling when doing sampling decoding.

The process is recursively repeated until the entire generation is complete.

C. Aligned Training by Target-Draft Attention

The method outlined above offers improvements in inference efficiency but introduces challenges in training. A critical issue is the discrepancy between the training and inference phases. If we rely solely on the target model's KV Cache during training, a gap emerges during inference because the draft model's KV Cache also plays a role. Utilizing both the target and draft model's KV Caches during training is not straightforward. A naive approach would be to directly incorporate the target KV Cache into the training process, replacing the last KV with that from the draft model. However, due to the mask attention's matrix operations, this approach only accounts for the generation of the last draft token and fails to consider the preceding tokens. To address this, we designed Target-Draft Attention to simulate the actual inference scenario more accurately. To implement Target-Draft Attention, we modify the training procedure to blend the KV Caches from both the target and draft models, ensuring that

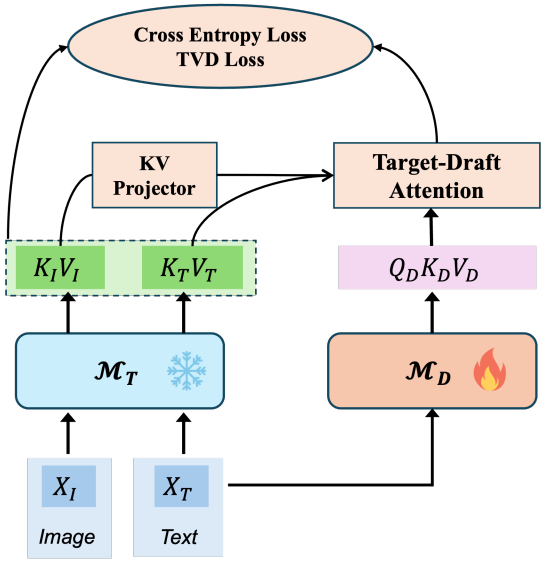


Fig. 2. Overview of training phases. Only a small number of parameters need to be trained.

the draft model learns to align with the target model's context while still leveraging its own contributions. Figure 2 shows our training framework. First let's recall the draft process of inference procedure. Assuming that now it will generate the *k*-th draft token

$$x'_{i+k+1} = M_D(\boldsymbol{q}'_{i+k}, \hat{\boldsymbol{K}}_{i+k}, \hat{\boldsymbol{V}}_{i+k}) \tag{14}$$

which involves the calculation of attention weights

$$\hat{\boldsymbol{a}}_{i+k} = (\boldsymbol{q'}_{i+k}\hat{\boldsymbol{K}}^T_{i+k}) \tag{15}$$

and subsequently, the cattention output

$$\hat{\boldsymbol{o}}_{i+k} = \hat{\boldsymbol{a}}_{i+k}\hat{\boldsymbol{V}}_{i+k} = (\boldsymbol{q'}_{i+k}\hat{\boldsymbol{K}}^T_{i+k})\hat{\boldsymbol{V}}_{i+k} \tag{16}$$

In training, we have the inputs

$$\boldsymbol{K}_n, \boldsymbol{V}_n, \boldsymbol{Q'}_n, \boldsymbol{K'}_n, \boldsymbol{V'}_n.$$

For $\gamma = 1$, a natural idea to do this is to replace $\boldsymbol{k}_n, \boldsymbol{v}_n$ in $\boldsymbol{K}_n, \boldsymbol{V}_n$ with $\boldsymbol{k'}_n, \boldsymbol{v'}_n$ from the draft model, but it yields $(\boldsymbol{Q'}_n\hat{\boldsymbol{K}}^T_n)\hat{\boldsymbol{V}}_n$, where $\boldsymbol{Q'}_n\hat{\boldsymbol{K}}^T_n = [\boldsymbol{Q'}_{n-1}\hat{\boldsymbol{K}}^T_n, \boldsymbol{q'}_n\hat{\boldsymbol{K}}^T_n]$ After masking, the result becomes $[\boldsymbol{Q'}_{n-1}\boldsymbol{K}^T_{n-1}, \boldsymbol{q'}_n\hat{\boldsymbol{K}}^T_n]$, where only the last vector $\boldsymbol{q'}_n\hat{\boldsymbol{K}}^T_n$ is what we need. To achieve accurate inference reproduction, construction n sets of qKV pairs $\{(\boldsymbol{q'}_i, \hat{\boldsymbol{K}}_i, \hat{\boldsymbol{V}}_i), i = 1, \cdots, n\}$ is necessary, but this approach imposes substantial memory overhead and inefficient resource usage.

So we need to optimize the calculation. From alignment purposes, we want to end up with

$$\hat{\boldsymbol{O}}_n = \{(\boldsymbol{q'}_i\hat{\boldsymbol{K}}^T_i)\hat{\boldsymbol{V}}_i, i = 1, \cdots, n\} \tag{17}$$

when the number of generated tokens is 1. To generate *(k+1)*-th token, we have

$$x'_{i+k+1} = M_D(\boldsymbol{q'}_{i+k}, \hat{\boldsymbol{K}}_{i+k}, \hat{\boldsymbol{V}}_{i+k}) \tag{18}$$

where

$$\hat{\boldsymbol{K}}_{i+k} = (\boldsymbol{K}_{i-1}, \boldsymbol{K'}_{i\sim i+k}) \tag{19}$$

$$\hat{\boldsymbol{V}}_{i+k} = (\boldsymbol{V}_{i-1}, \boldsymbol{V'}_{i\sim i+k}) \tag{20}$$

Because of the $softmax$, we must first compute the attention weights,

$$\begin{aligned}\hat{\boldsymbol{a}}_{i+k} &= (\boldsymbol{q'}_{i+k}\hat{\boldsymbol{K}}_{i+k}^{T}) \\ &= (\boldsymbol{q'}_{i+k}\boldsymbol{K}_{i-1}^{T}, \boldsymbol{q'}_{i+k}\boldsymbol{K'}_{i\sim i+k}^{T})\end{aligned} \tag{21}$$

then compute

$$\begin{aligned}\hat{\boldsymbol{o}}_{i+k} &= \hat{\boldsymbol{a}}_{i+k}\hat{\boldsymbol{V}}_{i+k} \\ &= \hat{\boldsymbol{a}}_{i+k}(\boldsymbol{V}_{i-1}, \boldsymbol{V'}_{i\sim i+k}) \\ &= \hat{\boldsymbol{a}}_{i+k}\boldsymbol{V}_{i-1} + \hat{\boldsymbol{a}}_{i+k}\boldsymbol{V'}_{i\sim i+k}\end{aligned} \tag{22}$$

Here, $\hat{\boldsymbol{A}}_n = \{\hat{\boldsymbol{a}}_{i+k},\ i = 1, \cdots, n-k\}$ is formed by $\boldsymbol{Q'}_n\boldsymbol{K}_n$ and $(\boldsymbol{Q'}_n\boldsymbol{K'}_n)$, while $\hat{\boldsymbol{O}}_n$ is formed by $\boldsymbol{A'}_n\boldsymbol{V}_n$ and $\boldsymbol{A'}_n\boldsymbol{V'}_n$. To avoid redundant calculations, we precompute these matrices and index relevant values to construct the target matrix, which greatly reduced training time and memory overhead.

IV. Experiments

A. Experimental setup

Models, datasets and tasks. We conducted experiments on LLaVA-7B and LLaVA-13B [17], encompassing the common sizes of current mainstream MLLMs. We evaluated our method across multiple tasks including mixed generic tasks (conversation, detailed description, and complex reasoning) on LLaVA-Bench In-the-Wild [17] dataset, Image captioning task on images from **coco** [15] dataset, and chain-of thought (CoT) reasoning on Science QA [21] dataset.

Metrics. Like other work about speculative decoding [2, 13, 32], we assess acceleration effects using the following metrics:

- *Walltime speedup ω:* The actual test speedup ratio relative to vanilla auto-regressive decoding.
- *Acceptance rate α:* The average ratio between the number of accepted tokens and the number of speculative tokens.
- *Block efficiency τ:* The average number of generated tokens per block (or target model forward) for a block size γ (or speculative steps).
- *Decoding speed δ:* The average number of generated tokens per second.

B. Main Results

1) Comparison with Usual Methods: Drawing on common practices from NLP tasks, we selected different small draft models as baselines. Since there are no small models suitable for rapid draft generation included in the LLaVA suite, we trained a series of small models to support our experiments, including finetuned-LLaMA (FT-LLaMA in Table I), distilled-LLaMA (DT-LLaMA), finetuned-LLaVA (FT-LLaVA), distilled-LLaVA (DT-LLaVA). All these small LLaMA model and the language model part of LLaVA is of 112M parameters. Specially, first, following Gagrani et al. (2024), we trained a small LLaMA-2 model from scratch (pretrained on RedPajama-Data-1T-Sample [1], then finetuned [8] on OIG-small-chip2 [10] and OpenAssistant [9], and distilled by seq-level distillationn. Furthermore, we use the small LLaMA model above as the language backbone of the LLaVA model. As shown in Table I, our method exhibits superior performance across several key metrics, significantly surpassing traditional draft models. Overall, our model showed significant walltime speedup on multiple datasets such as LLava bench in the wild, coco caption, and SQA. For instance, with a $\gamma = 3$ configuration, it achieved a speedup of 2.02 (mean of 3 datasets), far exceeding other baseline methods by $45.3\% - 61.6\%$. This indicates our method's efficiency in reducing actual processing time relative to the target model. In terms of acceptance rate, our model also achieved outstanding results across datasets, notably reaching rates of 0.62, clearly outperforming baseline models like finetuned-LLaMA and distilled-LLaVA.

TABLE I
Comparison with usual methods (the mean of different metrics across 3 datasets). Note: FT refers to the finetuned, and DT refers to the distilled.

Target Model	γ	Draft Model	ω	α	τ	δ
LLaVA-7B	3	FT-LLaMA	1.39	0.35	1.93	46.13
		DT-LLaMA	1.33	0.34	1.96	45.00
		FT-LLaVA	1.27	0.28	1.68	40.57
		DT-LLaVA	1.25	0.27	1.69	39.50
		Ours	**2.02**	**0.62**	**2.72**	**63.59**
LLaVA-7B	5	FT-LLaMA	1.37	0.34	2.55	42.77
		DT-LLaMA	1.37	0.34	2.54	43.71
		FT-LLaVA	1.21	0.28	2.22	38.35
		DT-LLaVA	1.21	0.28	2.21	38.34
		Ours	**2.06**	**0.62**	**3.92**	**65.02**
LLaVA-13B	3	FT-LLaMA	1.46	0.35	1.89	46.06
		DT-LLaMA	1.44	0.34	1.87	45.20
		FT-LLaVA	1.36	0.30	1.75	42.46
		DT-LLaVA	1.35	0.29	1.71	41.83
		Ours	**2.14**	**0.63**	**2.74**	**67.78**
LLaVA-13B	5	FT-LLaMA	1.44	0.35	2.60	45.29
		DT-LLaMA	1.44	0.35	2.61	45.66
		FT-LLaVA	1.32	0.30	2.35	42.20
		DT-LLaVA	1.31	0.29	2.37	41.64
		Ours	**2.24**	**0.62**	**3.99**	**70.45**

Our approach also excels in block efficiency, with the value up to 2.72, demonstrating robust capability in generating tokens. In decoding speed, our method stands out with speed reaching up to 64 token/s, exceeding the baseline by $37.8\% - 61.0\%$, indicating its suitability for real-time applications where quick response is crucial. In summary, the data in Table I convincingly demonstrates that our method, which integrates the strengths of enhanced speculative decoding and visual processing techniques, outperforming conventional methods across multiple performance metrics, thereby constituting an effective strategy for accelerating the inference of MLLMs.

C. Ablation study

1) Effectiveness of Target Model's KV Cache: To evaluate the impact of using the KV Cache from the target model, we conducted experiments comparing the inference performance with and without using the target model's cache. KV Cache provides valuable contextual information accumulated in previous steps, which can be leveraged by the draft model in speculative decoding. By accessing the cached information directly from the target model, the draft model can achieve better alignment with the target's state, thus enhancing the acceptance rate and speedup.

As shown in Figure 3, our experimental results indicate that using the target model's KV Cache leads to a significant im-

TABLE II
ABLATION ON VISION PROJECTOR (THE MEAN OF DIFFERENT METRICS ACROSS 3 DATASETS).

Target Model	γ	Vsion Projector	ω	α	τ	δ
LLaVA-7B	3	w/o	1.64	0.49	2.33	51.48
		w/	2.02$^{\uparrow}$	0.62$^{\uparrow}$	2.72$^{\uparrow}$	63.59$^{\uparrow}$
LLaVA-7B	5	w/o	1.56	0.47	3.21	48.98
		w/	2.06$^{\uparrow}$	0.62$^{\uparrow}$	3.92$^{\uparrow}$	65.02$^{\uparrow}$
LLaVA-13B	3	w/o	1.72	0.49	2.30	54.27
		w/	2.14$^{\uparrow}$	0.63$^{\uparrow}$	2.74$^{\uparrow}$	67.78$^{\uparrow}$
LLaVA-13B	5	w/o	1.70	0.48	3.26	53.69
		w/	2.24$^{\uparrow}$	0.62$^{\uparrow}$	3.99$^{\uparrow}$	70.45$^{\uparrow}$

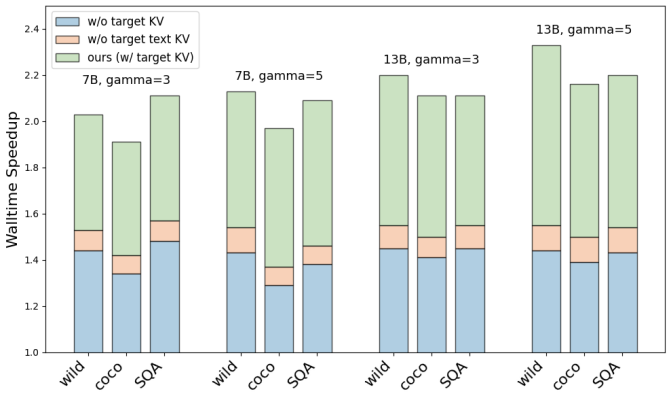


Fig. 3. Ablation on target model's KV Cache.

provement in walltime speedup. Specifically, KV Cache of the target model is directly related to the final output results, enabling the draft model to make more accurate predictions. This reuse of context not only improves the overall coherence of generated sequences but also speeds up the speculative decoding process. In contrast, models without access to the KV Cache are prone to deviate from the target distribution, resulting in a lower acceptance rate and slower inference. Therefore, KV Cache from the target model plays a crucial role in improving the efficiency of speculative decoding in multimodal large language models.

2) Effectiveness of Vision KV Projector: To evaluate the impact of the Vision KV Projector module, we conducted experiments that compared the model's performance with and without this module. The Vision KV Projector is designed to selectively compress and extract relevant vision key-value pairs, enabling the model to efficiently leverage visual context in speculative decoding. By employing Vision KV Projector, our method enables the draft model to align more closely with the target model on visual tasks, thereby enhancing the effectiveness of speculative decoding.

As shown in Table II, our experiments indicate that Vision KV Projector significantly improves both acceptance rate and walltime speedup. With Vision KV Projector enabled, the model demonstrates faster decoding, particularly in tasks with high visual complexity, as irrelevant visual data is filtered out. Additionally, we observed that the Vision KV Projector helps maintain coherence in multimodal contexts by preserving only the most relevant

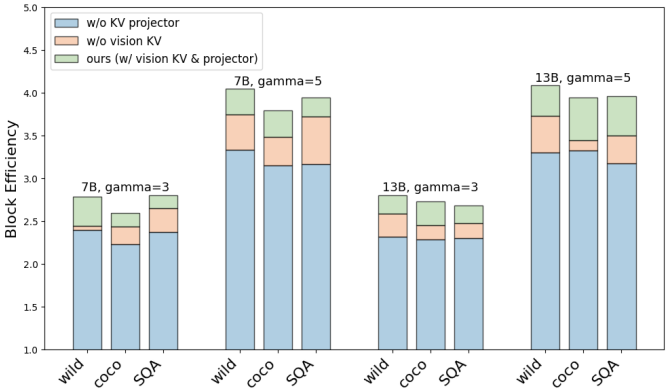


Fig. 4. Ablation on vision KV Cache.

visual cues, allowing the model to align visual information more effectively with textual context. In contrast, without this module, the model processes a larger volume of visual data, resulting in slower decoding and reduced alignment between the draft model and the target model. In summary, Vision KV Projector enhances the efficiency of speculative decoding by focusing on essential visual elements. This approach not only optimizes resource usage but also improves the alignment between the draft model and the target model in multimodal tasks.

3) Do Vision Information Really Important?: To investigate the relative importance of vision and text information in the KV cache, we conducted experiments by selectively disabling either the image KV cache or the text KV cache. This comparison allows us to analyze the impact of each modality on the performance of the draft model during speculative decoding.

As shown in Figure 4, our experimental results reveal that disabling the text KV cache has a more substantial negative effect on block efficiency compared to disabling the image KV cache. Specifically, without the text KV cache, the model experiences a significant drop in block efficiency. This is because text information provides a critical sequential context that is essential for accurate token prediction and verification. In contrast, while image information contributes to the overall context, it does not require the same level of sequential dependency as text, making it relatively less crucial for maintaining the flow of decoding. In summary, text information plays a important role, but uncompressed visual information may impair the performance of the draft model. This finding emphasizes that in speculative decoding of multimodal large models, textual information is a necessity, while visual information is a bonus that needs to be effectively utilized to be effective.

V. CONCLUSION

In this study, we introduced an innovative approach to accelerate the inference processes of MLLMs, focusing on the synergistic integration and information reuse of image and text modalities within MLLMs. This method makes efficient use of KV Cache and aligns the draft model and the target model for speculative decoding in multimodal scenario, greatly increasing the inference speed of MLLMs. Our experimental results demonstrate that this approach can achieve up to a 2x speedup across a variety of multimodal tasks, from visual question answering to complex reasoning, fully validating the effectiveness and potential of our method.

References

[1] Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023.

[2] Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. Glide with a cape: A low-hassle method to accelerate speculative decoding. *arXiv preprint arXiv:2402.02082*, 2024.

[3] Mukul Gagrani, Raghavv Goel, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. On speculative decoding for multimodal large language models. *arXiv preprint arXiv:2404.08856*, 2024.

[4] Ziyuan Huang, Kaixiang Ji, Biao Gong, Zhiwu Qing, Qinglong Zhang, Kecheng Zheng, Jian Wang, Jingdong Chen, and Ming Yang. Accelerating pre-training of multimodal llms via chain-of-sight. *arXiv preprint arXiv:2407.15819*, 2024.

[5] Yizhang Jin, Jian Li, Yexin Liu, Tianjun Gu, Kai Wu, Zhengkai Jiang, Muyang He, Bo Zhao, Xin Tan, Zhenye Gan, et al. Efficient multimodal large language models: A survey. *arXiv preprint arXiv:2405.10739*, 2024.

[6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[7] Mahsa Khoshnoodi, Vinija Jain, Mingye Gao, Malavika Srikanth, and Aman Chadha. A comprehensive survey of accelerated generation techniques in large language models. *arXiv preprint arXiv:2405.13019*, 2024.

[8] Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, 2016.

[9] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

[10] LAION.ai. Oig-small-chip2, 2023.

[11] Phuoc-Hoan Charles Le and Xinlin Li. Binaryvit: pushing binary vision transformers towards convolutional models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4664–4673, 2023.

[12] Yejin Lee, Anna Sun, Basil Hosmer, Bilge Acun, Can Balioglu, Changhan Wang, Charles David Hernandez, Christian Puhrsch, Daniel Haziza, Driss Guessous, et al. Characterizing and efficiently accelerating multimodal generation model inference. *arXiv preprint arXiv:2410.00215*, 2024.

[13] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[14] Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Jinfa Huang, Junwu Zhang, Yatian Pang, Munan Ning, et al. Moe-llava: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*, 2024.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[16] Zhihang Lin, Mingbao Lin, Luxi Lin, and Rongrong Ji. Boosting multimodal large language models with visual tokens withdrawal for rapid inference. *arXiv preprint arXiv:2405.05803*, 2024.

[17] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[18] Yijiang Liu, Huanrui Yang, Zhen Dong, Kurt Keutzer, Li Du, and Shanghang Zhang. Noisyquant: Noisy bias-enhanced post-training activation quantization for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20321–20330, 2023.

[19] Zuyan Liu, Benlin Liu, Jiahui Wang, Yuhao Dong, Guangyi Chen, Yongming Rao, Ranjay Krishna, and Jiwen Lu. Efficient inference of vision instruction-following models with elastic cache. *arXiv preprint arXiv:2407.18121*, 2024.

[20] Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, et al. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*, 2024.

[21] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

[22] Yanyuan Qiao, Zheng Yu, Longteng Guo, Sihan Chen, Zijia Zhao, Mingzhen Sun, Qi Wu, and Jing Liu. Vl-mamba: Exploring state space models for multimodal learning. *arXiv preprint arXiv:2403.13600*, 2024.

[23] Sucheng Ren, Zhengqi Gao, Tianyu Hua, Zihui Xue, Yonglong Tian, Shengfeng He, and Hang Zhao. Co-advise: Cross inductive bias distillation. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 16773–16782, 2022.

[24] swordlidev. Karmavlm, 2024.

[25] Ao Wang, Hui Chen, Zijia Lin, Sicheng Zhao, Jungong Han, and Guiguang Ding. Cait: Triple-win compression towards high accuracy, fast inference, and favorable transferability for vits. *arXiv preprint arXiv:2309.15755*, 2023.

[26] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[27] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[28] Yuxin Wen, Qingqing Cao, Qichen Fu, Sachin Mehta, and Mahyar Najibi. Efficient vision-language models by summarizing visual tokens into compact registers. *arXiv preprint arXiv:2410.14072*, 2024.

[29] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*, 2024.

[30] Ruyi Xu, Yuan Yao, Zonghao Guo, Junbo Cui, Zanlin Ni, Chunjiang Ge, Tat-Seng Chua, Zhiyuan Liu, Maosong Sun, and Gao Huang. Llava-uhd: an lmm perceiving any aspect ratio and high-resolution images. *arXiv preprint arXiv:2403.11703*, 2024.

[31] Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*, 2024.

[32] Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*, 2024.

[33] Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*, 2024.

[34] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*, 2023.

[35] Gaotong Yu, Yi Chen, and Jian Xu. Balancing performance and efficiency: A multimodal large language model pruning method based image text interaction. *arXiv preprint arXiv:2409.01162*, 2024.

[36] Chen Zhang, Zhuorui Liu, and Dawei Song. Beyond the speculative game: A survey of speculative execution in large language models. *arXiv preprint arXiv:2404.14897*, 2024.

[37] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

ALLMod: Exploring Area-Efficiency of LUT-based Large Number Modular Reduction via Hybrid Workloads

Fangxin Liu[1,2,†], Haomin Li[1,2,†], Zongwu Wang[1,2], Bo Zhang[3], Mingzhe Zhang[3], Shoumeng Yan[3], Li Jiang[1,2,*], and Haibing Guan[1]

1. Shanghai Jiao Tong University, 2. Shanghai Qi Zhi Institute, 3. Ant Group

*Corresponding Author {liufangxin, haominli, ljiang_cs}@sjtu.edu.cn

***Abstract*—Modular arithmetic, particularly modular reduction, is widely used in cryptographic applications such as homomorphic encryption (HE) and zero-knowledge proofs (ZKP). High-bit-width operations are crucial for enhancing security; however, they are computationally intensive due to the large number of modular operations required. The lookup-table-based (LUT-based) approach, a "space-for-time" technique, reduces computational load by segmenting the input number into smaller bit groups, pre-computing modular reduction results for each segment, and storing these results in LUTs. While effective, this method incurs significant hardware overhead due to extensive LUT usage. In this paper, we introduce ALLMod, a novel approach that improves the area efficiency of LUT-based large-number modular reduction by employing hybrid workloads. Inspired by the iterative method, ALLMod splits the bit groups into two distinct workloads, achieving lower area costs without compromising throughput. We first develop a template to facilitate workload splitting and ensure balanced distribution. Then, we conduct design space exploration to evaluate the optimal timing for fusing workload results, enabling us to identify the most efficient design under specific constraints. Extensive evaluations show that ALLMod achieves up to $1.65\times$ and $3\times$ improvements in area efficiency over conventional LUT-based methods for bit-widths of 128 and $8,192$, respectively.**

I. Introduction

Privacy computing, a crucial approach for safeguarding data security, has gained considerable attention in recent years due to increasing concerns about privacy and the protection of personal data on the Internet. This field encompasses a variety of applications, including homomorphic encryption (HE) [1], which protects the privacy of user data and models, and zero-knowledge proofs (ZKP) [2], [3], which ensure user privacy during transactions. These applications rely on public-key cryptography algorithms, such as Elliptic Curve Cryptography (ECC) [4] and RSA [5], which use modular reduction as a fundamental component to prevent overflow.

The modular reduction used in these algorithms often requires extremely high bit-widths. For instance, ECC demands a bit-width of at least 224 bits [6], while RSA typically requires at least $1,024$ bits [7]. Moreover, for most cryptographic schemes, higher bit-widths correspond to enhanced security [8]. Such high-bit-width operations pose challenges for efficiently deploying these algorithms and their applications on hardware. Common algorithms, like Barrett [9] and Montgomery [10], implement modular operations through multiplication. However, as bit-width increases, constructing high-performance multipliers to meet latency and throughput requirements becomes increasingly difficult.

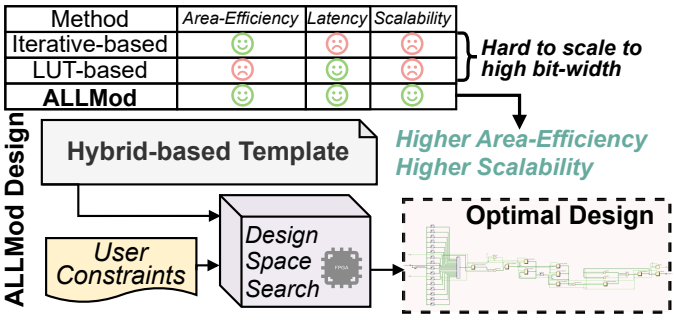


Fig. 1. Comparison of ALLMod to the existing modular reduction methods and ALLMod design overview.

Recently, lookup-table-based (LUT-based) methods [11], [12] have been proposed to achieve low-latency and high-throughput large-number modular reduction. This approach divides input bits into segments and pre-computes possible modular reduction results for each segment. These results are organized into lookup tables (LUTs), which can be implemented using block RAMs (BRAMs) on FPGAs. For each segment, the corresponding modular result is retrieved from the LUTs and combined through an adder tree to yield the final modular reduction result.

This approach converts complex modular reduction operations into simple lookup operations within LUTs, significantly reducing operational complexity and enabling high-throughput, low-latency large-number modular reduction. As a "space-for-time" trade-off, it utilizes BRAMs to store the LUTs and DSPs to implement adder trees that aggregate lookup results. However, as bit-width increases, this method

† These authors contributed equally. This work was partially supported by the National Key Research and Development Program of China (2024YFE0204300), National Natural Science Foundation of China (Grant No.62402311), and Natural Science Foundation of Shanghai (Grant No.24ZR1433700). This work was also supported by Ant Group through CCF-Ant Research Fund (CCF-AFSG RF20240304).

faces the challenge of excessive resource overhead, resulting in suboptimal area efficiency. On the other hand, the iterative method, though straightforward, requires only a minimal number of subtractors but exhibits high latency.

To address these challenges, we propose combining iterative and LUT-based methods to enhance area efficiency in modular reduction implementations. Our solution, ALLMod, reduces area overhead by partitioning input bits into distinct workloads and distributing these across different modular modules, maintaining high throughput. To facilitate workload partition and achieve balanced workload distribution, we design a template that guides implementation. Building on this, we introduce a design space exploration method to identify the optimal design under specified constraints. Experimental results show that ALLMod improves area efficiency by up to 3× compared to the conventional LUT-based method.

In summary, our contributions are as follows:

- We present a novel method to improve the area efficiency of LUT-based large-number modular reduction by smartly distributing the workload, enabling the integration of the iterative method with the LUT-based approach.
- We design a flexible template to facilitate workload division for implementations of varying bit-widths. This template, informed by latency analysis of both methods, ensures balanced workloads while minimizing area usage.
- Using such a template, we propose a design space exploration approach focused on achieving high area efficiency. This approach includes a resource evaluator and an iterative search algorithm, enabling the identification of Pareto-optimal designs under specified constraints, such as latency and area requirements.

II. Background

A. Large-Number Modular Reduction Methods

In cryptographic algorithms, large-number modular reduction can be expressed as $R = A \ mod \ M$, where A is a $2n$-bit number and M is a fixed n-bit number.

1) LUT-based Method: The LUT-based method [11], [12] employs lookup tables to store precomputed modular reduction results for possible segments of A, effectively reducing the computational load during runtime at the cost of additional space overhead. To leverage LUTs, the modular reduction operation must be transformed as follows:

$$\begin{aligned} R = A \ mod \ M &= \sum_{i=0}^{2n-1} (2^i a_i) \ mod \ M \\ &= (\sum_{i=n}^{2n-1} (2^i a_i) + \sum_{i=0}^{n-1} (2^i a_i)) \ mod \ M \\ &= ((\sum_{i=0}^{\frac{n}{k}-1} \sum_{j=0}^{k-1} (2^{n+ki+j} a_{n+ki+j})) + \sum_{i=0}^{n-1} (2^i a_i)) \ mod \ M \\ &= (\sum_{i=0}^{\frac{n}{k}-1} ((\hat{a}_i \times 2^{n+ki}) \ mod \ M) + \sum_{i=0}^{n-1} (2^i a_i)) \ mod \ M \end{aligned} \tag{1}$$

Algorithm 1: LUT-based Modular Reduction [12].

Data: $2n$-bit number $A = \sum_{i=n}^{2n-1}(2^i a_i)$, n-bit modulus M, and LUT's input bit-width k
Result: $R = A \ mod \ M$

Offline:
1 **for** $i \leftarrow 0, 1, \ldots, \lceil n/k \rceil - 1$ **do**
2 **for** $\hat{a}_i \leftarrow 0, 1, \ldots, 2^k - 1$ **do**
3 LUT$[i][\hat{a}_i] \leftarrow (\hat{a}_i \times 2^{n+ki}) \ mod \ M$ **Precompute**
4 **end**
5 **end**

Online:
6 $results \leftarrow []$
7 **for** $i \leftarrow 0, 1, \ldots, \lceil n/k \rceil - 1$ **parallel do**
8 $\hat{a}_i \leftarrow \sum_{j=0}^{k-1}(2^j a_{8i+j})$
9 $results[i] \leftarrow$ LUT$[i][\hat{a}_i]$ **Parallel Look up**
10 **end**
11 $R_0 \leftarrow$ **Add**$(results, \sum_{i=0}^{n-1}(2^i a_i))$ **Adder Tree**
12 $results \leftarrow []$
13 $n_r \leftarrow bitwidth(R_0)$
14 **for** $i \leftarrow 0, 1, \ldots, \lceil (n_r - n)/k \rceil - 1$ **parallel do**
15 $\hat{r}_i \leftarrow \sum_{j=0}^{k-1}(2^j R_0[ki + j])$
16 $results[i] \leftarrow$ LUT$[i][\hat{r}_i]$ **Parallel Look up**
17 **end**
18 $R_1 \leftarrow$ **Add**$(results, \sum_{i=0}^{n-1}(2^i R_0[i]))$ **Adder (Tree)**
19 $R \leftarrow R_1 \ or \ R_1 - M \ or \ R_1 - 2M$ **Subtract**

where $\hat{a}_i = \sum_{j=0}^{k}(2^j a_{n+8i+j})$, i.e. $\hat{a}_i$ is a k-bit number. The $\frac{n}{k}$ red items can be pre-computed offline and stored using LUTs, as the modulus M is typically fixed. Specifically, we use LUTs to pre-compute and store the values of $\hat{a}_i \times 2^{n+ki} \ mod \ M$. A k-input and n-output LUT is used to store all the possible values of $\hat{a}_i \times 2^{n+ki} \ mod \ M$. For the $(n = 128)$ case, an FPGA's BRAM with $36k$-bit capacity can be configured as an 8-bit input and 128-bit output LUT and 16 BRAMs are required in total. The retrieved results from LUTs and the blue item are added together using a $\frac{n}{k} + 1$-input adder tree to get an $(n + log_2 d)$-bit number, where $d = \frac{n}{k}$. The high $log_2 d$ bits of the sum are then input to the LUTs for an additional lookup, and the result is added to the low n bits. Finally, several subtractions are performed for adjustment.

Algorithm 1 outlines the detailed process of the LUT-based modular reduction. The high n-bits are divided to d segments, denoted as $\hat{a}_i$s, each containing k bits. Initially, each segment $\hat{a}_i$ is sent to the corresponding BRAM-based LUT to look up the pre-computed modular result of $\hat{a}_i \times 2^{n+ki} \ mod \ M$. Then, the d n-bit modular results, along with the low n-bit $\sum_{i=0}^{n-1}(2^i a_i)$, are added together to obtain a $n + log_2 d$-bit number R_0. The high $log_2 d$ bits of R_0 are then sent to the LUTs again, and the looked-up results are added with the low

n bits for R_0 to obtain R_1. The second round of lookups helps refine R_1 to the final result R, assisted by several subtraction operations for adjustment.

2) Iterative-based Method: The iterative-based modular reduction method [13] is much more straightforward: the modular operation is performed by repeatedly subtracting the modulus from the input number until the result is smaller than the modulus. To optimize the process, shift operations are employed during each iteration to facilitate efficient subtractions. It is worth noting that because the lower n-bits of the aligned modulus M are zeros, the subtractor only needs to handle the subtraction of up to n-bit numbers.

B. FPGA Implementation for Modular Reduction

Modular reduction implementations are widely deployed on FPGAs due to their high-performance parallel computing capabilities and the flexibility they offer for designing custom solutions tailored to specific needs. These implementations are primarily divided into two categories: multiplier-based methods and LUT-based methods.

Notable examples of multiplier-based methods include Barrett's algorithm [9] and Montgomery's algorithm [10], which rely on multiple high-performance high-bit-width multipliers to achieve high throughput and low latency. However, as bit-width increases, the complexity of designing these multipliers grows significantly, making it challenging to meet throughput requirements.

On the other hand, LUT-based methods [11], [12], as discussed earlier, are better suited to support high throughput in large-number modular reduction. However, with increasing bit-width, this approach faces significant resource overhead, resulting in poor area efficiency. LUTMR [14] proposes using the FPGA's native LUTs instead of BRAMs for implementing lookup tables, aiming to reduce latency and improve LUT usage by reusing patterns. While this optimization helps at the implementation level, it does not address the inherent limitations of the method itself, resulting in only modest improvements (no more than 5% LUT savings).

III. Analysis and Motivation

We conduct an in-depth analysis of the LUT-based and iterative methods for large-number modular reduction. Based on our profiling, we highlight key insights as follows:

Observation: Trade-offs between area-efficiency and latency. We analyze the area efficiency and latency of both methods across various bit-widths, as shown in Figure 2.[1] The LUT-based method offers lower latency but suffers from higher area overhead, while the iterative method exhibits better area efficiency at the cost of higher latency, especially for large bit-widths. In the LUT-based approach, lookup tables reduce online computation, thus minimizing latency. However, the area overhead increases significantly with larger bit-widths, as more BRAMs are needed to store the LUTs, and larger adder trees are required for result aggregation. Moreover, as

[1]The overhead of BRAMs, adders, and subtractors is converted into LUTs, based on the utilization report from Vivado's IP generator.

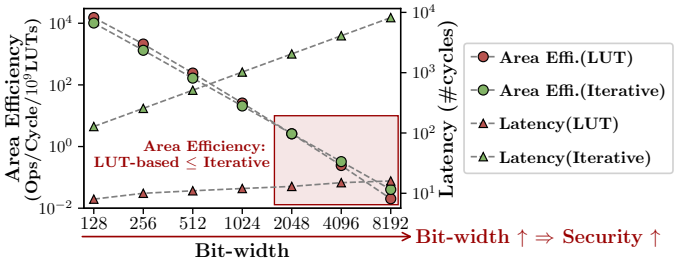


Fig. 2. Comparison of area efficiency and latency between LUT-based method and iterative-based method.

BRAM capacity is fixed, a higher bit-width limits the number of mappings a BRAM can store, increasing the number of result aggregations needed.

For example, with $n = 128$, a $36k$-bit BRAM can store an 8-bit input and 128-bit output LUT, requiring 16 BRAMs and a 16-input, 128-bit adder tree. However, for $n = 2,048$, a BRAM can only store a 4-bit input and $2,048$-bit output LUT, requiring 512 BRAMs and an adder tree with 512 inputs, where each adder must support at least $2,048$-bit addition.

Key Insight: Achieving Pareto Optimal Design through Method Fusion. From the previous observation, the LUT-based method offers low latency but suffers from high area overhead, while the iterative method provides better area efficiency at the cost of higher latency. Therefore, it is feasible to balance area efficiency and latency by distributing the workload in a way that explores the Pareto optimal design space. Existing methods have inherent trade-offs between area efficiency and latency, with no systematic approach to achieving both high area efficiency and low latency. This gap motivates our investigation into fusing the two methods to optimize modular reduction.

By leveraging the unique advantages of each method, we aim to reduce area consumption while maintaining efficient large-number modular reduction. In this paper, we analyze hybrid workloads across both methods to achieve enhanced performance and use this analysis to design a template for balanced workload distribution. We also explore the design space, considering area overhead and latency as flexible constraints, and identify the Pareto optimal boundary of our design. Finally, we propose an automatic search method to efficiently find the optimal design that aligns with user-specified constraints.

IV. Proposed ALLMod Design

A. Method Fusion based on Hybrid Workloads

For formal modeling, we assume that a BRAM can store mappings from a k-bit input to an n-bit output. We use TP, the number of modular reductions per cycle, to indicate the throughput of the design. The upper bound of TP is $MaxTP = 0.5Ops/cycle$, because each modular reduction requires 2 cycles for the lookup in the BRAMs. For a $2n$-bit input A, we divide it into two parts: the high $n - m$ bits are processed using the LUT-based method, while the low $n + m$ bits are handled by the iterative-based method.

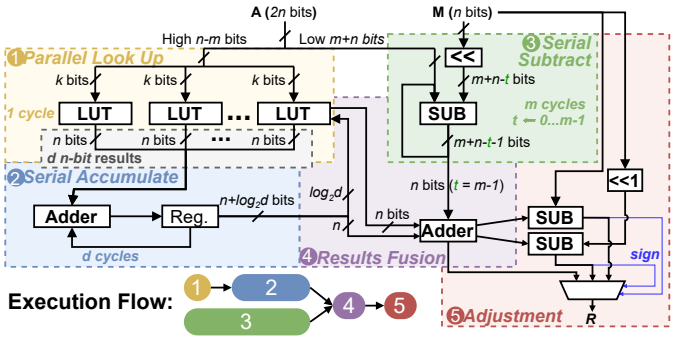


Fig. 3. ALLMod Template for balanced workload. Part ① and part ② support lookup and accumulation for LUT-based method. Part ③ supports serial subtraction for iterative-based method. Part ④ and part ⑤ are designed for fusing and adjusting the results.

Here, m is a design parameter that governs the workload distribution between the two methods. The specific derivation of this partitioning is as follows:

$$\begin{aligned} A &= \sum_{i=0}^{2n-1}(2^i a_i) = \sum_{i=n+m}^{2n-1}(2^i a_i) + \sum_{i=0}^{n+m-1}(2^i a_i) \\ &= (\sum_{i=0}^{\frac{n-m}{k}-1}\sum_{j=0}^{k-1}(2^{n+m+ki+j}a_{n+m+ki+j})) + \sum_{i=0}^{n+m-1}(2^i a_i) \\ &= \sum_{i=0}^{\frac{n-m}{k}-1}(\hat{a}_i \times 2^{n+m+ki}) + \sum_{i=0}^{n+m-1}(2^i a_i) \end{aligned} \tag{2}$$

where $\hat{a}_i = \sum_{j=0}^{k-1}(2^j a_{n+m+ik+j})$. Thus, the modular reduction can also be split into two parts of workloads:

$$\underbrace{(\sum_{i=0}^{\frac{n-m}{k}-1}(\hat{a}_i \times 2^{n+m+ki}))\ mod\ M}_{\textbf{Workload I} \text{ for LUT-based method}} + \underbrace{(\sum_{i=0}^{n+m-1}(2^i a_i))\ mod\ M}_{\textbf{Workload II} \text{ for iterative method}} \tag{3}$$

In ALLMod, **workload I** is solved by the LUT-based method, and **workload II** is solved by the iterative-based method. For scalability and area efficiency considerations, we employ serial adders to implement accumulation instead of an adder tree.

B. Template Design for balanced workload

Based on hybrid workloads, we propose a template design to maximize computation efficiency. As illustrated in Figure 3, the template consists of five parts. Part ① are d BRAMs for storing the pre-computed results of $\hat{a}_i \times 2^{n+m+ki}\ mod\ M$, where $d = \frac{n-m}{k}$. The high $n-m$ bits are fed into the BRAMs for parallel lookup. Part ② uses an adder for serially accumulating the d results from the BRAMs. Part ③ consists of a subtractor and a shifter for iterative subtractions. Part ④ is an adder for fusing the results from the two methods, including the low n bits of the accumulation results from part ②, the second round lookup result from part ①, and the serail subtraction result from part ③. Part ⑤ deploys several subtractors and a multiplexer for adjusting the final result.

The dataflow is also illustrated in Figure 3. For the **left workload**, the high $n-m$ bits are sent to part ① for parallel lookup, which takes one cycle. The results are then serially accumulated in part ② over d cycles. After accumulating the $n+\log_2 d$-bit result in part ②, the high $\log_2 d$ bits are sent back to part ① for a second round of lookup. Meanwhile, for the **right workload**, the low $n+m$ bits are sent to part ③, where they undergo $m-1$ cycles of subtraction. In part ④, the results from the previous three parts—part ①, part ②, and part ③—are added together for fusion. The final result is then adjusted using several subtractions in part ⑤. Regarding resource reuse, since the execution of part ② and part ③ does not overlap with that of part ④ and part ⑤, resource reuse is possible. The adder in part ② can be reused in part ④ for aggregation, and the subtractor in part ③ can also be reused in part ⑤ for adjustment. This further optimizes the design by reducing redundant hardware resources and enhancing efficiency.

Given that the two workloads are executed in parallel, the latencies of the two workloads should ideally be equal to ensure a balanced workload and minimize resource waste. The latency of the left workload is $d+1$ cycles ($d = \frac{n-m}{k}$), and the latency of the right workload is m cycles. Therefore, the following equation should be satisfied:

$$\frac{n-m}{k} + 1 = m \ \Rightarrow \ m = \frac{n+k}{k+1} \tag{4}$$

Apart from the BRAMs, the adder and subtractor in the template need to be copied $d \times TP$ times to ensure the throughput.

The template can reduce the area overhead of both BRAMs and adders/subtractors. Compared to the LUT-based method, the template can achieve $\frac{m}{n} = \frac{n+k}{n(k+1)} \approx \frac{1}{k+1}$ BRAM savings. Moreover, the original LUT-based method needs $\frac{2n}{k}$ adders/-subtractors, while ours just needs $d = \frac{n-m}{k} =$ pieces to reach maximum throughput $MaxTP$. For $n = 8,192$, a BRAM can only store mappings with 2-bit input, meaning $k = 2$. Consequently, our template can save up to 50% of the required BRAMs and over 65% of the required adders/subtractors.

C. Template-aided Design Space Exploration

To accommodate a broader range of design requirements, we explore the design space of ALLMod. Using the proposed template, which focuses on balanced workloads, we investigate the design space by considering both latency and area constraints as key parameters.

1) Parallel-Serial Hybrid for More Strict Latency Constraint: In cases where stricter latency requirements are imposed, the single-pass algorithm must be completed in fewer cycles. Since the iterative method cannot be parallelized, we propose reducing its workload while shifting more work to the table lookup method. Additionally, we aim to enhance the parallel processing capacity of specific components within the table lookup method.

To achieve this, we introduce a small adder tree into part ② of the template, which works alongside the serial accumulator to accelerate the accumulation process. This setup allows the

workload in the left part of the design to be completed in fewer cycles, helping to meet the stricter latency constraints. When configured with an adder tree that has x inputs, part ② can be completed in $\frac{n-m}{k} - x$ cycles, while part ③ will still take m cycles. The optimal design point is determined by finding m and x that satisfy the following conditions:

$$max(1 + \frac{n-m}{k} - x, m) \leq Latency_{req} \tag{5}$$

where $Latency_{req}$ is the user-specified number of cycles required for the design.

2) More Strict Area Constraint: When area constraints become more stringent, we assign a larger portion of the workload to part ③ (iterative subtraction) to reduce the BRAM overhead of part ② (parallel lookup). However, since the workload in part ③ cannot be accelerated in parallel, we need to ensure throughput by duplicating more of the components in part ③.

Assuming that part ③ is duplicated y times, its throughput is approximately $\frac{y}{m}$Ops/cycle. To guarantee throughput, we set $y = m \times TP$. Similarly, part ② must be duplicated $d \times TP$ times. Consequently, the task is to determine the value of m such that the design meets both area and throughput requirements:

$$\begin{aligned} &\frac{n-m}{k} \times Area_{BRAM} + d \times TP \times Area_{Adder} \\ &+m \times TP \times Area_{Subtractor} \leq \text{Area}_{req} \end{aligned} \tag{6}$$

where $Area_{req}$ is the user-specified area limit, and $Area_{BRAM/Adder/Subtractor}$ represents the area of a BRAM/adder/subtractor.

3) Automatic Search: The analysis of the design space only focuses on either lower latency or smaller area consumption. In practice, the design usually has constraints on both latency and area. Therefore, based on the previous analysis, we propose an automatic search approach to find the Pareto optimal design schemes that meet various constraints provided by users. As shown in Algorithm 2, our search method takes the bit-width n, and user-specified constraints as input, and outputs a list of Pareto optimal schemes. Users' constraints include latency requirement $Latency_{req}$, area requirement $Area_{req}$, and expected throughput TP.

First, we enumerate m from 0 to n to explore different workload segmentations. Specifically, $m = 0$ corresponds to the pure LUT-based method, while $m = n$ corresponds to the pure iterative method. Next, we enumerate the input width of the adder tree, $Width_{Tree}$, from 0 to $n - m$. When $Width_{Tree} = 0$, the template with a balanced workload, as introduced in Section IV-B, is used; when $Width_{Tree} = n - m$, part ② is entirely implemented using an adder tree for parallel accumulation. Each pair of values (m, $Width_{Tree}$) forms a key *parameter pair* for a candidate scheme.

For each *parameter pair*, we compute the cycles required to complete the modular reduction, as outlined in Section IV-B, and verify whether the latency requirement is met. Then, we calculate the number of BRAMs, adders, and subtractors needed for the scheme and estimate the corresponding area overhead. Feasible schemes that satisfy both latency and area constraints are added to a candidate list, $SchemeList$. Finally, we identify the Pareto optimal schemes from this list. Through such an automatic search process, users can easily identify the Pareto optimal design schemes that meet their specific constraints.

Algorithm 2: Automatic Design Space Search.

Data: Bit-width n, Input bit-width of BRAM(LUT) k, user-specified latency and area constraints $Latency_{req}$ and $Area_{req}$, expected throughput TP

Result: Pareto optimal scheme list $OptSchemeList$

```
SchemeList ← [ ]
for m ← 0 ... n do
    for Width_Tree ← 0 ... n − m do
        Latency_LUT ← 1 + max(n−m/k − Width_Tree, log2(Width_Tree))
        Latency_iter ← m
        Latency ← max (Latency_LUT, Latency_iter )
        if Latency ≤ Latency_req then
            N_BRAMs ← n−m/k
            N_Adders_Tree ← 2 × Width_Tree − 1
            N_Adders_Serial ← (n−m/k − Width_Tree) × TP
            N_Adders ← N_Adder_Tree + N_Adder_Serial
            N_Subtractors ← m × TP
            Area ← AreaEstimate(N_BRAMs, N_Adders, N_Subtractors)
            if Area ≤ Area_req then
                Scheme_feasible ← (m, Width_Tree, Cycles, Area)
                SchemeList.append(Scheme_feasible)
            end
        end
    end
end
OptSchemeList ← FindParetoOptimal(SchemeList)
```

In addition, our approach efficiently identifies Pareto optimal schemes. By enumerating both m and $Width_{Tree}$, the total number of feasible schemes, denoted as S, can be at most n^2. Using a priority queue to sort and select the optimal schemes, the time complexity for finding the Pareto optimal solutions is $O(S \log S) = O(n^2 \log n)$. Even for very large values of n, such as $8,192$, the entire search process completes in approximately ten minutes[2].

V. Evaluations

A. Evaluation Methodology

We compare ALLMod with the standard LUT-based method [12] and iterative method [13]. We implement a 128-bit prototype system for ALLMod on FPGA using Verilog, with a working frequency set to 200MHz. To validate the

[2]On a laptop with Intel i9-12900H CPU and 32GB memory.

TABLE I
COMPARISON OF LUT-BASED METHOD, ITERATIVE METHOD, AND ALLMOD (TEMPLATE), ALL ALIGNED TO $MaxTP$.

Bit-Width (n)	**Area Efficiency** (*Ops/Cycles/*10^9*LUTs*)			**Latency** (*Cycles*)			**Breakdown** (*BRAMs;Adders;Subtractors*)			**Hybrid Workloads**
	LUT-based [12]	Iterative [13]	**ALLMod** (Improve)	LUT-based	Iterative	**ALLMod**	LUT-based	Iterative	**ALLMod**	**ALLMod**
128	15258.79	10172.53	**25040.06** (1.65×)	9	128	20	16;3;1	0;0;64	15;8;8	113:143
256	2111.49	1326.85	**4521.12** (2.14×)	11	256	37	37;73;1	0;0;128	32;16;16	224:288
512	241.60	165.86	**550.18** (2.28×)	12	512	78	86;171;1	0;0;256	73;37;37	438:586
1024	25.75	20.73	**61.05** (2.37×)	13	1024	176	205;409;1	0;0;512	171;86;86	853:1195
2048	2.59	2.59	**6.45** (2.49×)	14	2048	415	512;1024;1	0;0;1024	410;205;205	1638:2458
4096	0.24	0.32	**0.65** (2.71×)	16	4096	1029	1366;2731;1	0;0;2048	1024;512;512	3072:5120
8192	0.02	0.04	**0.06** (**3.00**×)	17	8192	2736	4096;8191;1	0;0;4096	2731;1366;1366	5461:10923

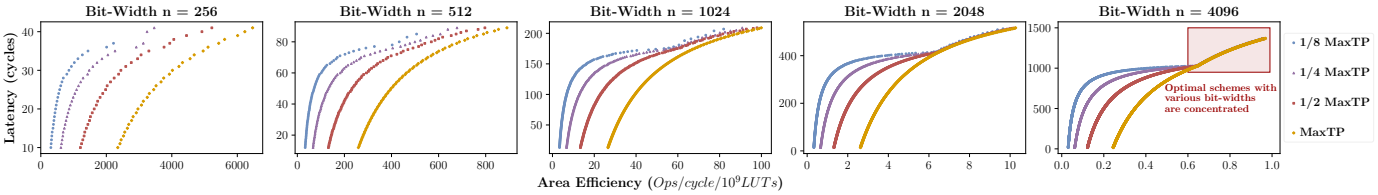


Fig. 4. Pareto optimal schemes of ALLMod for various throughputs and bit-widths. The optimal schemes with high area efficiency gradually converge as the bit-width increases.

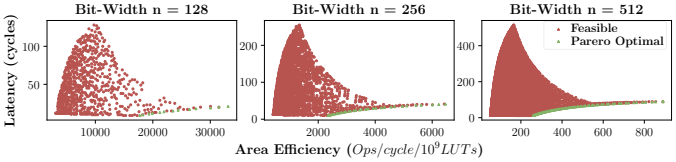


Fig. 5. Visualization of searched schemes for various bit-width at maximum throughput $MaxTP$. **Red** points indicate the feasible schemes and **green** points indicate the Pareto optimal schemes.

timing and functionality of our models, we synthesized them using the Xilinx Vivado Design Suite [15]. For evaluation, we use $\frac{Ops/cycle}{10^9 LUTs}$ as the metric of area efficiency. Given that the working frequency for both the baseline and our method is primarily determined by the latency of the n-bit adder/subtractor, we utilize $cycles$ as a standardized metric for latency comparison, assuming a similar frequency across implementations. This allows for a fair comparison of execution time independent of absolute clock speeds. The overhead of BRAMs, adders, and subtractors is converted with LUTs, based on the utilization report from the IP generator of Vivado.

B. Overall Performance

Table I compares the performance of our ALLMod template with the baseline method. The methods are all aligned to the maximum throughput $MaxTP$ that the LUT-based method can achieve. We show the area efficiency, latency, and the breakdown of resources for each method, and segmentation for hybrid workloads of ALLMod.

For small bit-width $n = 128$, ALLMod achieves the area efficiency of $25,000\ Ops/Cycles/10^9 LUTs$, which is $1.65\times$ higher than the LUT-based method. For large number modular reduction with $n = 8,192$, ALLMod achieves up to $3.0\times$ area efficiency improvement over the LUT-based method. It's worth noting that although the LUT-based method has lower latency, it needs too much area overhead to achieve the same throughput as ALLMod.

The breakdown results demonstrate that ALLMod can save a significant number of BRAMs and adders/subtractors. For $n = 8,192$, ALLMod saves over 30% of BRAMs and 66% of adders/subtractors compared to the LUT-based method. As the bit-width increases, the proportion of workload allocated to the right parts in the template gradually increases, which illustrates the significance of introducing a hybrid workload design for optimizing the LUT-based method.

C. Pareto Optimal with various throughput requirements

Figure 4 shows the Pareto optimal schemes of ALLMod for various throughputs and bit-widths. We iterate the throughput requirement from $MaxTP/8$ to $MaxTP$ for each bit-width. As the bit-width increases, the Pareto optimal schemes for different throughputs are more concentrated in the region with high area efficiency, meaning that the design gradually converges. Such a convergence demonstrates that ALLMod can effectively explore the design space and find optimal schemes for guiding the design.

D. Design space exploration with automatic search

Figure 5 visualizes the design space exploration results of ALLMod for various bit-widths. The distributions of feasible schemes under different bit-widths are basically the same. A significant concentration of feasible schemes is observed within the region of lower area efficiency and high latency, whereas a small number of feasible solutions are identified in the region of higher area efficiency. This distribution underscores the criticality of identifying Pareto optimal schemes. Our automatic search effectively filters out schemes with low area efficiency and high latency and finds area-efficient schemes with low latency.

VI. CONCLUSION

This paper presents ALLMod, an area-efficient method for large-number modular reduction, combining LUT-based and iterative approaches. We then design a balanced workload template to guide the segmentation. Additionally, we present an automatic search approach to explore the design space of ALLMod based on this template. Evaluations show that ALLMod outperforms the LUT-based method in terms of area efficiency and latency.

REFERENCES

[1] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, 2019, pp. 203–225.

[3] U. Fiege, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 210–217.

[4] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[5] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[6] L. Chen, D. Moody, A. Regenscheid, and A. Robinson, "Digital signature standard (dss)," 2023.

[7] J. Bos, M. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery, "On the security of 1024-bit rsa and 160-bit elliptic curve cryptography," 2009.

[8] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[9] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[10] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[11] B. Devlin, "Blockchain acceleration using fpgas—elliptic curves, zk-snarks, and vdfs," *ZCASH Foundation*, 2019.

[12] E. Öztürk, "Design and implementation of a low-latency modular multiplication algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 6, pp. 1902–1911, 2020.

[13] W. A. S. Wijesinghe, "A high-speed hardware algorithm for modulus operation and its application in prime number calculation," 2024. [Online]. Available: https://arxiv.org/abs/2407.12541

[14] A. Opasatian and M. Ikeda, "Lookup table modular reduction: A low-latency modular reduction for fast ecc processor," in *2023 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2023, pp. 1–6.

[15] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.

G-SpNN: GPU-Accelerated Passivity Enforcement for S-Parameter Modeling with Neural Networks

Lijie Zeng[1], Jiatai Sun[1], Xiao Wu[2], Dan Niu[3], Tianshi Wang,[4], Yibo Lin[5], Zuochang Ye[6], and Zhou Jin[7]
[1]SSSLab, China University of Petroleum-Beijing, China, [2]Huada Empyrean Software Co. Ltd, China,
[3]School of Automation, Southeast University, China, [4]HiSilicon Technologies Co. Ltd
[5]School of Integrated Circuits, Peking University, Beijing, [6]School of Integrated Circuits, Tsinghua University,
[7]College of Integrated Circuits, Zhejiang University, China
Email: z.jin@zju.edu.cn

***Abstract*—The increasing complexity of high-frequency circuits calls for efficient and accurate passive macromodeling techniques. Existing passivity enforcement methods, including those in commercial tools, often encounter convergence issues or compromise accuracy. The Domain-Alternated Optimization (DAO) framework seeks to restore accuracy through an additional optimization step but is hampered by high memory consumption and slow convergence, particularly for large-scale problems. This paper presents G-SpNN, a novel GPU-accelerated framework that recasts the passivity-enforced macromodeling problem as a neural network training task. This approach significantly enhances both the speed and scalability of passivity enforcement. Experimental results show that G-SpNN achieves an average speedup of 7.63× in convergence compared to DAO, while reducing memory usage by two orders of magnitude. This enables G-SpNN to handle complex, high-port-count circuits with greater accuracy and efficiency, paving the way for robust high-frequency circuit simulations.**

***Index Terms*—S-Parameter Macromodeling, Passivity Enforcement, Neural Network, GPU Acceleration**

I. INTRODUCTION

S-parameters, and frequency-domain parameters in general, pose significant challenges in circuit simulation. Running time-domain analyses with them, which typically involves frequency-domain interpolation and extrapolation, causality enforcement, impulse response computation and the convolution analysis, is computationally intensive and often leads to convergence issues. Macromodeling, also known as broadband SPICE (bbspice) in the circuit design community, offers a preferred alternative [1], but its effectiveness hinges on robust passivity enforcement to guarantee accurate and stable high-frequency simulations.

Early attempts to address passivity enforcement through convex optimization faced limitations due to their high computational complexity, scaling with $O(n^6)$ [1], where n represents the number of poles. This restricted their applicability to small-scale problems ($n < 100$) [1]. Mainstream methods typically adopt a two-step approach: first generating a macromodel without considering passivity constraints, often using techniques like Vector Fitting (VF) [2], and then applying specialized algorithms to restore passivity, such as the Eigenvalue Perturbation (EPM) [3], Residue Perturbation (RPM) [4], and Local Compensation (LC) [5]. However, these methods frequently compromise model accuracy, significantly increase computational costs, and are not robust in their performances, making them challenging to implement in practice.

Despite these challenges, almost all commercial solutions rely on these two-step methods. In practice, designers routinely observe commercial broadband SPICE fitting tools struggling to fit S-parameter data from Touchstone files, particularly for the simulations of radio-frequency integrated circuits (RFICs) and chip packaging. The iterative nature of these methods often introduces cumulative fitting errors, leading to reduced accuracy. To mitigate this, commercial tools often relax passivity constraints, either by internally setting a tolerance for violations or enforcing passivity only within limited bandwidths. While this approach may improve fitting success rates, it can hinder simulation convergence later in the design process.

The Domain-Alternated Optimization (DAO) framework [6] seeks to enhance accuracy by incorporating a third optimization step after the conventional two-step approach. By transforming the original modeling problem constrained by passivity criteria into an unconstrained optimization problem, DAO enables the use of unconstrained optimizers to improve macromodel accuracy while maintaining passivity throughout the process. However, DAO still suffers from high memory consumption and slow convergence, particularly when dealing with large-scale problems, limiting its practical adoption. For example, using the DAO method on a 64-port system resulted in an average memory usage of 22GB per iteration, while a 138-port system exceeded 31.8GB.

Therefore, passivity enforcement is arguably still an unsolved problem, especially for large-scale systems with many ports. Today, the increasing prevalence of complex integrated circuits, with a rising number of ports driven by trends like chiplets and multi-core architectures, further exacerbates the challenges of passivity enforcement [7]. This surge in complexity renders existing methods inadequate, necessitating new solutions capable of efficiently and accurately enforcing passivity in large-scale systems.

In the meanwhile, the development of neural networks, combined with the emergence of large-scale optimizers and optimization frameworks for training these networks, presents new possibilities for addressing the challenge of passivity enforcement [8]. These frameworks, which can be accelerated using GPUs, provide a missing piece in the three-step DAO

framework. By drawing an analogy between passivity enforcement and neural network training, specifically the process of finding optimal weights [9] [10], it becomes possible to handle large-scale problems more effectively.

This work proposes G-SpNN, a novel GPU-accelerated framework that recasts passivity-enforced macromodeling as a neural network training problem. This framework, built upon the deep learning toolkit PyTorch, maps the constrained optimization problem of passivity enforcement onto an unconstrained optimization objective within the neural network training process. By leveraging PyTorch's automatic differentiation mechanism, G-SpNN eliminates the need for manual gradient derivation, simplifying the implementation and improving efficiency. Moreover, G-SpNN employs vectorized representations and QR decomposition to streamline error function calculations and utilizes the LBFGS method to approximate the Hessian inverse matrix, thereby accelerating convergence.

G-SpNN's key contributions include:

1) Casting the passive macromodeling problem to neural network training, thus leveraging GPU acceleration.
2) Using the LBFGS method to efficiently approximate the Hessian inverse matrix, proven to work the best in practice.
3) Keeping the memory usage almost constant with an increasing number of ports. Experimental results show that G-SpNN not only converges more stably and quickly than DAO, with a average speedup of 7.63×, its memory usage can be reduced by two orders of magnitude in test cases.

From above, G-SpNN offers a robust passivity-enforced macromodeling framework, presented in more detail in the following sections.

II. PRELIMINARIES

A. Macromodel Generation of S-Parameters

S-parameters describe signal scattering in the frequency domain. Macromodeling fits a rational function to frequency data s_k (frequency points) and $H(s_k) \in \mathbb{C}^{m\times m}$ (system response):

$$f(s_k) = \sum_{n=1}^{N_q} \frac{c_n}{s_k - a_n} + d + s_k h \tag{1}$$

where c_n, a_n, d, h are fitting coefficients.

To improve stability and efficiency, Eq. (1) is often expressed in a state-space form [11]:

$$G(s_k) = C(s_k \cdot I - A)^{-1}B + D \tag{2}$$

where $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}, C \in \mathbb{C}^{m\times n}, D \in \mathbb{C}^{m\times m}$, and I as the identity matrix.

For this system, $G(s_k)$ needs to satisfy two conditions [12]:

1) The error needs to be minimized.

$$Error = \min\left(\sum_k |G(s_k) - H(s_k)|\right) \tag{3}$$

2) It needs to satisfy the passivity requirement [13].

$$G(s_k) + G^H(s_k) \geq 0 \tag{4}$$

where G^H represents the conjugate transpose of G.

B. Three-Step Modeling Framework

The first step in passivity macromodeling is to fit Eq. (1) using methods such as VF [2] or Rational Fitting [14], and then transform it into the state-space representation as shown in Eq. (5), thereby obtaining the initial matrix.

$$G_0(s_k) = C_0(s_k \cdot I - A_0)^{-1}B_0 + D_0 \tag{5}$$

Although this step can easily provide a locally optimal fit with minimal error, the resulting matrix may not necessarily satisfy the required passivity [15].

The second step involves passivity fixing on the initial $G_0(s_k)$ to construct $G_1(s_k)$ by using EPM, RPM and LC, such that $G_1(s_k)$ satisfies the passivity [16].

The third step begins with $G_1(s_k)$ and solves the optimization model shown in Eq. (6) to find a system $G_2(s_k)$ that minimizes the error [17].

$$\begin{cases} \min\limits_{C,D} \sum\limits_k |G_2(s_k) - H(s_k)| \\ \text{subject to: } G_2(s_k) \text{ is passive} \end{cases} \tag{6}$$

C. Domain-Alternated Optimization

Actually, the third step of the Three-Step Modeling Framework often faces challenges in enforcing passivity constraints. Robust implementation of Eq. (6) is impractical [18] [19]. To address this problem, the DAO framework simplifies the optimization process by introducing a system $W(s_k)$ as optimization variables, thereby removing passivity constraints and converting the problem into an unconstrained form [6].

Algorithm 1 DAO Algorithm

1: **Step 1 :** Perform SPF on the initial passive system $G_1(s_k)$ to obtain $W(s_k)$.
2: **Step 2 :** Use the matrices (L, Q) of $W(s_k)$ as parameters and perform unconstrained optimization to minimize error.
3: **Step 3 :** Perform PFE to the optimized $W(s_k)$ to obtain the final system $G(s_k)$.

The Spectral Factorization (SPF) operation in Algorithm 1 is given by Eq. (7)-(10). Its essence lies in computing the matrices L and Q in order to derive $W(s_k)$ as shown in Eq. (11) [20].

$$R = Q^T Q = D + D^T \tag{7}$$

$$\tilde{A} = BR^{-1}C - A, \quad \tilde{B} = BQ^{-1}, \quad \tilde{C} = C^T R^{-1} C \tag{8}$$

$$K\tilde{A} + \tilde{A}^T K - K\tilde{B}\tilde{B}^T K - \tilde{C} = 0 \tag{9}$$

$$L = Q^{-T}C - Q^{-T}B^T K \tag{10}$$

$$W(s_k) = L(s_k \cdot I - A)^{-1}B + Q \tag{11}$$

The Partial Fractional Expansion (PFE) operation, defined by Eq. (12)-(14), focuses on computing the matrices C and D to transform $W(s_k)$ into $G(s_k)$ [21] [22] [23].

$$M = \text{kron}(A^T, I_n) + \text{kron}(I_n, A^T) \tag{12}$$

$$\text{vec}(K) = -M^{-1}\text{vec}(L^T L) \tag{13}$$

$$C = B^T K + Q^T L, \quad D = \frac{1}{2}Q^T Q \tag{14}$$

where kron$(\cdot)$ represents the Kronecker product, and vec(K) represents the vectorized form of matrix K [24].

Based on the principles and theorems related to SPF and PFE [21], the system $G(s_k)$ derived from $W(s_k)$ is guaranteed to maintain passivity. Consequently, the original optimization problem in Eq. (6) can be reformulated as an unconstrained optimization problem in Eq. (15).

$$\begin{cases} \min\limits_{L,Q} \sum\limits_{k} |G(s_k) - H(s_k)| \\ \text{subject to:} \\ W(s_k) = L(s_k \cdot I - A)^{-1} + Q \\ G(s_k) = pfe(W(s_k)) \end{cases} \tag{15}$$

III. G-SpNN

For the optimization problem defined in Eq. (15), it is feasible to directly compute the gradient vector and Hessian matrix [25]. However, compared to Eq. (6), the larger computational scale results in increased time costs and memory consumption, brings significant challenges. To this end, we propose G-SpNN, which achieves an optimal passive macromodel with faster convergence and lower memory consumption. More importantly, it exhibits excellent scalability, capable of handling ultra-large-scale passive macromodeling problems.

The overall workflow of G-SpNN is shown in Figure 1. Starting with a given passive system $G_1(s_k)$, an unconstrained system $W(s_k)$ is first derived by SPF transformation and represented as a layer in the neural network. Next, the PFE transformation is applied to generate a new network layer $G(s_k)$, corresponding to a passive system. This reformulates the problem as a neural network training task. During forward propagation, the system, together with the tabulated data $H(s_k)$, is used to compute the loss value. For efficient training, the LBFGS method is futher incorporated with backpropagation to compute gradients and update the network parameters. Once the iteration stopping criteria are satisfied, the optimized passive system is obtained. Details are in Algorithm 2.

Algorithm 2 Proposed G-SpNN Algorithm

1: **Preprocess:** Calculate R_F, b, δ, M^{-1} from (20), (22) and (12)
2: **SPF Transformation:** Apply SPF to matrices A, B, C, D from (2) to obtain A, B, L, Q
3: **Initialize:** Set optimizer with parameters L, Q, define number of epochs
4: **for** i in {1, 2, ..., epoch} **do**
5: Apply PFE to transform L, Q into C, D
6: Vectorize C, D into y, and L, Q into x
7: Compute loss: $f(x)$ from (21)
8: Backpropagate and using chain rule: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial x}$
9: Approximate the Hessian inverse via k LBFGS iterations
10: Update L, Q
11: **end for**
12: **Final Transformation:** Convert L, Q to final C, D using PFE to obtain final $G(s_k)$

A. Analogy to Neural Network Training

Both solving Eq. (15) and training a neural network represent fundamentally nonlinear optimization challenges. In this context, the error function in Eq. (15) can be viewed as analogous to the prediction error encountered in neural network training, highlighting a direct connection between the two optimization frameworks. This analogy not only emphasizes the shared nature of the problems but also suggests that leveraging neural network techniques can facilitate faster optimization. As illustrated in Figure 2, the comparison between the two problem-solving approaches is visually clear. In neural network training, each data instance consists of an input vector x_i and a corresponding label y_i, and the predicted label $\phi(x_i, w)$ is obtained through the neural network. The training task is to minimize an objective function, which primarily consists of prediction errors.

By drawing an analogy between unconstrained optimization, as expressed in Eq. (15), and neural network training, we represent parameters L, Q in simplified form as w, and treat parameters s_k and $H(s_k)$ (derived from the simulation or measurement process) as data instances. The neural network processes these data instances and computes $g(s_k, w)$ ($g(\cdot, w)$ represents $G(s)$ in Eq. (15), where A and B are constants), after which the corresponding objective function is calculated to iteratively update the network, ultimately yielding the optimal solution to Eq. (15). Through this construction, the unconstrained optimization problem can be seamlessly transformed into the neural network training process, making it possible to leverage advanced deep learning techniques to address the issues in the DAO framework.

It is important to note that, as shown in Figure 1, during the complete equivalent neural network training process, the transformations between L, Q and C, D through PFE and SPF do not affect the gradient computation. The gradient still follows the chain rule and is automatically handled by PyTorch, a feature that the DAO framework lacks.

B. Loss Function

During the forward propagation phase of neural network training, the need to compute prediction errors N times, with each error computed through matrix norm calculations, introduces significant complexity throughout the training process. To address this issue, this section will reformulate the objective function from Eq. (15) to make the neural network training process more computationally efficient.

For the objective function in Eq. (15):

$$Error = \min_{C,D} \sum_{k} |G(s_k) - H(s_k)| \tag{16}$$

We can vectorize it as follows:

$$Error_vec = |Fy - h| \tag{17}$$

where:

$$y = \begin{bmatrix} \text{vec}(C) \\ \text{vec}(D) \end{bmatrix} \qquad \text{F}_i = \begin{bmatrix} \text{kron}([s_i I - A]^{-1}, I_n) & I_m^2 \end{bmatrix} \tag{18}$$

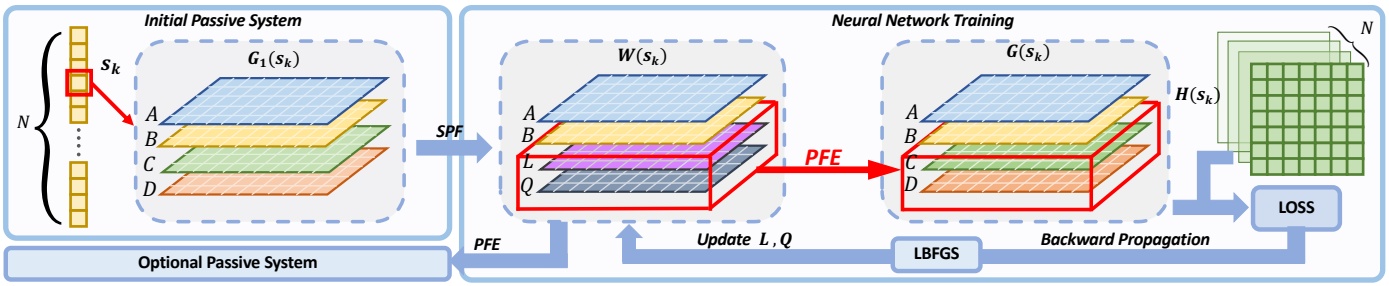


Fig. 1: The overall framework of G-SpNN.

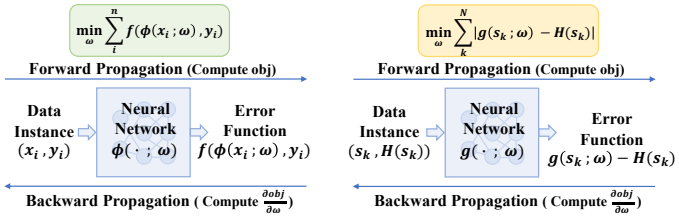


Fig. 2: Analogy between unconstrained optimization and neural network training. (a) Train a network for weights $\mathbf{w}$. (b) Solve a set of parameters for macromodel with $\mathbf{w} = (\mathbf{L}, \mathbf{Q})$.

To further simplify the computation of the objective function, a more detailed decomposition of Eq. (17) can be performed. The specific decomposition process is as follows. Firstly, QR factorization can be applied to F in Eq. (19):

$$h = \begin{bmatrix} \mathrm{vec}(\mathrm{Re}(H(s_1))) \\ \vdots \\ \mathrm{vec}(\mathrm{Re}(H(s_N))) \\ \mathrm{vec}(\mathrm{Im}(H(s_1))) \\ \vdots \\ \mathrm{vec}(\mathrm{Im}(H(s_N))) \end{bmatrix} \quad F = \begin{bmatrix} \mathrm{Re}(F_1) \\ \vdots \\ \mathrm{Re}(F_N) \\ \mathrm{Im}(F_1) \\ \vdots \\ \mathrm{Im}(F_N) \end{bmatrix} \tag{19}$$

$$F = Q_F R_F \tag{20}$$

Subsequently, Eq. (17) can be further transformed into the following form Eq. (21), which also represents the final definition of the loss function:

$$loss = Error_vec = (R_F y - b)^T (R_F y - b) + \delta^2 \tag{21}$$

where:

$$b = Q_F^T h, \quad \delta^2 = h^T h - b^T b \tag{22}$$

By decomposing F into Q_F and R_F, the quadratic term $(R_F y - b)^T (R_F y - b)$ is simplified to a standard least-squares problem, which is computationally more efficient. This simplification reduces the need for repeated computation of matrix norms, which would otherwise be required for every iteration in the optimization process. Furthermore, the decomposition helps in minimizing the number of operations required to update gradients and compute the loss, thereby enhancing overall computational efficiency.

C. Further Optimization for Memory and Time Consumption

Based on the new definition of the loss function, optimization problem Eq. (15) can be further refined as follows:

$$\begin{cases} \text{variable: } L, Q \\ \text{min: } f(y) = (R_F y - b)^T (R_F y - b) + \delta^2 \\ \text{subject to:} \\ x = \begin{bmatrix} \mathrm{vec}(L) \\ \mathrm{vec}(Q) \end{bmatrix} \\ y = pfe(x) \end{cases} \tag{23}$$

For nonlinear optimization problems like Eq.(23), traditional methods such as DAO typically compute the gradient vector and Hessian matrix to determine the search direction and step size, thereby accelerating convergence. However, the full calculation of the Hessian matrix imposes significant memory and time overhead [26], especially for high-dimensional optimization problems. Therefore, we employ the LBFGS method [27] to approximate the inverse of the Hessian matrix, which reduces memory usage while still accurately selecting the gradient descent direction, thus accelerating neural network convergence.

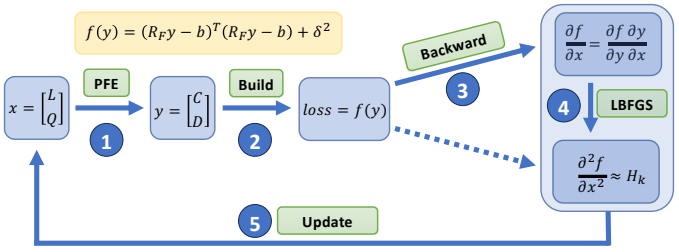


Fig. 3: Computational graph with LBFGS Method.

Figure 3 illustrates the computational graph. The input parameters are derived from Eq.(11) (the unconstrained system $W(s)$ in Figure 1), which include the parameter matrices L and Q. These matrices are vectorized into x. The computational graph includes five steps. ***Step 1***, x undergoes the PFE operation to generate parameters y (the passive system $G(s)$). ***Step 2***, using y along with other matrices from Eq. (20) and Eq. (22), the loss function is constructed. ***Step 3***, automatic differentiation is employed to compute the gradient information, applying the chain rule to perform backpropagation and calculate the first-order derivative of the loss function with respect to the network weights x. ***Step 4***, the LBFGS method is used to approximate the inverse Hessian matrix. ***Step 5***, the previously calculated

first-order derivative information and the approximated inverse Hessian matrix are used to update the parameters L and Q of the initial passive system. Above all, this completes one optimization iteration.

In the approximation process of ***Step 4***, it is important to note that the LBFGS algorithm does not store the full Hessian matrix directly [28]. Instead, it maintains a limited history of recent iterations to perform the calculations. The update formula for the Hessian inverse approximation in LBFGS is given by:

$$H_{k+1} = H_k - \frac{H_k \Delta g_k \Delta g_k^T H_k}{\Delta g_k^T \Delta x_k} + \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} \tag{24}$$

where:

1) H_k is the Hessian inverse approximation at iteration k.
2) $\Delta g_k = (\frac{\partial f}{\partial x})_{k+1} - (\frac{\partial f}{\partial x})_k$ is the change in the gradient.
3) $\Delta x_k = x_{k+1} - x_k$ is the change in the parameter.

The LBFGS is well-suited for this optimization problem due to its efficiency in managing large-scale variables and its ability to converge rapidly to a solution with a limited memory overhead. As experiments have shown, this optimization method, which combines efficient memory management and rapid convergence speed, provides robust support for optimizing complex circuit models.

IV. Experiments

A. Experimental Setup

We implement and test the proposed G-SpNN on an i7-14700KF @5.6GHz CPU with 32GB of memory, and a GeForce RTX 4070 SUPER GPU with 12GB of VRAM. G-SpNN is implemented based on PyTorch and compared against the state-of-the-art (SOTA) framework DAO, which is implemented in MATLAB and is open-sourced on GitHub[1].

For the DAO, we utilize MATLAB's built-in method $user.MemUsedMATLAB$ to record the memory consumption. For the G-SpNN, we monitor the GPU memory usage by calling PyTorch's $torch.cuda.memory_allocated()$.

The input files used in the experiments are standard Touchstone format files, derived from actual real-world simulations. After reading the files, the parameters s_k and $H^{m\times m}(s_k)$ $(1 \le k \le N)$ can be obtained. Before conducting the comparative experiments, the first two steps of macromodeling (as outlined in the preliminaries section) need to be performed on the raw data. Specifically, this involves applying the VF and LC method to process the data and generate four matrices: $A^{n\times n}$, $B^{n\times m}$, $C^{m\times n}$, and $D^{m\times m}$. The detailed information about these matrices and the results of the VF and LC method can be found in Tables I and II.

In this experiment, steady-state error (SS Error) is used as the basis for comparison. According to the data in Table II, the VF method initially produces a result with a very low steady-state error for the test samples. However, this result does not satisfy the passivity requirement. After applying the LC method for passivity correction, the passivity condition is met, but the steady-state error increases to some extent.

[1]https://www.github.com/yezuochang/pmm

TABLE I: Touchstone files.

Num	Case	Ports	n	m	N
1	Telluride	11	56	11	258
2	test_5	30	199	30	2000
3	sp125_uniform_2	64	342	64	400
4	CKDIST_TUNEDBUF	64	338	64	2000
5	pll_testcase	138	727	138	300

TABLE II: Fitting results of VF and LC.

Num	VF			LC		
	SS Error	Time(s)	Passivity	SS Error	Time(s)	Passivity
1	1.52e-4	0.8906	non-passive	5.91e-1	0.1875	passive
2	1.72e-7	17.78	non-passive	6.56e-5	7.64	passive
3	6.29e-4	25.59	non-passive	6.56e-4	6.906	passive
4	4.27e-3	96.89	non-passive	5.16e-3	17.21	passive
5	5.49e-4	75.23	non-passive	5.70e-4	91.79	passive

B. Passivity and Error

Table III presents the passivity results and steady-state errors for the G-SpNN and DAO methods. It is important to note that the steady-state error of the VF method represents the theoretical lower bound of the errors for both the G-SpNN and DAO methods. From the results, it is clear that both methods maintain passivity and further reduce errors compared to the LC method, with the G-SpNN error being closer to the VF error. Figure 4 shows the fitting results and errors for two points (H_{11} and H_{14}) in the H matrix of Case 1. It can be seen that G-SpNN provides a better overall fit to the original data, with errors lower than both the VF and LC method and the DAO method.

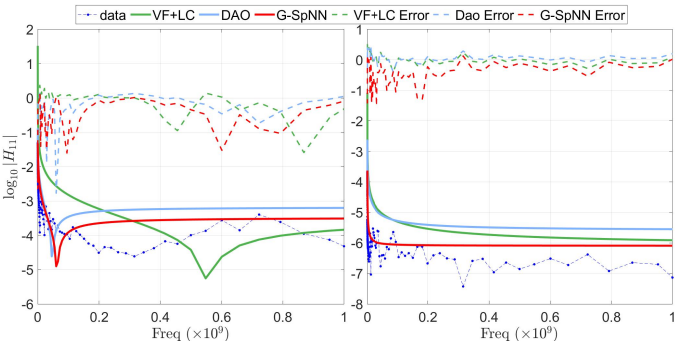


Fig. 4: Fitting accuracy of LC, DAO and G-SpNN.

C. Convergence Speed and Memory Usage

In Table III, we present the runtime, iteration count, initial loss, and final loss for both the G-SpNN and DAO methods. The convergence criterion is defined as a change in the loss function $\Delta\text{Loss} < 10^{-3}$. It is important to note that for Case 1, although the runtime of DAO appears shorter, the comparison of the final loss and steady-state error shows that DAO actually experiences pseudo-convergence and does not reach the optimal solution. In contrast, G-SpNN demonstrates better convergence performance. For Case 3 and Case 4, it should be noted that the DAO method is forcibly terminated during the iteration process due to memory overflow and does not reach the predefined convergence criterion. For Case 5, DAO experiences a memory overflow during the first iteration and could not complete the iteration.

TABLE III: Comparison of G-SpNN and DAO. The "–" indicates memory overrun during execution.

Num	Initial Loss	DAO [6]					G-SpNN				
		Time (s)	#Iteration	Final Loss	SS Error	Passivity	Time (s)	#Iteration	Final Loss	SS Error	Passivity
1	6.17e12	17.47	93	4.8656	2.26e-3	passive	94.19	232	4.19e-2	2.46e-4	passive
2	6.50e8	104.65	7	3.86e-2	2.65e-5	passive	97.93	328	3.89e-2	2.65e-5	passive
3	17.36	2116.48	4	17.21	6.51e-4	passive	300.46	803	17.15	6.47e-4	passive
4	2.31e3	3923.14	9	2.24e3	4.97e-3	passive	176.56	469	2.14e3	4.69e-3	passive
5	78.32	–	–	–	–	–	403.35	456	77.62	5.58e-4	passive
Average		1540.44	28	565.53	1.98e-3		214.49	458	446.97	1.23e-3	

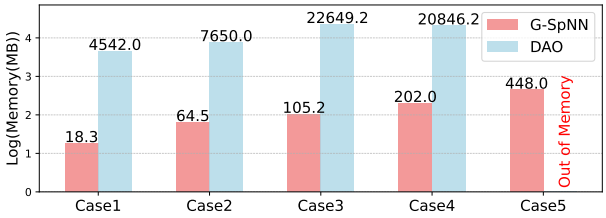


Fig. 5: G-SpNN vs. DAO in memory consumption.

For the tested cases, G-SpNN achieves an average speedup of 7.63× compared to DAO, and this speedup will increase even further with larger test cases. Therefore, G-SpNN is highly suitable for stable iterative convergence on large-scale cases.

We record the memory usage at each iteration to calculate the average memory consumption, as shown in Figure 5. Due to the high time and space complexity of the DAO method, we limit the number of poles in the VF method for cases 3-5 to ensure computational feasibility (which leads to higher SS Error and limits the reduction in loss). Nevertheless, it is still evident that as the test scale increases, DAO's memory usage rises significantly, while G-SpNN's memory usage remains relatively stable. The test results show that DAO's average memory consumption is 171.3 times that of G-SpNN.

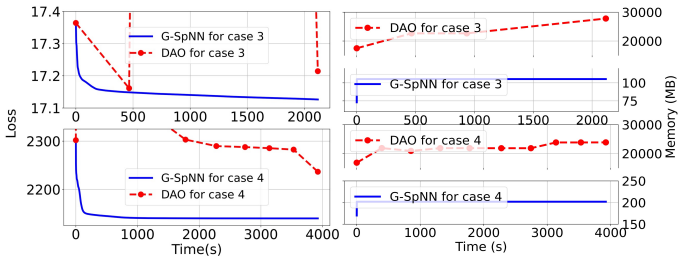


Fig. 6: G-SpNN vs. DAO in loss and memory over time (Case 3 and Case 4).

D. Detailed Analysis

To further compare the convergence efficiency of DAO and G-SpNN, we use Case 3 and Case 4 as examples, taking DAO's runtime for these cases as the benchmark. Figure 6 shows the loss and memory usage variations during iterations. It can be observed that G-SpNN has a smoother convergence process with better performance, achieving lower loss values per unit time compared to DAO. In contrast, DAO experiences gradient explosion, causing poor convergence and high memory usage, with significant fluctuations leading to memory explosions and premature termination. G-SpNN maintains significantly stable and lower memory usage throughout.

E. Comparison with Adam

For neural network training, optimization algorithms can leverage either first-order or higher-order gradient information. A commonly used first-order method is Adam, which is less sensitive to learning rate selection and performs well in many scenarios. We conduct tests comparing G-SpNN + Adam and G-SpNN + LBFGS on case 3 (using the same learning rate and configurations, only substituting the gradient computation method), with G-SpNN running for 80 seconds.

As shown in Figure 7, the Adam optimizer not only fails to reach lower loss values but also demonstrates instability, making its convergence behavior unpredictable. In contrast, the LBFGS method enables G-SpNN to progress steadily toward convergence, owing to second-order information guiding more effective update directions.

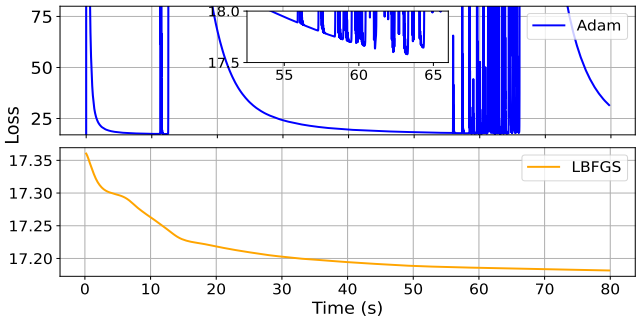


Fig. 7: Loss over time for G-SpNN with Adam and with LBFGS (Case 3).

V. Conclusion

In this paper, we introduce a novel perspective by mapping the passivity-enforced macromodeling problem into a neural network training task. The proposed optimization framework, G-SpNN, leverages advanced deep learning techniques as well as the LBFGS method. Compared to the current SOTA framework DAO, G-SpNN achieves significant improvements in both convergence speed and memory efficiency, which showcases exceptional capabilities to handle larger systems with a greater number of ports effectively.

Acknowledgment

The corresponding author is Zhou Jin. This work was supported by NSFC (Grant No.62234010, 92473107, 62204265, U23A20301, 62374031), Beijing National Research Center for Information Science and Technology (BNRist), Semiconductor Technology Innovation Center (Beijing) (Grant No.QYJS-2021-0900-B), and NSF of Jiangsu Province (BK20240173).

REFERENCES

[1] C. Coelho, J. Phillips, and L. Silveira, "A convex programming approach for generating guaranteed passive approximations to tabulated frequency-data," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, no. 2, pp. 293–301, 2004.

[2] B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting," IEEE Transactions on Power Delivery, vol. 14, no. 3, pp. 1052–1061, 1999.

[3] S. Grivet-Talocia, "An adaptive sampling technique for passivity characterization and enforcement of large interconnect macromodels," IEEE Transactions on Advanced Packaging, vol. 30, no. 2, pp. 226–237, 2007.

[4] B. Gustavsen, "Fast passivity enforcement for pole-residue models by perturbation of residue matrix eigenvalues," IEEE Transactions on Power Delivery, vol. 23, no. 4, pp. 2278–2285, 2008.

[5] T. Wang and Z. Ye, "Robust passive macro-model generation with local compensation," IEEE Transactions on Microwave Theory and Techniques, vol. 60, no. 8, pp. 2313–2328, 2012.

[6] Z. Ye, T. Wang, and Y. Li, "Domain-alternated optimization for passive macromodeling," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, pp. 2244–2255, 2015.

[7] T. Bradde, A. Zanco, and S. Grivet-Talocia, "Bivariate macromodeling with passivity constraints," Electrical Performance of Electronic Packaging and Systems. IEEE, pp. 1–3, 2021.

[8] Z. Jin, W. Li, Y. Bai, T. Wang, Y. Lu, and W. Liu, "Machine learning and gpu accelerated sparse linear solvers for transistor-level circuit simulation: A perspective survey," in 2024 29th Asia and South Pacific Design Automation Conference. IEEE, pp. 96–101, 2024.

[9] Y. Jiang, J. Song, X. Yin, X. Dong, S. Sun, Y. Lin, Z. Jin, X. Yang, and C. Zhuo, "A parallel simulation framework incorporating machine learning-based hotspot detection for accelerated power grid analysis," in Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD, pp. 1–7, 2024.

[10] J. Sun, X. Zha, C. Wang, X. Wu, D. Niu, W. Xing, and Z. Jin, "Pseudo adjoint optimization: Harnessing the solution curve for spice acceleration," in International Conference on Computer Aided Design, 2024.

[11] D. Deschrijver, M. Mrozowski, T. Dhaene, and D. De Zutter, "Macromodeling of multiport systems using a fast implementation of the vector fitting method," IEEE Microwave and Wireless Components Letters, vol. 18, no. 6, pp. 383–385, 2008.

[12] B. Gustavsen, "Passivity enforcement by residue perturbation via constrained non-negative least squares," IEEE Transactions on Power Delivery, vol. 36, no. 5, pp. 2758–2767, 2020.

[13] S. Grivet-Talocia, "Passivity enforcement via perturbation of hamiltonian matrices," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 51, no. 9, pp. 1755–1769, 2004.

[14] C. Coelho, J. Phillips, and L. Silveira, "Robust rational function approximation algorithm for model generation," in Proceedings 1999 Design Automation Conference, pp. 207–212, 1999.

[15] J. Rodriguez, "A loewner/mpm—vf combined rational fitting approach," in 2020 IEEE Power & Energy Society General Meeting. IEEE, pp. 1–1, 2020.

[16] A. Carlucci, T. Bradde, and S. Grivet-Talocia, "Improving robustness to termination conditions in passivity enforcement of rational macromodels," IEEE Transactions on Components, Packaging and Manufacturing Technology, pp. 1–1, 2024.

[17] L. F. Rodrigues, L. P. Ihlenfeld, and G. H. da Costa Oliveira, "A novel subspace identification approach with passivity enforcement," Automatica, vol. 132, pp. 109798, 2021.

[18] L. P. Ihlenfeld and G. H. Oliveira, "A faster passivity enforcement method via chordal sparsity," Electric Power Systems Research, vol. 204, pp. 107706, 2022.

[19] S. Grivet-Talocia, L. M. Silveira et al., "5 post-processing methods for passivity enforcement," Also of Interest, pp. 139, 2021.

[20] D. Youla, "On the factorization of rational matrices," IRE Transactions on Information Theory, vol. 7, no. 3, pp. 172–189, 1961.

[21] B. Anderson, K. Hitz, and N. Diem, "Recursive algorithm for spectral factorization," IEEE Transactions on Circuits and Systems, vol. 21, no. 6, pp. 742–750, 1974.

[22] B. Anderson, "A system theory criterion for positive real matrices," SIAM Journal on Control, vol. 5, no. 2, pp. 171–182, 1967.

[23] P. Lancaster, Algebraic Riccati Equations. Oxford Science Publications/The Clarendon Press, Oxford University Press, 1995.

[24] K. Filipiak, D. Klein, A. Markiewicz, and M. Mokrzycka, "Approximation with a kronecker product structure with one component as compound symmetry or autoregression via entropy loss function," Linear Algebra and its Applications, vol. 610, pp. 625–646, 2021.

[25] Z. Ye, Y. Li, M. Gao, and Z. Yu, "A novel framework for passive macromodeling," in Proceedings of the 48th Design Automation Conference, pp. 546–551, 2011.

[26] W.-K. Chen, Y. Zhang, B. Jiang, W.-H. Fang, and G. Cui, "Efficient construction of excited-state hessian matrices with machine learning accelerated multilayer energy-based fragment method," The Journal of Physical Chemistry A, vol. 124, no. 27, pp. 5684–5695, 2020.

[27] R. Bollapragada, J. Nocedal, D. Mudigere, H.-J. Shi, and P. T. P. Tang, "A progressive batching l-bfgs method for machine learning," in International Conference on Machine Learning, pp. 620–629, 2018.

[28] M. Borhani, "Multi-label log-loss function using l-bfgs for document categorization," Engineering Applications of Artificial Intelligence, vol. 91, pp. 103623, 2020.

MOSS: Multi-Modal Representation Learning on Sequential Circuits

Mingjun Wang[1,2,3,4], Bin Sun[1,3], Jianan Mu*[1,3], Feng Gu[1,2,3,4], Boyu Han[5], Tianmeng Yang[6], Xinyu Zhang[1,3], Silin Liu[1,3], Yihan Wen[7], Hui Wang[4], Jun Gao[4], Zhiteng Chao[1,3,4], Husheng Han[1,3], Zizhen Liu[1,3], Shengwen Liang[1,3], Jing Ye[1,3,4], Bei Yu[2], Xiaowei Li[1,3], Huawei Li*[1,3]

[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences [2]The Chinese University of Hong Kong. [3]University of Chinese Academy of Sciences [4]CASTEST Co., Ltd. [5]Stanford University [6]Peking University [7]Beijing University of Technology

***Abstract*—Deep learning has significantly advanced Electronic Design Automation (EDA), with circuit representation learning emerging as a key area for modeling the relationship between a circuit's structure and functionality. Existing methods primarily use either Large Language Models (LLMs) for Register Transfer Level (RTL) code analysis or Graph Neural Networks (GNNs) for netlist modeling. While LLMs excel at high-level functional understanding, they struggle with detailed netlist behavior. GNNs, however, face challenges when scaling to larger sequential circuits due to long-range information dependencies and insufficient functional supervision, leading to decreased accuracy and limited generalization. To address these challenges, we propose MOSS, a multimodal framework that integrates GNNs with LLMs for sequential circuit modeling. By enhancing D-type Flip-Flop (DFF) node features with embeddings from fine-tuned LLMs on RTL code, we focus the GNN on critical anchor points, reducing reliance on long-range dependencies. The LLM also provides global circuit embeddings, offering efficient supervision for functionality-related tasks. Additionally, MOSS introduces an adaptive aggregation method and a two-phase propagation mechanism in the GNN to better model signal propagation and sequential feedback within the circuit. Experimental results demonstrate that MOSS significantly improves the accuracy of functionality and performance predictions for sequential circuits compared to existing methods, particularly in larger circuits where previous models struggle. Specifically, MOSS achieves a 95.2% accuracy in arrival time prediction.**

I. Introduction

The application of deep learning (DL) techniques in EDA has garnered significant attention [1], [2], leading to notable advancements in areas such as routing [3], [4], synthesis [5], [6], and testing [7], [8]. Among these DL-driven EDA innovations, circuit representation learning [9], [10] has emerged as a promising research paradigm that propels predictions in subsequent EDA tasks. To further advance the development of circuit representation learning, modeling sequential circuits is particularly crucial, as they constitute the most commonly used circuit modules in practical electronic designs.

Existing works on circuit representation learning can mainly be divided into two categories, each modeling different modalities of circuit descriptions. The first category focuses on the register transfer level (RTL) code–a formal language that abstractly describes circuit behavior–utilizing large language models (LLMs) to provide a high-level understanding of functionality [11]. However, this approach faces challenges when it comes to analyzing the specific behavior and connectivity of individual cells in the netlist [12]. The second category employs graph neural networks (GNNs) for end-to-end learning, modeling the relationship between circuit functionality and structure [13]. Compared to the first approach, GNNs are closer to circuit structures, allowing for a more accurate representation of circuit details. Correspondingly, GNN-based models have achieved approximately 80% prediction accuracy for the toggle rates of sequential circuits within 1,000 gates. Such fine-grained tasks appear to be challenging for language models. Therefore, it seems to be a promising approach to use GNNs to model circuit structures for circuit representation.

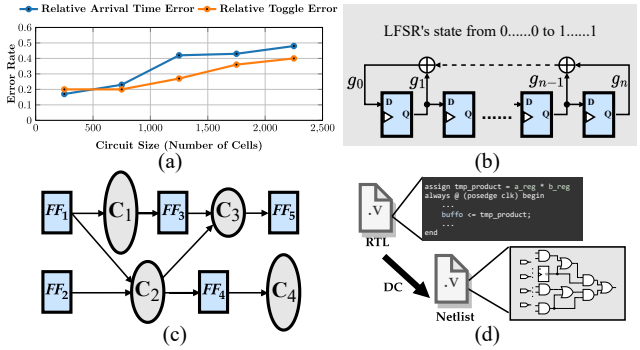


Fig. 1 Challenges and motivations for representation learning on sequential circuits. (a) Challenge 1: Performance degradation of existing methods in complex circuit prediction tasks. Both metrics are computed by summing the error ratio to the true value for each node and then averaging over all nodes. (b) Challenge 2: Sequential circuits can have almost infinite truth tables. (c) Motivation 1: DFFs divide sequential circuits. (d) Motivation 2: Different modalities provide different levels of abstraction in structure and functionality.

Unfortunately, when using GNNs to model larger sequential circuits for predicting their functionality and performance, we encounter significant challenges. The first challenge is the difficulty in handling long-range information dependencies, which leads to decreased accuracy on larger circuits. We implemented a GNN model, adopting asynchronous updates and other techniques from [14], and conducted tests on circuits of varying sizes to predict toggle rates and arrival times. The results, shown in Fig. 1(a), indicate that existing methods exhibit a substantial increase in error rates when applied to larger sequential circuits. For example, in a circuit with 2,000 gates, the prediction error rate exceeds 40%. One primary reason for this decline is that modeling larger circuits requires learning information from more distant nodes compared to smaller circuits. Despite employing asynchronous update strategies to propagate long-range information layer by layer, these advanced GNN designs remain susceptible to the loss of distant information, leading to decreased accuracy.

The second challenge is the difficulty in forming effective training supervision for the functionality of sequential circuits, resulting in weak capability in functionality-related tasks. Sequential circuits exhibit complex functional behaviors due to their stateful nature, making it challenging to capture functionality effectively with GNNs. The latest work, DeepSeq2 [14], artificially compresses the truth tables of individual nodes to form training supervision. However, this method has several drawbacks: Inefficiency for Large-Scale Circuits: As shown in Fig. 1(b), sequential circuits can have almost infinite truth tables because of the myriad possible states and input sequences over time. This makes the approach of compressing truth tables inefficient. Inadequate Description of Overall Circuit Functionality:

Manually extracting features from truth tables on each cell for training may aid in comparing individual nodes, but it is challenging to form a complete functional description of large-scale sequential circuits. This approach limits the ability to predict the complete circuit behavior and restricts the model's generalization capability. Consequently, this constraint limits performance on downstream tasks that require understanding the circuit's global functionality.

Interestingly, the structural characteristics of sequential circuits can be exploited to reduce the long-range dependencies encountered when modeling larger circuits. As shown in Fig. 1(c), sequential circuits inherently contain certain "anchor points", specifically D-type Flip-Flops (DFFs). These anchor points aggregate upstream input information and propagate it downstream, thereby driving the circuit's functional state. Furthermore, the interconnections between DFFs determine performance metrics such as arrival times. Therefore, by progressively and accurately modeling each DFF and its immediate connectivity network, we can localize the critical information within shorter-range dependencies. This approach reduces the reliance on long-range information propagation. However, existing methodologies have not yet taken advantage of this aspect for targeted modeling.

On the other hand, compared to the vast number of cell-level nodes present in the netlist, the RTL code offers a more macroscopic description of circuit connections and the functionality they form. As illustrated in Fig. 1(d), complex connectivity structures in a netlist can be succinctly represented by a single line of RTL code, providing a clear overview of the circuit's functionality. Therefore, if we can simultaneously learn from both RTL code and circuit netlist structures, integrating local and global information collaboratively, it will significantly enhance the efficiency of circuit representation learning. Moreover, by utilizing the formal language of RTL code-which provides an effective abstraction of functionalityand an LLM fine-tuned on circuit data to interpret the RTL, we can generate more accurate functional descriptions.

In this paper, we propose MOSS, a multimodal framework that integrates GNN with LLMs. We fine-tune an LLM on RTL code to generate embeddings that capture global circuit functionality. These embeddings enhance the features of DFF cells in the netlist, enabling the GNN to focus on critical anchor points and reduce reliance on long-range dependencies. The LLM also provides a global functionality embedding, offering efficient supervision for functionality-related tasks. Additionally, we introduce an adaptive aggregation method and a two-stage propagation mechanism to better model signal propagation and sequential feedback within the circuit. By combining structural insights from GNNs with functional understanding from LLMs, MOSS improves the accuracy of functionality and performance predictions for sequential circuits, particularly in larger circuits where existing methods struggle.

The contributions of this paper are summarized as follows:

- We propose MOSS, a multimodal framework that combines LLMs with GNNs for sequential circuit modeling, enhancing the collaboration between local node embeddings and global representations for circuit functionality and structure.
- We introduce an adaptive aggregation method and a two-phase propagation mechanism in the GNN to better model signal propagation and sequential feedback within circuits.
- MOSS significantly improves the accuracy of functionality and performance predictions for sequential circuits, outperforming existing methods on larger circuits. For example, it achieves a 95.2% accuracy in arrival time prediction.

II. Background and Related Work

A. Graph Neural Networks for EDA Tasks

Graph neural networks (GNNs) are a class of deep learning models specifically designed to process graph-structured data [15]. By operating directly on graphs, they capture complex dependencies and interactions between nodes. GNNs iteratively aggregate information from neighboring nodes to update node representations [16], enabling them to capture both local and global structural features [17]. In the field of EDA, GNNs are widely used for tasks such as circuit representation learning, performance prediction, and fault detection due to their high adaptability to circuit structures [18], [19].

For instance, FGNN2 [20] and DeepGate2 [21] focused on combinational circuits, which lack the timing dependencies inherent in sequential circuits. The DeepSeq series [14], [22] sought to incorporate temporal information into GNNs by training on logic and flip probabilities to model timing behavior. DeepSeq2 advanced this approach by introducing disentangled structure, function, and timing representations. However, as in Fig. 1, DeepSeq series struggles with long-range information dependencies. This shortcoming leads to a substantial decline in accuracy when applied to larger-scale circuits. Additionally, the DeepSeq series performs learning on And-Inverter Graphs (AIGs), which, compared to real netlists composed of standard cells, have nodes with simpler and uniform functions. In EDA scenarios, crucial labels such as timing are annotated at the standard cell level rather than the AIG level. Thus, AIG-based models cannot capture these metrics directly, limiting industrial applicability. In contrast, our work models netlists composed of standard cells.

B. Large Language Models and Fine-Tuning for EDA Tasks

Large language models (LLMs) have shown promise in various EDA tasks, such as design flow automation, error detection, and hardware generation [23]. Pre-trained on vast amounts of textual data, LLMs excel at understanding RTL code and descriptions [24]. Fine-tuning techniques like low-rank adaptation (LoRA) enable efficient adaptation of LLMs to EDA-specific tasks [25]. However, while LLMs can capture the functional semantics from RTL code, their ability to model the complex structural dependencies inherent in circuits is limited, as they were initially designed for processing textual data [26]. Therefore, LLMs alone are insufficient for circuit design tasks that require both functional understanding and detailed structural representation.

C. Multimodal Alignment and Contrastive Learning

Multimodal alignment and contrastive learning have been successfully employed to integrate diverse data types [27], such as text and images [28], by aligning their representations in a shared embedding space [29]. Prominent models like CLIP [30] demonstrate the potential of contrastive learning for cross-modal tasks, including retrieval and classification. These approaches leverage different modalities' strengths to enhance the performance of representation learning [31].

III. Problem Formulation

Sequential circuits, modeled as directed graphs $G = (V, E)$, where V represents circuit components (e.g., logic gates, DFFs) and E represents component connections, pose unique challenges due to the temporal dependencies introduced by state elements. The key problem is to learn node embeddings $\mathbf{H} \in \mathbb{R}^{|V| \times d}$ that encode both structural and temporal features. These embeddings should support downstream tasks such as toggle rate prediction, arrival time prediction, and circuit classification:

$$\mathbf{H} = f(G, \mathbf{X}, \mathbf{T}) = f(V, E, \mathbf{X}, \mathbf{T}), \tag{1}$$

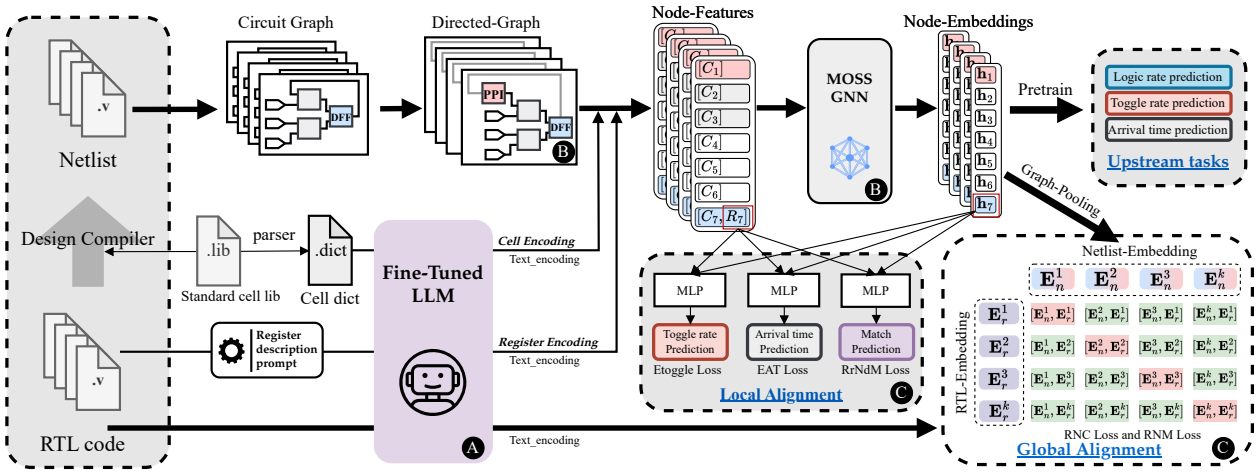


Fig. 2 Overview of the MOSS framework.

where $\mathbf{X} \in \mathbb{R}^{|V| \times d_x}$ represents structural features, and $\mathbf{T} \in \mathbb{R}^{|V| \times d_t}$ represents temporal features.

The main challenge of circuit representation learning is capturing long-range dependencies and temporal dynamics in sequential circuits, while most current GNNs struggle with over-smoothing and fail to retain critical temporal information. Integrating multimodal data, such as netlists and RTL descriptions, is crucial for comprehensively understanding functional and structural aspects.

IV. MOSS Framework

Fig. 2 describes MOSS's framework. We integrate LLM and GNN to capture both functional semantics from RTL code and structural dependencies from netlists. Our key approaches are listed below:

- LLM-enhanced node features (Fig. 2 Ⓐ): The LLM is fine-tuned on RTL data to generate contextual embeddings, enriching the GNN's node features, particularly for register nodes (DFFs). Details are described in Section IV-A.
- Graph construction and modeling (Fig. 2 Ⓑ): The netlist is represented as a directed graph. MOSS uses multiple aggregators and Pseudo Primary Input (PPI)/DFF nodes to model temporal dependencies in sequential circuits. Details are described in Section IV-B.
- Local and global alignment (Fig. 2 Ⓒ): MOSS aligns GNN embeddings with LLM-generated RTL embeddings. Locally, tasks like toggle rate and arrival time prediction are optimized using Etoggle Loss (enhanced toggle loss), EAT Loss (enhanced arrival time loss) and RrNdM (RTL-register to Netlist-DFF matching loss). Globally, RNC Loss (RTL-Netlist contrastive loss) and RNM Loss (RTL-Netlist matching loss) align the netlist and RTL embeddings. Details are described in Section IV-C.

A. LLM Fine-Tuning and Node Feature Enhancement

To address the challenges of modeling sequential circuits, we enhance the GNN's node features by leveraging a fine-tuned LLM to generate contextual embeddings for DFF nodes and embeddings for each logic cell. Additionally, to provide more effective training supervision for circuit functionality tasks, we use the LLM to generate embeddings for the complete RTL code. To this end, we specifically fine-tuned an LLM to enhance its understanding of RTL code.

LLM Fine-Tuning. We fine-tuned Yi-Coder-9B-Chat [32], an open-source code generation model, on 31,701 RTL descriptions to extract contextual embeddings for registers and logic cells.

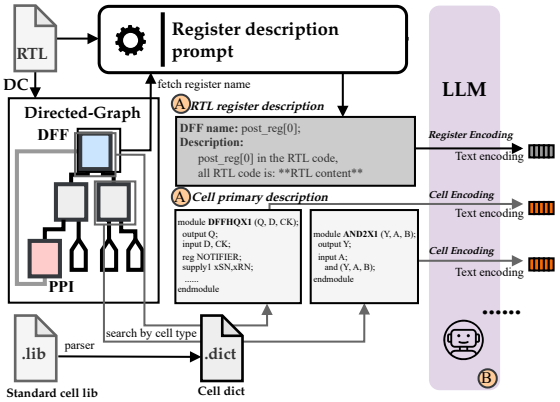


Fig. 3 LLM feature extraction process for GNN nodes.

Data Extraction and RTL Descriptions. To enhance the features of circuit nodes, we identify each node in the circuit and extract its description as shown in Fig. 3 Ⓐ . We first use Design Compiler (DC) [33] to synthesize the netlist, resulting in a directed graph structure. We then identify each elements of DFFs and logic cells. Further, for each node in the netlist, we extract its cell description, which typically includes structural and functional information. Specifically for DFFs, we locate the corresponding registers in the RTL code and generate Register Description Prompts. These prompts enable the LLM to describe the context and functionality of each DFF, capturing both local and global functional relationships.

LLM-GNN Feature Integration and Alignment. Using the extracted node information and descriptions, we generate embeddings with the LLM as illustrated in Fig. 3 Ⓑ . We use mean pooling to aggregate token embeddings from the LLM, providing a comprehensive representation of the input sequence. LLM-generated node embeddings are concatenated with each node's structural features in the GNN. For DFFs, the produced embeddings combine the RTL description and the DFF cell description, capturing both structural and functional information. For logic cells, the embeddings corresponds to each cell's cell description.

B. Graph Construction and Temporal Dependency Modeling

In our GNN design, we first utilize the multimodal characteristics of MOSS to generate node embeddings. Next, we design an adaptive aggregator to better accommodate standard cell-level netlists. Finally,

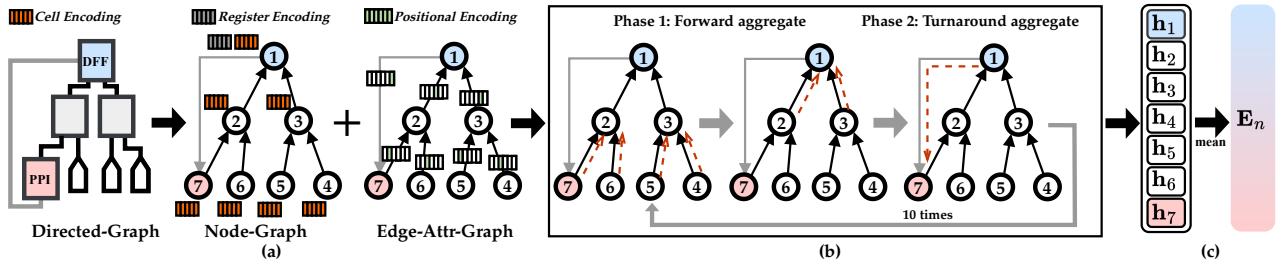


Fig. 4 (a) Graph structure in MOSS; (b) Two-phase Asynchronous temporal propagation with multi-encoding; (c) Mean-pooling strategy for graph representation.

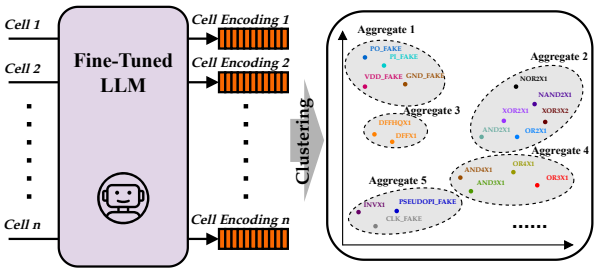


Fig. 5 Adaptive-aggregator design with adaptive clustering.

we employ a two-phase asynchronous temporal propagation mechanism to more accurately model signal propagation in circuits.

Graph Construction. In constructing the graph, we diverge from prior works that rely on manually embedding labels derived from the circuit graph's structural information. Instead, we leverage the multimodal capabilities of MOSS by utilizing the LLM to process descriptions of the RTL code and cell nodes. This approach generates embeddings for the graph nodes, as illustrated in Fig. 4(a).

First, for each cell node, we utilize the LLM to generate embeddings based on the cell's functional and structural descriptions in the standard cell library. Specifically, for the "anchor points" in sequential circuit structures, the DFFs, we further enhance their embeddings. We overlay the LLM embeddings of the contextual structure and functional descriptions of the corresponding register nodes from the RTL code onto the DFF nodes. This enhancement helps the GNN to capture a wider range of information more accurately at DFF nodes.

Finally, since different edges of cells have varying propertiesfor example, in certain logic gates, different inputs have different influences on the outputwe perform positional encoding on the edges.

Adaptive-Aggregator Design. Because we operate at the standard cell level rather than at the AIG level used in existing designs, our netlists contain cells with diverse functions and structures, and different cell libraries may include varying cell types. Therefore, we propose a clustering-based adaptive aggregator for standard cell-level netlists to adapt to different kinds of cells automatically.

As shown in Fig. 5, MOSS first uses Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and hierarchical clustering to dynamically group nodes based on their LLM-derived embeddings. DBSCAN clusters nodes based on functional similarity, detecting clusters of varying density without specifying the number of clusters in advance. Hierarchical clustering further refines these clusters by considering both functional similarities and structural dependencies such as fan-in and fan-out.

Then, each group is processed by a dedicated aggregator that learns to aggregate information based on the characteristics of the nodes. Notably, we employ graph attention mechanisms for aggregation, ultimately forming different attention-based aggregators for each category. This adaptive aggregator approach enables MOSS to flexibly adapt to different node types and interactions, capturing unique behaviors essential for modeling complex circuits.

Two-Phase Asynchronous Temporal Propagation Mechanism. To accurately model temporal dependencies in sequential circuits, MOSS adopts a two-phase asynchronous temporal propagation mechanism as in [14]. The propagation mechanism, illustrated in Fig. 4(b), includes forward and backward phases to handle the feedback behavior of DFFs. It is worth noting that both phases use asynchronous updates to more closely resemble signal propagation in circuits.

In the first phase, Forward Propagation, signals are propagated from primary inputs (PIs) to the inputs of DFFs through combinational logic gates. This phase captures the forward signal flow in the circuit, simulating how input signals affect downstream components.

In the second phase, referred to as Turnaround Propagation, the outputs from the DFFs are fed back into the circuit, simulating the characteristic feedback loops of sequential elements. This feedback is crucial for modeling the timing dependencies and state transitions intrinsic to sequential circuits, allowing the GNN to capture how past states influence future behavior. To capture both short-range and long-range dependencies, the two-phase propagation process is repeated multiple times (e.g., 10 iterations).

Afterward, as shown in Fig. 4(c), the final node representations are aggregated using mean-pooling, generating comprehensive embeddings that effectively summarize the temporal and structural information in the circuit.

C. Local and Global Alignment Strategy

Our local and global alignment strategy ensures that the GNN-generated netlist embeddings remain consistent with the LLM-derived RTL embeddings, capturing both local and global circuit characteristics. This section details the loss functions used to achieve alignment, focusing on key tasks such as toggle rate prediction, arrival time prediction, and register-DFF matching.

Local alignment aims to align nodes through task-specific losses, optimizing the GNN's ability to represent local circuit behavior. The local alignment contains several prediction tasks, and their loss functions are Etoggle loss, EAT loss, and RrNdM loss. Etoggle Loss minimizes the difference between predicted and actual toggle rates, capturing logic gates' dynamic behavior. EAT Loss focuses on predicting signal arrival times, ensuring the GNN accurately models timing dependencies. RrNdM Loss ensures proper alignment between RTL registers and netlist DFFs, minimizing mismatches in register-to-DFF mapping. We implement these loss functions as smooth L1 loss functions.

Global alignment ensures consistency between the overall structure and functionality of netlist and RTL embeddings through graph

```
# rtl_encoder - Text Transformer
# netlist_encoder - GNN
# R[n, l] - minibatch of aligned RTL codes
# N[n, c, m] - minibatch of aligned netlists
# W_n[d_n, d_r] - projection of embedding
# MLP[2*d_r, 1] - MLP for binary classification
# t - temperature parameter
# extract embedding of each modality
R_f = rtl_encoder(R)      #[n, d_r]
N_f = netlist_encoder(N) #[n, d_n]
# joint multimodal embedding [n, d_r]
R_e = l2_normalize(R_f, axis=1)
N_e = l2_normalize(np.dot(N_f, W_n), axis=1)
# RTL-Netlist Contrastive (RNC) Loss
RNCLogits = np.dot(R_e, N_e.T) * np.exp(t)
RNCLabels = np.arange(n)
loss_r = cross_entropy_loss(logits, labels, axis=0)
loss_n = cross_entropy_loss(logits, labels, axis=1)
RNCLoss = (loss_r + loss_n) / 2
# RTL-Netlist Matching (RNM) Loss
RNMLogits = [MLP(np.concatenate(R_e[i], N_e[j]))
             for i in range(n) for j in range(n)]
RNMLabels = np.eye(n)
RNMLoss = SmoothL1Loss(RNMLogits, RNMLabels)
```

Fig. 6 Numpy-like pseudocode for loss function of global alignment.

pooling and alignment losses. While RNC loss aligns the global structure, RNM loss enforces functional consistency between netlist and RTL. RNC and RNM calculation method is shown in the pseudocode in Fig. 6.

MOSS employs a multi-task learning strategy to generalize across multiple circuit-related tasks. By optimizing a weighted combination of task-specific loss functions, MOSS captures local node-level behaviors and global circuit-wide patterns. The overall loss is formulated as:

$$L_{\text{total}} = \sum_{i=1}^{n} \lambda_i L_{\text{taski}}, \tag{2}$$

where λ_i dynamically adjusts to balance the contribution of each task, ensuring that no single task dominates the learning process. This approach enhances the model's ability to generalize effectively across diverse sequential circuit tasks.

MOSS employs a dual alignment strategy that integrates local task-specific losses (Etoggle, EAT, and RrNdM) with global consistency losses (RNC and RNM). This approach captures both fine-grained circuit dynamics and global structural behavior, enhancing the model's accuracy and generalizability, particularly for complex sequential circuits.

V. Experimental Results

A. Experimental Settings

Datasets. We collected 31,701 RTL designs and synthesized them using Synopsys Design Compiler (DC), applying multiple rounds of optimization. For each RTL, we generated several distinct circuits, with sizes ranging from 100 to 5000 cells, all based on real-world designs. From this collection, we further selected a subset of RTLs with diverse functionalities and significant variations as our experimental dataset.

Model Variants. We evaluate several MOSS variants to understand the impact of critical components: "MOSS" is the full model; "MOSS w/o A" is the full model withoutht local-global alignment strategy; "MOSS w/o AA" includes LLM features but omits adaptive-aggregator and alignment; "MOSS w/o FAA" is the model without all the LLM feature enhancement, the adaptive-aggregator design, and the alignment strategy. Besides, we implement DeepSeq2 [14], the state-of-the-art GNN-based model, for the comparison.

Evaluation Metrics. We assess all models using four metrics: the accuracy of arrival time prediction (ATP) for each DFF, the accuracy of toggle rate prediction (TRP) for each cell, the accuracy of power prediction (PP) for the circuit, and the accuracy of functional equivalence prediction (FEP). Note here the prediction accuracy is measured by

$$\text{accuracy} = 1 - \text{mean relative error}. \tag{3}$$

FEP measures the rate of correctly identifying functionally equivalent RTL-netlist pairs. Higher values indicate better performance. Ground truth data is collected using commercial EDA tools. Arrival Time (AT) is obtained via timing analysis on DFF nodes using PrimePower [34] and Synopsys DC [33]. Toggle Rate is derived from VCS [35] simulations over 60,000 cycles with random inputs, and power is reported by PrimePower based on their toggle rates. These ground truths ensure accurate benchmarking of MOSS's predictions.

Platform Configuration. Experiments were performed on an 8-GPU system with NVIDIA A100 GPUs. We implemented MOSS in PyTorch and trained using the Adam optimizer, with a learning rate of 6×10^{-4} and a batch size of 32. Each experiment ran for 45 epochs, with early stopping based on validation performance to prevent overfitting.

B. Experimental Results and Analysis

In this section, we evaluate the performance of MOSS across various tasks, comparing it with its variants and the baseline model DeepSeq2. The experiments focus on critical tasks such as arrival time prediction, toggle rate prediction, power prediction, and functional equivalence checking. We list results in TABLE I and TABLE II, and analyze the training loss curves shown in Fig. 7 and Fig. 8.

Arrival Time Prediction (ATP). For arrival time prediction (ATP), MOSS achieves an average accuracy of 95.2%, significantly outperforming DeepSeq2 (79.1%). In larger circuits, such as *mult_16x32_to_48*, MOSS maintains high accuracy (94.3%), while DeepSeq2 drops to 57.6%. This highlights MOSS's ability to handle long-range dependencies, a known challenge in larger circuits. The alignment strategy has a limited impact on this task, as demonstrated by the performance of MOSS w/o A (94.9%), which remains high but is slightly lower than the full model.

Toggle Rate Prediction (TRP). For toggle rate prediction (TRP), MOSS achieves 87.5% accuracy, outperforming DeepSeq2 (76.4%). In circuits such as *pipeline_reg*, MOSS reaches 92.4%, whereas MOSS w/o FAA drops to 63.6%, underscoring the importance of the adaptive aggregation and LLM enhancements, which are key for capturing dynamic behaviors in sequential circuits.

Power Prediction (PP). For power prediction (PP), MOSS leads with 96.3% accuracy, compared to DeepSeq2's 88.4%. The LLM-enhanced node features allow MOSS to model power consumption patterns more effectively, particularly in circuits like *wb_data_mux*, where MOSS achieves 96.2%, significantly higher than MOSS w/o FAA (82.6%). The smaller performance gap between MOSS and MOSS w/o A (95.1%) indicates that alignment is less critical for power estimation compared to feature enhancement.

Functional Equivalence Prediction (FEP). We selected several datasets with 512 pairs of RTL and netlist in each set and calculated the average correctness of their predictions. In functional equivalence checking, MOSS excels with an average accuracy of 93.7%, far

TABLE I Performance Comparison of MOSS Framework Variants

Circuit	#Cells	DeepSeq2 [14]			MOSS w/o FAA			MOSS w/o AA			MOSS w/o A			MOSS		
		ATP	TRP	PP	ATP	TRP	PP	ATP	TRP	PP	ATP	TRP	PP	ATP	TRP	PP
max_selector	278	81.4	78.7	94.6	47.0	75.8	88.6	82.3	85.2	94.5	95.4	89.4	99.9	**95.6**	**90.5**	**99.9**
pipeline_reg	610	77.6	83.6	91.4	52.2	63.6	63.4	80.5	88.3	90.2	94.2	92.1	94.1	**94.5**	**92.4**	**94.6**
prbs_generator	643	87.8	76.5	71.7	57.2	72.7	81.7	78.6	82.5	90.8	92.0	**87.4**	94.5	**93.0**	85.4	**95.1**
shift_reg_24	731	86.9	80.9	90.4	58.2	63.9	75.4	82.4	85.6	92.5	**96.2**	**90.2**	**97.6**	95.8	89.0	97.5
error_logger	812	79.5	83.2	94.3	58.5	59.0	81.3	81.2	80.2	95.3	94.5	85.4	99.5	**95.0**	**86.3**	**99.7**
signed_mac	1306	66.4	77.3	**95.6**	26.9	56.1	73.6	76.5	78.4	88.6	93.8	83.8	92.3	**94.5**	**85.3**	94.1
wb_data_mux	1364	95.7	64.3	88.6	45.3	25.5	82.6	85.4	75.6	88.5	98.8	82.9	91.2	**99.1**	**83.3**	**96.2**
mult_16x32_to_48	4144	57.6	66.6	80.1	19.3	40.1	54.1	75.2	72.3	85.4	93.9	84.8	91.5	**94.3**	**87.9**	**93.5**
Average	-	79.1	76.4	88.4	45.6	57.1	75.1	80.3	81.0	90.7	94.9	87.0	95.1	**95.2**	**87.5**	**96.3**

ATP: Arrival Time Prediction accuracy (%). **TRP**: Toggle Rate Prediction accuracy (%). **PP**: Power Prediction accuracy (%).
MOSS w/o FAA: MOSS without Feature Enhancement, Adaptive-Aggregator and Alignment.
MOSS w/o AA: MOSS without Aligment and Adaptive-Aggregator. **MOSS w/o A**: MOSS without Alignment.

TABLE II RTL-netlist functional equivalence prediction accuracy (FEP) on different circuit sources

Circuit	MOSS w/o FAA	MOSS w/o AA	MOSS w/o A	MOSS
github_0	4.8	19.4	24.1	**91.4**
github_1	5.3	20.3	33.6	**95.0**
github_2	10.0	23.7	32.0	**94.3**
huggingface_0	7.9	16.4	19.5	**94.1**
huggingface_1	8.7	18.3	22.9	**93.6**
huggingface_2	14.1	21.1	27.5	**93.5**
Average	8.5	19.9	26.6	**93.7**

Bold numbers indicate the best performance for each circuit
All circuits are from public repositories (GitHub and HuggingFace)

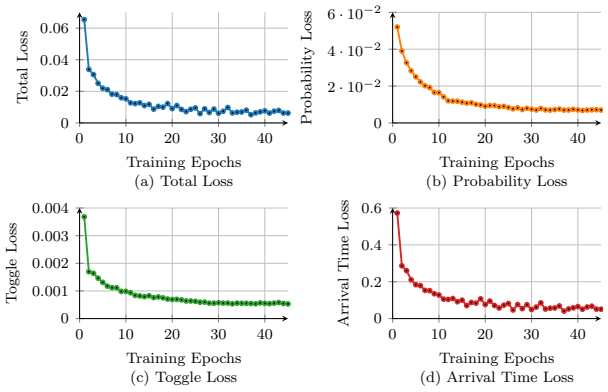


Fig. 7 Losses in the pre-training section.

exceeding MOSS w/o FAA (8.5%) , MOSS w/o AA (19.9%) and MOSS w/o A (26.6%). This significant drop in variants highlights the necessity of both feature enhancement and alignment to maintain functional consistency between RTL and netlist representations. For example, in the *github_0* circuit sets, MOSS achieves 91.4%, while MOSS w/o A only reaches 24.1%, confirming the critical role of multimodal alignment. This experiment shows that by leveraging multimodal learning, MOSS achieves superior circuit functionality modeling, enabling functional equivalence checking.

Loss Analysis. As illustrated in Fig. 7, in the pre-training phase, all loss components steadily decrease, with the total loss (see Fig. 7(a)) showing continuous improvement. Probability loss (see Fig. 7(b)), toggle loss (see Fig. 7(c)) and arrival time loss (see Fig. 7(d)) decrease steadily, indicating that MOSS learns to capture both timing and dynamic behaviors effectively.

Fig. 8 shows the global loss in multimodal alignment. The total

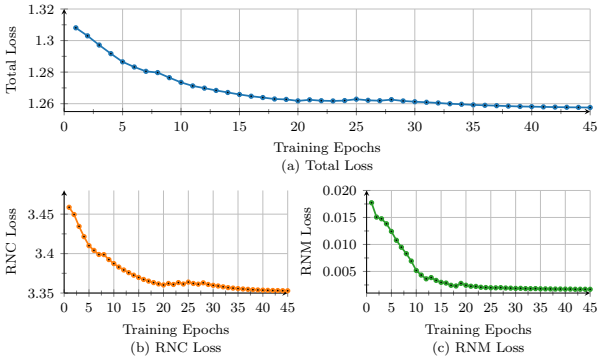


Fig. 8 Global Losses in the multimodal alignment section.

loss stabilizes around 1.26 after 45 epochs, indicating effective convergence. The RNC loss (see Fig. 8(b)) and RNM loss (see Fig. 8(c)) also converge smoothly, confirming the success of the alignment strategy. The RNM loss reaches as low as 0.002, indicating that the RTL-netlist matching is highly accurate.

The experimental results demonstrate that MOSS effectively addresses the challenges of circuit representation learning, particularly in handling larger circuits with long-range dependencies. By integrating LLM-enhanced features and GNN-based structural learning, MOSS significantly improves the accuracy of arrival time, toggle rate, and power predictions, while ensuring functional consistency through multimodal alignment.

VI. Conclusion

We propose MOSS, a multimodal framework combining GNNs and LLMs to enhance sequential circuit representation. MOSS addresses long-range information loss by leveraging LLMs for global semantic understanding and GNNs for local structural features. It also introduces adaptive aggregation and two-stage propagation to improve signal modeling. Experimental results show that MOSS significantly boosts accuracy for tasks like toggle rate and arrival time prediction, achieving 95.2% accuracy in arrival time prediction for large circuits, outperforming existing methods. We hope that MOSS can not just offer an effective solution for circuit functionality and performance prediction, but also stimulate more future research.

Acknowledgement

This paper is supported in part by the Chinese Academy of Sciences under grant No. XDB0660102, and in part by the National Natural Science Foundation of China (NSFC) under grant No. (62090024). The corresponding authors of this paper are Jianan Mu and Huawei Li.

REFERENCES

[1] L. Dai, B. Wang, X. Cheng, Q. Wang, and X. Ni, "The application of deep learning technology in integrated circuit design," *Energy Informatics*, vol. 7, no. 1, pp. 1–20, 2024.

[2] C. Tian, L. Chen, Y. WANG, S. Wang, J. Zhou, Y. Pang, and Z. Du, "A survey on fpga electronic design automation technology based on machine learning," *Journal of Electronics & Information Technology*, vol. 45, no. 1, pp. 1–13, 2023.

[3] J. Yan, X. Lyu, R. Cheng, and Y. Lin, "Towards machine learning for placement and routing in chip design: a methodological overview," *arXiv preprint arXiv:2202.13564*, 2022.

[4] A. Al-Hyari, H. Szentimrey, A. Shamli, T. Martin, G. Grewal, and S. Areibi, "A deep learning framework to predict routability for FPGA circuit placement," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 14, no. 3, pp. 1–28, 2021.

[5] R. Roy and S. Godil, "Machine Learning for Logic Synthesis," in *Machine Learning Applications in Electronic Design Automation*. Springer, 2022, pp. 183–204.

[6] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. Süsstrunk, and G. De Micheli, "Deep learning for logic optimization algorithms," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–4.

[7] A. Kumar, S. L. Tripathi, and K. S. Rao, *Machine Learning Techniques for VLSI Chip Design*. John Wiley & Sons, 2023.

[8] A. Choubey and S. B. Choubey, "Machine Learning for Testing of VLSI Circuit," in *VLSI and Hardware Implementations Using Modern Machine Learning Methods*. CRC Press, 2021, pp. 23–40.

[9] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 61–66.

[10] P. Shrestha, S. Phatharodom, and I. Savidis, "Graph representation learning for gate arrival time prediction," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2022, pp. 127–133.

[11] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[12] C.-T. Ho and H. Ren, "Large Language Model (LLM) for Standard Cell Layout Design Optimization," *arXiv preprint arXiv:2406.06549*, 2024.

[13] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "CktGNN: Circuit graph neural network for electronic design automation," *arXiv preprint arXiv:2308.16406*, 2023.

[14] S. Khan, Z. Shi, Z. Zheng, M. Li, and Q. Xu, "DeepSeq2: Enhanced Sequential Circuit Learning with Disentangled Representations," *arXiv preprint arXiv:2411.00530*, 2024.

[15] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[16] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 793–803.

[17] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[18] E. Chien, M. Li, A. Aportela, K. Ding, S. Jia, S. Maji, Z. Zhao, J. Duarte, V. Fung, C. Hao *et al.*, "Opportunities and challenges of graph neural networks in electrical engineering," *Nature Reviews Electrical Engineering*, vol. 1, no. 8, pp. 529–546, 2024.

[19] A. Said, M. Shabbir, B. Broll, W. Abbas, P. Völgyesi, and X. Koutsoukos, "Circuit design completion using graph neural networks," *Neural Computing and Applications*, vol. 35, no. 16, pp. 12 145–12 157, 2023.

[20] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, Y. Huang, and B. Yu, "FGNN2: A Powerful Pre-Training Framework for Learning the Logic Functionality of Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[21] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "DeepGate2: Functionality-aware circuit representation learning," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–9.

[22] S. Khan, Z. Shi, M. Li, and Q. Xu, "DeepSeq: Deep Sequential Circuit Learning," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2024, pp. 1–2.

[23] Z. He and B. Yu, "Large Language Models for EDA: Future or Mirage?" in *Proceedings of the 2024 International Symposium on Physical Design*, 2024, pp. 65–66.

[24] A. Korvala, "Analysis of LLM-models in optimizing and designing VHDL code," 2023.

[25] U. Sharma, B.-Y. Wu, S. R. D. Kankipati, V. A. Chhabria, and A. Rovinski, "OpenROAD-Assistant: An Open-Source Large Language Model for Physical Design Tasks," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2024, pp. 1–7.

[26] M. Abdollahi, S. F. Yeganli, M. A. Baharloo, and A. Baniasadi, "Hardware Design and Verification with Large Language Models: A Literature Survey, Challenges, and Open Issues," 2024.

[27] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 2, pp. 423–443, 2018.

[28] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International conference on machine learning*. PMLR, 2022, pp. 12 888–12 900.

[29] A. Rahate, R. Walambe, S. Ramanna, and K. Kotecha, "Multimodal co-learning: Challenges, applications with datasets, recent advances and future directions," *Information Fusion*, vol. 81, pp. 203–239, 2022.

[30] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning (ICML)*, 2021, pp. 8748–8763.

[31] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *Ieee Access*, vol. 8, pp. 193 907–193 934, 2020.

[32] A. Young, B. Chen, C. Li, C. Huang, G. Zhang, G. Zhang, H. Li, J. Zhu, J. Chen, J. Chang *et al.*, "Yi: Open foundation models by 01. ai," *arXiv preprint arXiv:2403.04652*, 2024.

[33] Synopsys, Inc., *Design Compiler® User Guide*, Synopsys, Inc., Mountain View, CA, USA, Apr. 2023.

[34] Synopsys , Inc., "Primepower user guide," Synopsys, Inc., Mountain View, CA, USA, Mar. 2022.

[35] Synopsys, Inc., *VCS User Guide*, Synopsys, Inc., Mountain View, CA, USA, Sep. 2019.

EPICS: Efficient Parallel Pattern Fault Simulation for Sequential Circuits via Strongly Connected Components

Mingjun Wang[1,2,3], Hui Wang[3], Jianan Mu*[1,3], Xinyu Zhang[1,2], Bin Sun[1,2], Yihan Wen[4], Zizhen Liu[1,3], Feng Gu[1,2,3], Jun Gao[3], Shengwen Liang[1,2], Jing Ye*[1,2,3], Xiaowei Li[1,2], Huawei Li[1,2]
[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences [3]CASTEST Co., Ltd. [4]Beijing University of Technology

***Abstract*—As functional safety of electronic chips gains importance in autonomous vehicles and aerospace, standards like ISO 26262 mandate high diagnostic coverage, requiring extensive gate-level fault simulations. However, for large-scale industrial sequential circuits, these simulations are time-consuming, creating a significant bottleneck in chip development. Prior approaches have focused on reducing computational complexity and optimizing CPU hardware usage by minimizing redundant computations during fault propagation and leveraging bit-level parallel processing capabilities. Techniques like parallel-pattern and event-driven simulations have improved performance in combinational circuits but face limitations in sequential circuits due to timing dependencies within loops. The challenge lies in parallelizing simulations across different cycles without violating these dependencies, which is exacerbated by the complex feedback structures in SCCs. In this work, we propose a novel parallel-pattern fault simulation framework that combines loop fusion with efficient event traversal to accelerate sequential circuit simulations. By compiling simple loops into larger nodes, we reduce the number of feedback events without introducing excessive redundancy. For larger SCCs, we develop specialized algorithms for selecting loop entrance nodes based on indegree analysis and implement the lazy propagation strategy for internal nodes. This approach minimizes simulation events caused by inaccurate predictions and reduces overhead associated with false event propagation. We integrate these techniques into our simulation framework, EPICS, which strategically mixes compiled and event-driven simulations to optimize performance. Experimental results demonstrate that EPICS achieves a 5.94× speedup over state-of-the-art commercial tool while maintaining the same fault coverage.**

I. Introduction

Since sectors like autonomous vehicles and aerospace have become critical to modern life, the functional safety of electronic chips has gained increased attention [1]. ISO 26262 standard [2] outlines safety requirements for automotive-grade chips [3], including MCUs, ASICs, and SoCs [4]. Meeting the requirements demands high diagnostic coverage [5], requiring extensive fault simulations [6]. Fine-grained fault simulation requires injecting faults into gate-level netlist and analyzing output changes under input patterns [7]. Due to the large scale of industrial circuits and numerous fault insertion points, sequential circuit gate-level fault simulation can take weeks [8], creating a significant bottleneck in chip development [9]. Accelerating gate-level fault simulation is urgent [10].

Despite the current apparent and ongoing demand for acceleration, the challenge of speeding up fault simulation in sequential circuits is far from new [11]. This long-standing issue continues to be of critical importance, with significant room for improvement. Previous research can be divided into two main areas [12]: reducing computational complexity and optimizing for CPU hardware. The focus has been on minimizing redundant computations while propagating good and faulty values across the circuit and efficiently executing bit-level gate simulations on CPUs with high bit-parallel processing capabilities. The concurrent simulation uses event differentials to avoid redundant faults and good values simulations under the same inputs [13]. To better utilize CPUs' multi-bit parallel processing abilities, PARIS introduced the parallel pattern fault simulation [14], and PROOFS introduced the parallel fault simulation [15]. They merge the propagation of multiple single-bit signals into a single word for bit-level parallel processing [16]. In the latest tools, parallel-pattern and event differentials strategies have been used to achieve significant performance improvements in the simulation of combinational circuits [17], [18]. However, for sequential circuits, the use of parallel-pattern or parallel-fault is limited, and further improvement in simulation performance is difficult.

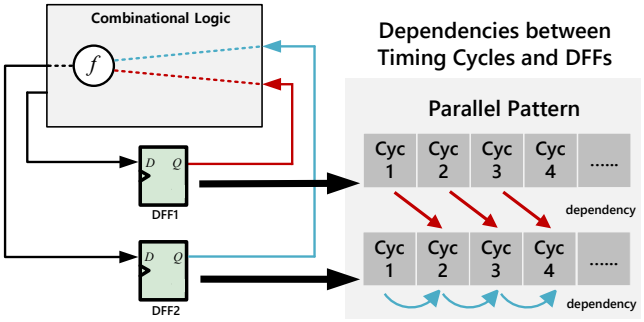


Fig. 1 Feedback Loop Example in Circuit.

This is because sequential fault simulation with parallel patterns or faults presents challenges. Placing parallel faults on different bits for parallel processing conflicts with concurrent algorithms that consolidate consistent events between fault occurrences. Parallel-pattern simulates inputs from different cycles on different bits. However, there exists a contradiction between parallel patterns and the timing dependencies. As illustrated in Fig. 1, the state of DFF 2 at cycle $n+1$ depends on the state of DFF 1 at cycle n and the state of DFF 2 at cycle n. This dependency makes it difficult to parallelize the simulation across different cycles. In large-scale sequential circuits, strongly connected components (SCCs), defined in graph theory as nodes that can mutually reach each other

along directed edges, further complicate this issue. The method proposed in [14] predicts the future states of DFFs for 64 cycles and then simulates them. However, due to potential prediction errors, all 64 cycles still need to be executed sequentially to determine the true values of all registers for each cycle. If the actual values of registers for a specific cycle match the predictions, subsequent pathways need not be executed. Nevertheless, updating the predicted future value of a DFF leads to simulations of all cells along the loop. In summary, the timing dependencies introduced by loops limit the performance improvements achievable with parallel patterns in sequential circuit fault simulation.

One insight is to compile and fuse loops into larger nodes, reducing simulation events caused by feedback. However, compiling loops into single nodes can prevent the detection of redundant events within the internal nodes, introducing excessive redundancy. Therefore, we opt to compile only simple loops. When traversing nodes in large SCCs, selecting loop entrance nodes is crucial, as it guides event propagation within the loop. If entrance nodes receive more feedback, prediction errors may increase, leading to greater simulation overhead. Additionally, allowing each node to receive inputs from preceding nodes directly can overly rely on the prediction accuracy of entrance nodes, increasing simulation costs. Therefore, another key insight is developing specialized algorithms for loop entrance selection and node updates in complex, large SCCs to reduce the simulation overhead associated with parallel patterns.

To this end, we propose a parallel-pattern framework that combines loop fusion with efficient event traversal, ultimately surpassing the performance of commercial tools. Initially, we employ a hybrid approach that integrates compiled fusion and event-driven methods within the circuit, merging shorter loop feedbacks within SCCs into single nodes to reduce the number of feedback events. For larger SCCs, we introduce loop entrance selection algorithms based on node indegree analysis and implement node lazy propagation algorithms. These algorithms, in conjunction with a guess-based parallel-pattern mechanism, significantly reduce the number of events propagated on loops caused by inaccurate predictions. We have realized these technologies in a system we call EPICS. Experimental results show that EPICS is faster than the latest commercial tool. Our contributions are summarized as follows:

- We propose a novel traversal framework for parallel-pattern simulation in sequential circuits, specifically designed for SCCs. Our method includes heuristic loop entry point selection based on structural analysis and a lazy propagation strategy for nodes, which reduces the overhead of false event propagation.
- We leverage the strengths of both compiled and event-driven simulations by strategically mixing them within the circuit, optimizing memory access and reducing scheduling overhead during the simulation process.
- Utilizing these techniques, we developed an efficient fault simulation framework for sequential circuits, achieving a 5.94× performance improvement compared to the commercial tool while maintaining the same fault coverage.

II. Background and Related Work

A. Temporal Correlation and Feedback Loops

The complexity of sequential circuits arises from state dependencies and feedback loops [19]. Unlike combinational circuits, the state of a sequential circuit depends on both current inputs and previous states, creating temporal correlations across multiple clock cycles and significantly increasing simulation complexity [20]. Feedback loops introduce cyclic dependencies, represented in graph theory as Strongly Connected Components (SCCs) [21], where any two nodes can reach each other through directed edges [22]. SCCs force simulators to iteratively update circuit states over multiple cycles, adding to the computational burden [23]. Algorithms such as Tarjan's [24] and Kosaraju's [25] are commonly used to identify SCCs, optimizing feedback loop handling and reducing redundant computations [26].

B. Fault Simulation in Sequential Circuits

Fault simulation is essential for verifying functional safety of sequential circuits and evaluating test pattern effectiveness [27], [28]. Stuck-at fault model is commonly used [29]. Due to the need to track state across multiple clock cycles, fault simulation in sequential circuits is inherently more complex than in combinational circuits [30]. Feedback loops further complicate this process, requiring the management of extensive state information and repeated computations [7], [31].

C. Parallel Simulation Techniques

Various parallel simulation techniques have been proposed to alleviate computational bottlenecks [32]. Parallel pattern single fault propagation (PPSFP) simulation method processes multiple input patterns simultaneously, leveraging hardware word length for speedup [14], [33]. While this method excels in combinational circuits [34], state dependencies and feedback loops in sequential circuits limit the achievable parallelism [35], resulting in diminished speedups [36]. Parallel fault simulation reduces redundant computations by simulating multiple faults concurrently [37], yet its efficiency is constrained by fault interdependencies and the overhead of managing large state information in complex circuits [38].

Classic simulators like PROOFS [15] and PARIS [14] employ different optimization strategies. PROOFS uses event-driven parallel fault simulation, focusing on sparse fault effects to minimize unnecessary computation [15], while PARIS is a parallel-pattern fault simulator [14]. Despite their successes in specific scenarios, both struggle with the performance demands of large-scale sequential circuits with intricate feedback loops. Later tools like HOPE [16] and VISION [33] enhance these approaches by optimizing fault dropping and state initialization, improving simulation efficiency.

D. Compiled and Event-Driven Simulation

Sequential circuit simulation typically uses compiled and event-driven methods [39], [40]. Compiled simulation translates circuit logic directly into machine code, eliminating interpretation overhead and improving speed, particularly in

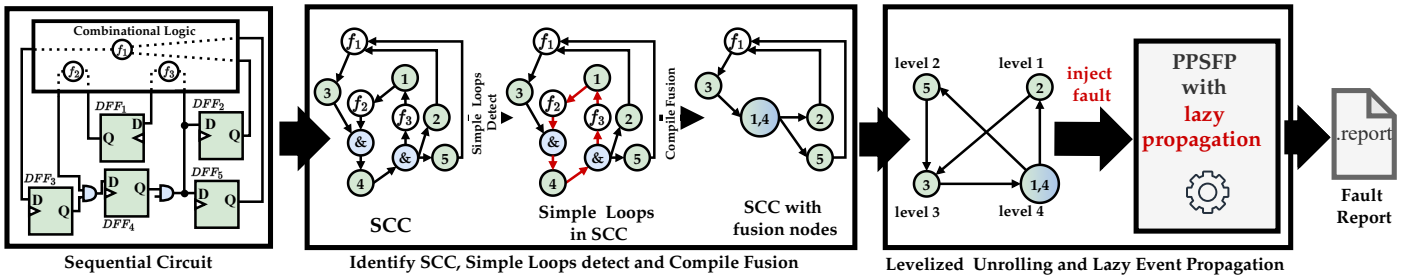


Fig. 2 EPICS Hybrid Fault Simulation Framework.

combinational circuits. However, the exponential growth of state space in sequential circuits due to flip-flops and feedback loops makes the approach less efficient, as it incurs significant compilation and storage overhead. Event-driven simulation triggers computations only when signal changes occur, reducing unnecessary calculations [41]. While effective in circuits with sparse changes, frequent event scheduling in circuits with many feedback loops can result in performance bottlenecks.

III. EPICS Framework

Our proposed framework, illustrated in Fig. 2, takes a gate-level circuit, input patterns, and a fault list as inputs to produce a fault simulation report. The framework consists of two main components: Firstly, we process the netlist by identifying the SCCs using Tarjan's algorithm [24]. For simple timing loops within these SCCs, we compile them into single nodes to reduce the number of simulation events. Secondly, for traversing large SCCs, we propose an indegree-based strategy to select the least dependent node as the entry point for speculative SCC traversal. Additionally, we introduce a lazy propagation strategy, where updates are deferred until upstream values confirm that the current node's prediction is incorrect. Finally, the framework outputs a fault coverage report.

IV. Compiled Simulation for Simple Timing Loops

In sequential circuit simulation, small timing loops typically consist of a few DFFs and simple combinational logic. Due to their short feedback paths and low dependency, compiled simulation is efficient in handling such loops. Based on SCC identification, we select small timing loops containing 1-2 DFFs and fewer combinational logic gates for compiled simulation. This approach integrates the combinational logic and DFF state transitions into a single functional composite node, avoiding event-driven simulation's frequent event scheduling overhead. By processing machine word length bit pattern data in parallel, the system simulates multiple states in a single event until the circuit reaches a stable state. Compiled simulation consolidates state changes over multiple clock cycles into one computation, significantly enhancing simulation efficiency.

As shown in Fig. 3, in the feedback loop, input a and b affect the states of DFFs A and B through logic gates C and D, and these state changes are fed back into the logic gates, forming a closed loop. Compiled simulation converts this loop into an equivalent instruction sequence, eliminating the need for frequent scheduling in event-driven simulations. State updates

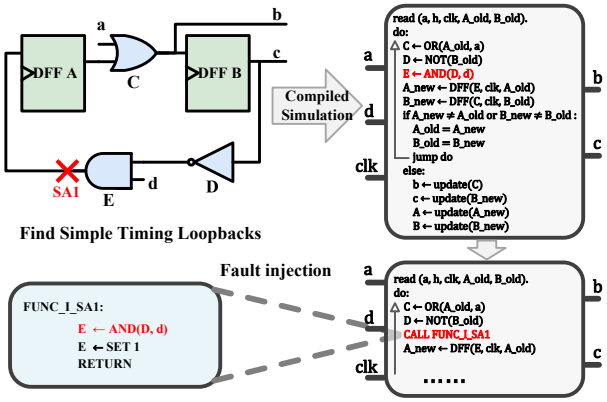


Fig. 3 Compiled Simulation and Fault Injection Example

and output computations for each clock cycle are handled in parallel until the feedback loop reaches a stable state.

Meanwhile, fault injection is achieved by modifying the compiled instruction sequence. For example, Fig. 3 shows a stuck-at-1 (SA1) fault injected at gate E, where the output of E is forced to 1, propagating the fault through the circuit, reducing memory access delays and improving fault simulation efficiency. Fault injection directly modifies the instruction sequence with minimal changes to the original instructions.

Compiled simulation is particularly well-suited for efficiently simulating small timing loops. Combined with fault injection techniques, this significantly reduces simulation time and avoids the cyclic scheduling issues of event-driven simulations while maintaining accuracy and improving performance.

V. Levelized Unrolling and Lazy Event Propagation

The loop simulation requires starting from an entry point, traversing the loop, and feeding the timing values back to the entry. Since the entry node guides the value updates for all nodes in the loop, the accuracy of the entry node's predicted value significantly impacts the number of simulations required. Therefore, we select the node with the highest prediction accuracy as the loop entry. Upon establishing the foundation of each loop, we propose a lazy propagation strategy at each node. To prevent significant redundant computations across all nodes on a loop due to inaccurate predictions at the entrance, each node refrains from readily accepting event propagation from higher levels until it can ascertain the presence of errors in subsequent predictions.

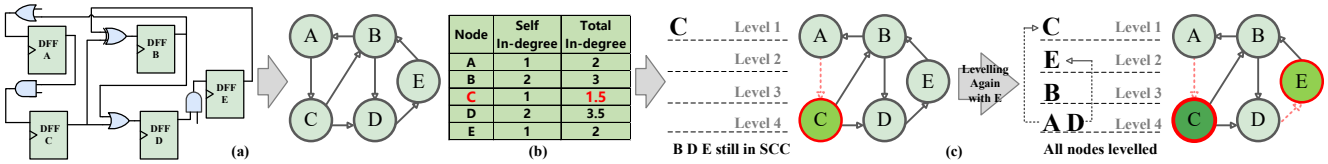


Fig. 4 Indegree-Based Levelized Unrolling of SCC.

A. *Indegree-based Levelized Unrolling of SCC*

Given the complexity of the circuit structure and connections, we propose a heuristic method based on analyzing the structure of the DFF graph. Through experimental analysis, we found that if a DFF depends on more sequential nodes—i.e., it has a higher indegree—its future cycle values become more uncertain and harder to predict. Conversely, a DFF with a lower indegree is more likely to generate future values based on its own past state, resulting in lower uncertainty.

Based on this observation, we propose Algorithm 1, which uses indegree analysis to identify efficient entry points for SCC simulation. Algorithm 1 consists of three main steps: calculating indegrees, selecting entry points, and then leveling the nodes based on the selected entry points. We illustrate this process using an example shown in Fig. 4.

Algorithm 1: Indegree-based Levelized Unrolling of Nodes in SCC

Input: SCC of a circuit graph
Output: Hierarchically unrolled node sequence L

1 Layer list $L \leftarrow \emptyset$, Processed node set $U \leftarrow \emptyset$,
2 Current layer $level \leftarrow 0$;
3 **foreach** $v \in SCC$ **do**
4 $\quad d_{\text{self}}(v) \leftarrow$ self indegree;
5 $\quad d_{\text{pred}}(v) \leftarrow$ predecessor indegree;
6 $\quad d_{\text{total}}(v) \leftarrow d_{\text{self}}(v) + \frac{1}{2} d_{\text{pred}}(v)$;
7 **while** $SCC \neq \emptyset$ **do**
8 $\quad v_{\min} \leftarrow \arg\min_{v \in \text{SCC}} d_{\text{total}}(v)$;
9 $\quad$ Cut input edges of $v_{\min}$;
10 $\quad$ **if** $L[level]$ *does not exist* **then**
11 $\quad\quad L[level] \leftarrow \emptyset$;
12 $\quad L[level] \leftarrow L[level] \cup \{v_{\min} \cup$ nodes with all predecessors in higher layers$\}$;
13 $\quad$ SCC $\leftarrow$ SCC $\setminus L[level]$;
14 $\quad U \leftarrow U \cup L[level]$;
15 $\quad$ **if** *cycles remain in SCC* **then**
16 $\quad\quad$ continue;
17 $\quad level \leftarrow level + 1$;
18 **return** L;

The first step (lines 4-5 of the algorithm) calculates the self indegree ($d_{\text{self}}(v)$) and predecessor indegree ($d_{\text{pred}}(v)$) for each node v in the SCC, and then computes the total indegree ($d_{\text{total}}(v)$) using the following formula:

$$d_{\text{total}}(v) = d_{\text{self}}(v) + \frac{1}{2} \sum_{u \in \text{Predecessors}(v)} d_{\text{self}}(u). \quad (1)$$

In Fig. 4(b), each node's calculated indegrees are shown. For instance, node C has a self-indegree of 1 and a total indegree of 1.5, making it a suitable candidate for the initial levelized unrolling step due to its lower dependency on other nodes.

Next, lines 7-10 of the algorithm select the node with the smallest total indegree-$v_{\min}$-and begin the unrolling process. As shown in Fig. 4(c), node C is selected first because it has the smallest total indegree. This node is then placed in Level 1 of the unrolling hierarchy, and its input edges are temporarily disconnected to break any cyclical dependencies.

The algorithm then places $v_{\min}$ and all other nodes whose predecessors are already assigned to higher layers into the current layer. This ensures that nodes dependent on already processed nodes are handled in a topologically valid unrolling.

If cycles still exist in the SCC after the initial unrolling (lines 13-14 of the algorithm), the process is repeated by selecting the next node with the smallest total indegree. In Fig. 4(c), nodes B, D, and E still form an SCC after processing C. The algorithm proceeds by selecting node E due to its lower total indegree and continues the unrolling process.

Finally, the current layer is incremented, and the algorithm proceeds until all nodes are unrolled, as illustrated in Fig. 4(c), where all nodes are assigned to respective levels. The result is a hierarchical node sequence that can be efficiently simulated while minimizing redundant computations.

Algorithm 2: Lazy Event Propagation

Input: Sequential circuit with DFFs, SCCs, initial states, and input stimuli
Output: Updated circuit states and simulation results

1 Initialize a lazy event table L to record lazy events;
2 Set current_time_point = 0;
3 **while** *Simulation is not complete* **do**
4 $\quad$ **for** *each DFF* v **do**
5 $\quad\quad$ **if** *Event triggered at current_time_point* **then**
6 $\quad\quad\quad$ Update state of v;
7 $\quad\quad\quad$ Propagate event to dependent nodes;
8 $\quad\quad\quad$ Clear lazy events of v from L;
9 $\quad\quad$ **if** *Lazy event exists for* v *at current_time_point in* L **then**
10 $\quad\quad\quad$ Propagate event to dependent nodes;
11 $\quad\quad\quad$ Clear lazy events of v from L;
12 $\quad$ **for** *each DFF* v **do**
13 $\quad\quad$ **if** v *is scheduled to change at a future time* t' **then**
14 $\quad\quad\quad$ Update state of v;
15 $\quad\quad\quad$ Record lazy event for v at time t' in L;
16 $\quad$ **if** *All events at current_time_point are processed* **then**
17 $\quad\quad$ Increment current_time_point;
18 **return** *Updated circuit states and simulation results*;

B. *Lazy Event Propagation*

Building on the hierarchical unfolding of SCCs, we propose a lazy propagation algorithm 2 aimed at reducing unnecessary state propagation and event scheduling. This strategy is particularly effective when managing complex SCCs and fault propagation. The core idea is to propagate events only when absolutely necessary, thus avoiding redundant computations.

The first part of algorithm 2 (lines 1-2) initializes a lazy event table, which records events that are postponed to be processed at an appropriate time. This event table helps the

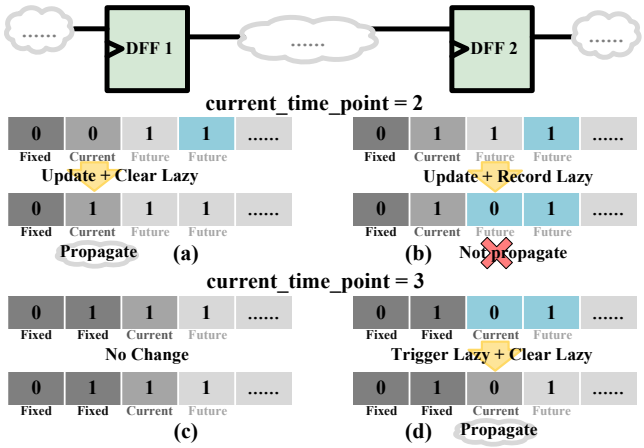


Fig. 5 Lazy event propagation in sequential circuits (lazy events are indicated in blue).

simulator efficiently manage which events need immediate processing and which can be delayed.

Figure 5(a) illustrates a scenario where DFF1 has an event triggered at current_time_point = 2. At this moment, DFF1's state is updated, and the event is propagated to dependent nodes. Simultaneously, DFF1's lazy event record is cleared to ensure no redundant computations occur. This behavior is described in lines 5-8 of the algorithm. If no event is triggered at a clock cycle (such as current_time_point = 3 in Figure 5(c)), the algorithm simply waits without making any state changes, thereby reducing unnecessary propagations and scheduling.

Lines 13-15 in algorithm 2 handle situations where state changes are scheduled to occur in the future. In Figure 5(b), at current_time_point = 2, DFF2 records a lazy event to be triggered at current_time_point = 3 and updates itself, but the event is not propagated immediately. This delayed processing helps avoid multiple unnecessary event propagates. When the future clock cycle arrives (as in current_time_point = 3 in Figure 5(d)), the lazy event is processed, DFF2's event is propagated forward, and DFF2's lazy event record is cleared. This process is described in lines 10-11, ensuring unnecessary propagations do not occur before the appropriate time.

Finally, once all events for a clock cycle have been processed, current_time_point increments, as described in line 17 of algorithm 2, and the simulator continues to process pending events for the next clock cycle. This design allows the simulator to perform simulations efficiently, minimizing computation while ensuring correctness.

VI. Experimental Results

A. Experimental Settings

We implemented EPICS in C++ and evaluated it on a Red Hat Enterprise Linux Server 7.9, equipped with an Intel Xeon® Platinum 8260 CPU @ 2.40GHz and 1.5TiB of RAM. For our evaluation, we compared EPICS against two other simulators that support gate-level fault simulation, representing both commercial and academic tools. The first is Z01X [42], a state-of-the-art commercial simulator. The second is VISION [33], a well-known academic simulator. Since the results reported for VISION in [33] were obtained on an outdated Sun Sparc 2 workstation, we reimplemented VISION to run on our modern hardware platform, resulting in a version we refer to as VISION+. This reimplemented version demonstrated significantly shorter simulation times than the original, owing to modern hardware advancements and optimizations. To ensure a fair and consistent comparison, we evaluated all simulators, including EPICS, Z01X, and VISION+, on the same platform and under the same settings.

TABLE I Statistics of Benchmark Circuits

Circuit	#Gates	#Cycles
b20	19,358	20,000
b22	28,929	20,000
b18	103,838	20,000
openC906	624,058	20,000
openC910	5,323,400	20,000
Comm1	78,739	28,286
Comm2	3,103,038	36,800
Comm3	4,300,720	23,480

Benchmark Circuits. We evaluated EPICS on a diverse set of benchmark circuits, including both academic and industrial designs, to assess its scalability and applicability. TABLE I summarizes the benchmarks circuits used in this paper, which cover small cores in academic papers ITC'99 [43] (*b20*, *b22*, *b18*), open-source RISC-V CPUs [44] (*openC906* , *openC910*), and commercial processors (*Comm1*, *Comm2*, *Comm3*). In our experiments, real functional patterns were used for commercial processors (*Comm1*, *Comm2*, *Comm3*), while random patterns were applied to other circuits. The simulation cycles for each circuit are also shown in TABLE I.

Fault list and fault coverage. We generated stuck-at faults for gates and DFFs in the circuits and set observation points at all outputs. The performance of each simulation was evaluated and the simulation runtime was reported. The fault coverage achieved by our proposed framework is consistent with that of the commercial tool Z01X across all benchmarks, demonstrating the correctness of the proposed fault simulation framework.

B. Evaluation on Benchmarks and Comparisons

We conducted performance tests on all the aforementioned benchmark circuits and compared the runtime of the proposed EPICS with that of the baseline methods. The experimental results are shown in Fig. 6, including the runtime of each algorithm and the corresponding speedup ratios. As observed, the proposed EPICS achieves the best performance across all benchmark circuits. For a small circuit b20 with 19,358 gates, the runtime of Z01X, VISION+, and EPICS are 405s, 172s, and 42s, respectively. For the commercial circuit Comm3 with 4,300,720 gates, the runtime of Z01X, VISION+, and EPICS are 82,968s, 117,314s, and 22,084s, respectively. For the largest circuit openC910 with 5,323,400 gates, the runtime of Z01X, VISION+, and EPICS are 26,904s, 23,906s, and 6,719s, respectively. On average, EPICS achieves a 5.94× speedup over Z01X, and a 4.23× speedup over VISION+ across all benchmark circuits. The experimental results demonstrate that the proposed EPICS exhibits excellent simulation performance

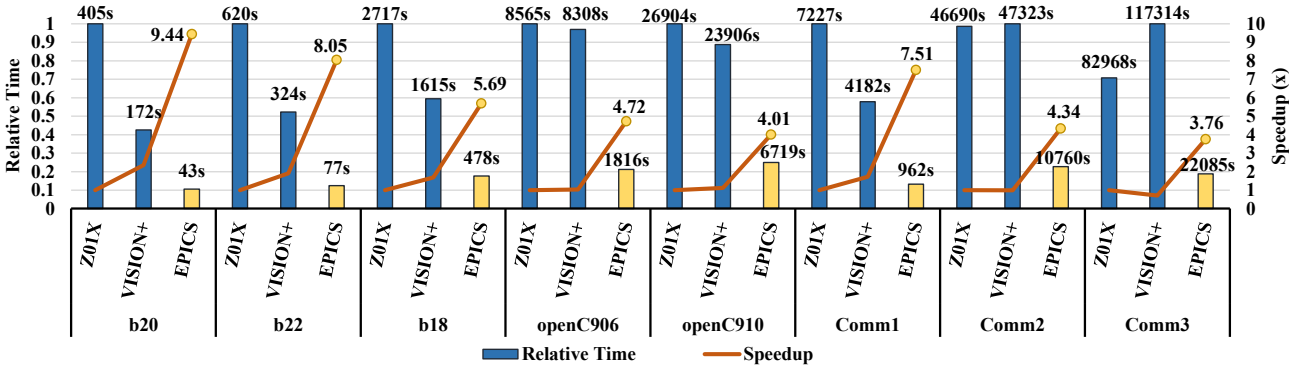


Fig. 6 Performance comparison of various simulators.

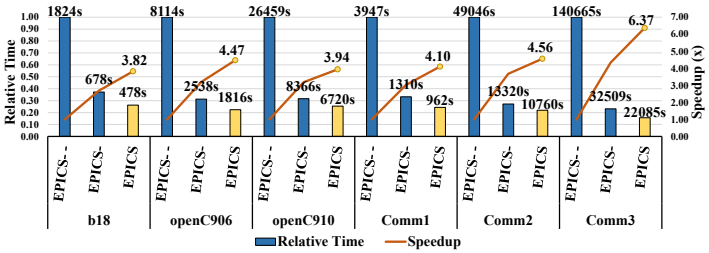


Fig. 7 Ablation Study on Optimizations: EPICS-- (No Optimization), EPICS- (Includes the SCC traversal strategy), EPICS.

across circuits of various scales, achieving significant acceleration compared to both the commercial tool and the state-of-the-art academic approach. Notably, EPICS reduces the simulation time for the commercial circuit Comm3 from approximately 22 hours with commercial tools to just 6 hours.

C. Ablation study

We conducted an ablation study to evaluate the acceleration effect of small loop compilation and SCC traversal strategy. We implemented another two versions of EPICS: EPICS--, which excludes both the small loop compilation and SCC traversal strategies, and EPICS-, which includes the SCC traversal strategy but excludes small loop compilation. Aside from these optimizations, both versions have the same implementation as the original EPICS. The results are listed in Fig. 7. It shows that EPICS- achieves 3.28× speedup over EPICS-- on average. Meanwhile, EPICS achieves 1.32× speedup over EPICS- on average, and 4.33× speedup over EPICS--. These indicate that both the SCC traversal strategy and the compilation of small loops contribute significantly to simulation acceleration.

TABLE II compares the total number of simulation events processed by each version. EPICS demonstrates a significant reduction in event count, which directly contributes to its reduced simulation time overhead. The compilation for small SCCs reduces event-driven overhead by eliminating the need for frequent event scheduling within these regions. As a result, EPICS processes 51% fewer events than EPICS-- in b18. The SCC traversal strategy is crucial in minimizing event count, especially in circuits with complex feedback loops. In Comm3, EPICS processes only 13.6% of the events required by EPICS--, demonstrating the efficiency of these combined strategies. On average, EPICS achieves a 47% reduction in event count compared to EPICS- and a 4.63× reduction compared to EPICS--, confirming the impact of its optimizations.

TABLE II Event Counts and Ratios Compared to EPICS

Circuit	EPICS Events	EPICS- Events	EPICS- Ratio(×)	EPICS-- Events	EPICS-- Ratio(×)
b20	**1.06e9**	1.31e9	1.24	3.75e9	3.55
b22	**1.71e9**	2.29e9	1.34	7.21e9	4.21
b18	**1.31e10**	1.98e10	1.51	5.51e10	4.20
openC906	**1.69e10**	2.48e10	1.47	8.02e10	4.74
openC910	**6.13e10**	9.49e10	1.55	2.60e11	4.24
Comm1	**1.04e10**	1.59e10	1.52	4.43e10	4.24
Comm2	**1.10e11**	1.62e11	1.48	4.95e11	4.50
Comm3	**2.38e11**	3.97e11	1.67	1.75e12	7.35
Average	-	-	1.47	-	4.63

Events: Number of simulation events generated.
Ratio (×): Event count ratio compared to EPICS.

The performance gains achieved by EPICS stem from the synergistic effect of its optimizations. Compiled simulation reduces event scheduling overhead in simple loops, while the SCC traversal strategy avoids unnecessary event propagation. Together, these optimizations enable EPICS to efficiently handle SCCs with minimal computational overhead.

VII. Conclusion

This paper introduced EPICS, an efficient framework for parallel-pattern fault simulation in sequential circuits, addressing the challenges of strongly connected components (SCCs). Combining compiled simulation, lazy event propagation, and hierarchical SCC unrolling, EPICS reduces computational overhead and enhances performance without sacrificing fault coverage. Experimental results show that EPICS achieves 5.94× speedup over commercial tool Z01X while maintaining equivalent fault coverage, demonstrating its scalability for large-scale industrial designs.

Acknowledgement

This paper is supported in part by the National Natural Science Foundation of China (NSFC) under grant No. (92373206, 92473203). The corresponding authors are Jianan Mu and Jing Ye.

REFERENCES

[1] F. J. Belmonte, S. Martín, E. Sancristobal, J. A. Ruipérez-Valiente, and M. Castro, "Overview of embedded systems to build reliable and safe adas and ad systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 13, no. 4, pp. 239–250, 2020.

[2] G. Griessnig and A. Schnellbach, "Development of the 2nd edition of the iso 26262," in *Systems, Software and Services Process Improvement: 24th European Conference, EuroSPI 2017, Ostrava, Czech Republic, September 6–8, 2017, Proceedings 24*. Springer, 2017, pp. 535–546.

[3] W. Chen and J. Bhadra, "Practices and challenges for achieving functional safety of modern automotive socs," *IEEE Design & Test*, vol. 36, no. 4, pp. 31–47, 2019.

[4] K. Suo, L. Vu, M. R. Islam, N. Dhar, T. N. Nguyen, S. He, and X. Wu, "A systematic investigation of hardware and software in electric vehicular platform," in *Proceedings of the 2024 ACM Southeast Conference*, 2024, pp. 9–17.

[5] W. Granig, D. Hammerschmidt, and H. Zangl, "Calculation of failure detection probability on safety mechanisms of correlated sensor signals according to iso 26262," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 10, no. 2017-01-0015, pp. 144–155, 2017.

[6] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI test principles and architectures: design for testability*. Elsevier, 2006.

[7] A. Miczo, *Digital logic testing and simulation*. John Wiley & Sons, 2003.

[8] M. M. Peng and J. S. Kuang, "A gpu-based fault simulator for small-delay faults," *Advanced Materials Research*, vol. 753, pp. 2235–2242, 2013.

[9] R. Baranowski, S. Di Carlo, N. Hatami, M. E. Imhof, M. A. Kochte, P. Prinetto, H.-J. Wunderlich, and C. G. Zoellin, "Efficient multi-level fault simulation of hw/sw systems for structural faults," *Science China Information Sciences*, vol. 54, pp. 1784–1796, 2011.

[10] M. Wang, H. Wang, J. Mu, Z. Liu, J. Gao, J. Ye, H. Li, and X. Li, "Efficient functional safety method for gate-level fine-grained digital circuits with iso-26262," in *2024 IEEE International Test Conference in Asia (ITC-Asia)*. IEEE, 2024, pp. 1–6.

[11] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham, "Vlsi logic and fault simulation on general-purpose parallel computers," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 12, no. 3, pp. 446–460, 1993.

[12] M. Maddela, "Elec 7250 sequential parallel fault simulator."

[13] M. Abramovici, M. A. Breuer, and K. Kumar, "Concurrent fault simulation and functional level modeling," in *Proceedings of the 14th Design Automation Conference*, 1977, pp. 128–137.

[14] N. Gouders and R. Kaibel, "Paris: A parallel pattern fault simulator for synchronous sequential circuits," in *1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers*. IEEE, 1991, pp. 542–545.

[15] T. M. Niermann, W.-T. Cheng, and J. H. Patel, "Proofs: A fast, memory-efficient sequential circuit fault simulator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 2, pp. 198–207, 1992.

[16] H. K. Lee and D. S. Ha, "Hope: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.

[17] Synopsys, *TestMAX ATPG and TestMAX Diagnosis User Guide*, Synopsys, July 2021.

[18] Mentor, *Tessent® Scan and ATPG User's Manual*, document revision 18 ed., Mentor, A Siemens Business, 2020.

[19] K.-T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 544–548, 1990.

[20] N. Miskov-Zivanov and D. Marculescu, "Multiple transient faults in combinational and sequential circuits: A systematic approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1614–1627, 2010.

[21] J. F. Hurford and T. C. Hartrum, "Improving conservative vhdl simulation performance by reduction of feedback," *ACM SIGSIM Simulation Digest*, vol. 26, no. 1, pp. 196–201, 1996.

[22] H. He, T. Xu, J. Chen, Y. Cui, and J. Song, "A granulation strategy-based algorithm for computing strongly connected components in parallel," *Mathematics*, vol. 12, no. 11, p. 1723, 2024.

[23] D. V. Overhauser, *Fast timing simulation of MOS VLSI circuits*. University of Illinois at Urbana-Champaign, 1989.

[24] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[25] J. E. Hopcroft, J. D. Ullman, and A. V. Aho, *Data structures and algorithms*. Addison-wesley Boston, MA, USA:, 1983, vol. 175.

[26] C. A. Larsen, S. M. Schmidt, J. Steensgaard, A. B. Jakobsen, J. v. de Pol, and A. Pavlogiannis, "A truly symbolic linear-time algorithm for scc decomposition," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2023, pp. 353–371.

[27] A. H. Khan, Z. H. Khan, and Z. Weiguo, "Model-based verification and validation of safety-critical embedded real-time systems: Formation and tools," in *Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles*. Springer, 2013, pp. 153–183.

[28] T. Ahmed *et al.*, "Evaluation of fibonacci test pattern generator for cost effective testing," 2013.

[29] S. Mourad and E. J. McCluskey, "Fault models," in *Testing and Diagnosis of VLSI and ULSI*. Springer, 1988, pp. 49–68.

[30] K. Cho and R. E. Bryant, "Test pattern generation for sequential mos circuits by symbolic fault simulation," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, 1989, pp. 418–423.

[31] M. Mojtahedi and W. Geisselhardt, "New methods for parallel pattern fast fault simulation for synchronous sequential circuits," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. IEEE, 1993, pp. 2–5.

[32] H.-P. Chang, S. G. Chappell, C. H. Elmendorf, and L. D. Schmidt, "Comparison of parallel and deductive fault simulation methods," *IEEE Transactions on Computers*, vol. 100, no. 11, pp. 1132–1138, 1974.

[33] R. Nair and D. S. Ha, "Vision: An efficient parallel pattern fault simulator for synchronous sequential circuits," in *Proceedings 13th IEEE VLSI Test Symposium*. IEEE, 1995, pp. 221–226.

[34] D. Harel, R. Sheng, and J. Udell, "Efficient single fault propagation in combinational circuits," *Communications of the ACM*, vol. 29, no. 4, pp. 300–311, 1986.

[35] C.-P. Kung and C.-S. Lin, "Parallel sequence fault simulation for synchronous sequential circuits," *Journal of Electronic Testing*, vol. 9, pp. 267–277, 1996.

[36] J. Kõusaar, R. Ubar, S. Kostin, S. Devadze, and J. Raik, "Parallel critical path tracing fault simulation in sequential circuits," in *2018 25th International Conference" Mixed Design of Integrated Circuits and System"(MIXDES)*. IEEE, 2018, pp. 305–310.

[37] E. Rudmocl and J. H. Patel, "Overcoming the serial logic simulation bottleneck in parallel fault simulation," in *Proceedings Tenth International Conference on VLSI Design*. IEEE, 1997, pp. 495–501.

[38] E. M. Rudnick, T. M. Niermann, and J. H. Patel, "Methods for reducing events in sequential circuit fault simulation." in *ICCAD*. Citeseer, 1991, pp. 546–549.

[39] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2014, pp. 1–6.

[40] B. L. Keller, D. P. Carlson, and W. Maloney, "The compiled logic simulator," *IEEE Design & Test of Computers*, vol. 8, no. 1, pp. 21–34, 1991.

[41] B. Becker, M. Keim, and R. Krieger, "Hybrid fault simulation for synchronous sequential circuits," *Journal of Electronic Testing*, vol. 15, pp. 219–238, 1999.

[42] Synopsys, Inc., *Z01X™ Simulator Manufacturing Assurance User Guide*, Synopsys, Inc., Sep. 2023, copyright © 2023 Synopsys, Inc.

[43] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers*, vol. 17, no. 3, pp. 44–53, 2000.

[44] "OpenXuantie," 2021, [Online]. Available: https://github.com/T-head-Semi/.

PastATPG: A Hybrid ATPG Framework for Better Test Compaction with Partial Assignment SAT

Zhiteng Chao*[1,3,4], Xindi Zhang*[2], Xinyu Zhang[1,3], Jianan Mu†[1], Zizhen Liu[1], Shengwen Liang[1], Shaowei Cai†[2], Jing Ye[1,3,4], Xiaowei Li[1,3,4], Huawei Li†[1,3,4]

[1]*State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences*

[2]*Key Laboratory of System Software (CAS) and State Key Laboratory of Computer Science, Institute of Software, CAS*

[3]*University of Chinese Academy of Sciences*

[4]*CASTEST Co., Ltd.*

chaozhiteng20g@ict.ac.cn, zhangxd@ios.ac.cn, {zhangxinyu23s,mujianan,liuzizhen,liangshengwen}@ict.ac.cn, caisw@ios.ac.cn, {yejing, lxw, lihuawei}@ict.ac.cn

***Abstract*—In automatic test pattern generation (ATPG), SAT-based methods are typically used to complement structural approaches, especially for addressing hard-to-detect faults. However, as the size and complexity of circuits grow, SAT-based ATPG faces challenges like pattern inflation and excessive runtime, limiting its overall performance. The key problem lies in the fact that current mainstream SAT solvers perform complete assignments for all primary inputs of the fault's transitive fanin cone without considering the detection of other faults, making test compaction extremely difficult and time consuming. In this paper, a novel SAT solver PA-MiniSat is proposed, which is capable of generating partial assignments for solving variables and significantly reduces the number of specified bits in test cubes. As an extension of MiniSat, it employs a full-literal watching technique and a circuit-adapted heuristic branching strategy, achieving overall improved performance in ATPG. Based on PA-MiniSat, a hybrid ATPG framework PastATPG is proposed for better test compaction, which tightly integrates structural algorithms with the SAT solver into the unified test compaction flow. Experimental results demonstrate that our method outperforms other SAT solvers in pattern compaction and, in some cases, even surpasses commercial ATPG tools in terms of speed. The code is available at https://github.com/sklp-eda-lab/PastATPG.**

I. Introduction

With the growing complexity of integrated circuits, identifying deeply embedded faults becomes increasingly challenging for automatic test pattern generation (ATPG), leading to a rise in pattern count and a decline in fault coverage. For hard-to-detect faults, traditional structural algorithms like the D-algorithm [1] and PODEM [2] often struggle to generate test patterns within a limited time. After running these structural methods with limited backtracks, the remaining single-fault test generation problems are typically encoded into conjunctive normal form (CNF) and solved using SAT solvers [3]–[10].

Although SAT ATPG exhibits strong fault detection capabilities and guarantees completeness, it often generates a significant number of patterns to detect a small subset of hard-to-detect faults, leading to a sharp increase in testing costs. The root cause of pattern explosion is the difficulty in compacting the test cubes generated by SAT solvers. Test compaction [11]–[13] is a common component in structural ATPG, with the fundamental idea being to use a single pattern to detect multiple faults by attempting to merge the test cubes generated for a set of faults. However, since the SAT solver only considers the transitive fanin cone for individual faults and assigns values to all primary inputs within that cone, while lacking awareness of the control values of logic gates, these factors result in redundancy. It becomes challenging for test compaction algorithms to effectively reduce the number of patterns. Fig. 1 illustrates the test cubes generated by the SAT solver for two faults. For the faults, f_1, f_2, presented in the circuit on the left, the SAT solver produces the cubes *10010* and *X1111*, respectively. *X* represents an unspecified bit, which can be either 0 or 1. However, when considering the case with minimal assignments to the primary inputs, we observe that the test cubes *10XXX* and *XXXX1* together are capable of detecting both faults and can be merged into a single pattern, *10XX1*. This aspect presents a challenge for the SAT solver to address during the solving process. To enhance the performance of test compaction, it is essential to maximize the number of unspecified bits (X bits) for test cubes. To the best of our knowledge, no existing research has effectively addressed the issue of excessive specified bits in the test cubes generated during the SAT engine's solving process. This challenge complicates the integration of SAT-based ATPG into the test compaction frameworks commonly utilized in structural methodologies.

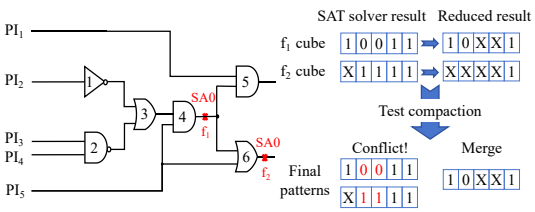


Fig. 1: An example of obstacles to test compaction caused by current mainstream SAT solvers in generating test cubes.

This paper proposes a hybrid ATPG framework, PastATPG, which integrates SAT-based ATPG and structural methods into a unified test compaction approach. To enable more efficient test compaction, a SAT solver with the ability to generate partial

*Equal contribution. †Corresponding authors.

assignments for variables, PA-MiniSat, is introduced. To the best of our knowledge, PA-MiniSat is the first SAT solver in the world capable of generating as many unspecified bits in test cubes as possible without introducing additional CNF encoding overhead. The main contributions are summarized as follows:

- We extended MiniSat [14] to develop PA-MiniSat, a partial assignment SAT solver that maximizes unspecified variables, significantly reducing the specified bits in test cubes. With a circuit-adapted heuristic branching strategy, test generation efficiency is improved compared with the original MiniSat solver in some cases.
- We design a hybrid framework which tightly integrates structural algorithms and the SAT solver into the unified test compaction flow with sufficient unspecified bits (X bits) PA-MiniSat provides.
- Experimental results indicate that PastATPG, integrated with PA-MiniSat, produces more compact patterns compared to other mainstream SAT solvers. In some cases, it also demonstrates a speed advantage, even surpassing commercial ATPG tools.

II. Preliminaries

A. ATPG System Overview

SAT based ATPG.

SAT-based ATPG leverages efficient SAT solvers to address ATPG problems by converting them into a conjunctive normal form (CNF) model, which represents the fault conditions and signal propagation between primary inputs (PIs) and primary outputs (POs). A CNF formula consists of clauses that are disjunctions of literals, with each literal being a variable or its complement, assigned a logic value of 0 or 1. The SAT solver either finds an assignment that satisfies the CNF formula or proves that no such assignment exists (UNSAT). Modern SAT solvers are powered by the conflict-driven clause learning (CDCL) algorithm [15], widely used in various applications.

In the context of ATPG, a fault is detectable if at least one bit in the faulty circuit's POs differs from the corresponding bit in the good circuit, for a given input pattern. If all outputs match, the fault is undetected. SAT-based ATPG transforms both the faulty and good circuits into CNF models, linking them through an XOR gate at the outputs to detect differences. In this setup, solving SAT corresponds to finding a test cube, while UNSAT proves a fault is untestable. For instance, in a stuck-at-fault scenario, the affected sub-circuit is extracted as shown in Fig. 2, and the circuit structure is then converted into CNF, where each signal path is represented by boolean variables and constraints.

Traditional SAT ATPG is typically used as a supplement to structural ATPG frameworks, as shown in Fig. 3. In the first phase, structural ATPG attempts to generate test patterns for all faults within a specified number of backtracks. Faults that exceed the backtrack limit are marked as ATPG aborted and passed to the second phase, where SAT ATPG either generates test patterns for the remaining hard-to-detect faults or determines them as untestable.

Test Compaction.

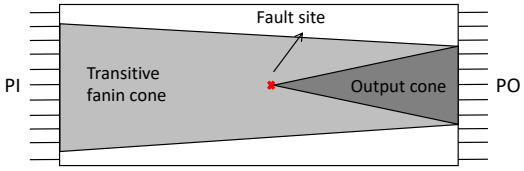


Fig. 2: Extraction of the relevant circuit parts: the shaded region of the circuit is encoded as the good circuit, while the darker shaded area representing the output cone is encoded again using different variables as the bad circuit.

Test compaction techniques are applied in structural ATPG to reduce the number of test patterns, categorized into static and dynamic compaction [16], [17]. Static compaction attempts to merge multiple test cubes, while dynamic compaction uses the remaining unspecified bits (X bits) in a test cube to generate tests by iteratively invoking the ATPG engine for other faults without affecting the previously assigned bits. The test compaction techniques used in structural ATPG are difficult to directly apply to SAT ATPG due to the excessive number of specified bits in its test cubes, making compaction challenging.

In [11], [13], dynamic compaction was introduced into SAT ATPG, but it did not address the core issue of excessive specified bits in SAT ATPG test cubes. Another challenge associated with the application of dynamic compaction in SAT ATPG is its extremely slow speed, as it necessitates invoking the SAT solver at each iteration. Moreover, when applied solely to hard-to-detect fault sets, the high frequency of solver failures results in substantial waste of computational resources.

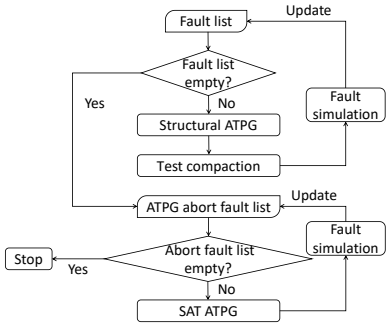


Fig. 3: In the traditional ATPG framework, the structural method is performed first, with the SAT solver applied to faults that result in aborts.

B. Related Work

In [3], SAT solvers were firstly applied for generating test patterns for stuck-at faults. TranGen [4] introduced an efficient error path pruning technique for the transition fault model, effectively identifying the longest sensitization path for each fault. PASSAT [5] incorporated multi-valued logic encoding to handle unknown and tri-state values, addressing more complex circuit design needs. TG-Pro [9] improved ATPG efficiency by modifying the semantics of sensitization variables and introducing new fault propagation constraints, thereby reducing

the number of variables. In [10], SAT ATPG was accelerated through CNF preprocessing and designing new conflict management heuristics. TIGUAN [8] proposed a two-phase parallel SAT ATPG framework.

Previous methods have sought to optimize SAT ATPG test cubes. For instance, PASSAT introduced X bits via multi-variable encoded CNF, but this increased CNF size and impacted performance without reducing specified bits in test cubes. In [18], local justification analysis was used to relax primary inputs, but integrating this with SAT ATPG is slow due to SAT ATPG's lack of necessary information. In [19], a PBO-based static compaction method added post-ATPG required extensive preprocessing and overhead. Multiple-target test generation (MTTG) [20]–[23] generated single test cubes for related faults, but selecting compatible fault groups is challenging and adds overhead with high failure rates.

III. PastATPG: A Hybrid ATPG Framework with Unified Test compaction Leveraging PA-MiniSat

A. Overview

We propose a hybrid and high-performance ATPG framework, PastATPG, as outlined in Fig. 4. By designing a partial assignment SAT solver named PA-MiniSat, previously structural test compaction techniques can now be directly applied to the test cubes generated by SAT ATPG. We designed a unified test compaction framework as shown in the dashed box in Fig. 4. For each fault processed in dynamic compaction, a

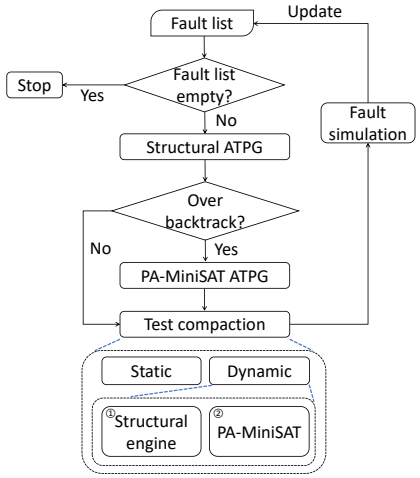


Fig. 4: Overview of PastATPG.

structural algorithm engine is first invoked. After reaching the backtrack limit, the SAT solver is then called. To enhance processing speed, we employ a circuit-adaptive branching strategy to select new assignment variables, which enables a measurable improvement in acceleration. We chose to extend MiniSat due to its high performance, lightweight design, ease of integration, and strong extensibility. MiniSat often demonstrates excellent performance on CNF encodings of circuit problems, making it an ideal foundation for our work.

B. PA-MiniSat Design

SAT is an effective tool but is generally time-consuming in most cases. Calling the SAT solver once for a fault, we can obtain a test cube in which all primary inputs within the fault's transitive fanin cone are assigned specific values. However, suppose SAT solvers can generate a partial assignment solution, i.e., with 'X' values. In that case, we can maintain the efficiency of the SAT-based ATPG method while also providing a test cube easy to compact.

The CDCL Framework and 2-literal Watching.

Modern SAT solvers are based on the CDCL framework, and the main reasoning method is boolean constraint propagation (BCP). BCP iteratively finds unit clauses (clauses with all falsified literals except for one unassigned literal) and assigns the unassigned variables until there is no unit clause or a conflict is driven. Here, "falsified" refers to assigning values of 0 to positive literals and 1 to negative literals. For "satisfied", the assignments are vice versa. All the mainstream SAT solvers cannot provide models with 'X' values because their internal BCP implementations are based on 2-literal watching [24], which is a popular clause management technique and will be explained later. To fetch partial assignments from SAT solvers, we proposed to use full-literal watching and introduced a counter for unsatisfied original clauses.

Algorithm 1: PA-MiniSat Framework

Input: ACNF formula F
Output: 'UNSAT' or assignment with 'X' value.

1 $\alpha := \emptyset$; # assigned variable trail
2 $_{pa:}$ $n := nClauses(F)$; # unsatisfied clauses number
3 **while** $True$ **do**
4 $\quad (conf, \alpha') \leftarrow BCP(\alpha)$;
5 $\quad _{pa:}$ $n \leftarrow n - upd_satisfied(\alpha' \setminus \alpha)$;
6 $\quad \alpha \leftarrow \alpha'$;
7 $\quad$ **if** $conf \neq NULL$ **then**
8 $\quad\quad pos \leftarrow analyze()$;
9 $\quad\quad \alpha \leftarrow \alpha[0 : max(0, pos)]$; # backtrack;
10 $\quad\quad _{pa:}$ $n \leftarrow n + upd_falsified(\alpha[max(0, pos) :])$;
11 $\quad\quad$ **if** $pos < 0$ **then return** 'UNSAT';
12 $\quad$ **else**
13 $\quad\quad$ ~~$_{org:}$ **if** no unassigned variable **then return** α;~~
14 $\quad\quad _{pa:}$ **if** $n = 0$ **then return** α;
15 $\quad\quad lit \leftarrow branching()$; $\alpha \leftarrow \alpha \cup \{lit\}$;
16 $\quad\quad _{pa:}$ $n \leftarrow n - upd_satisfied(lit)$;

To better understand our method, we need to introduce the necessary parts of CDCL and 2-literal watching. A typical CDCL is shown in Algorithm 1 (ignoring lines marked $_{pa:}$). CDCL maintains a trail α for currently assigned literals (positive literals mean variables assigned true and vice versa), which is empty at the beginning (line 1). In the main loop, CDCL performs BCP based on the current α, which can be updated to α' with some newly assigned literals $\alpha' \setminus \alpha$ (lines 4, 6). If there is a conflict in BCP (a variable assigned true and

false at the same time) (line 7), then CDCL will analyze it, getting a learnt clause and a backtrack position pos (line 8). The variables' assignment after pos in the trail will be canceled (line 9). Further, if the conflict is from the original clauses, CDCL returns 'UNSAT' (line 11). On the other hand, if BCP fails to find conflicts (line 12), CDCL first checks whether all the variables have been assigned and returns the full assignment if so (line 13). Otherwise, CDCL will pick a new unassigned variable, choose the assignment, and update the trail (line 15).

During the BCP process, a typical CDCL scans the clauses containing the negation of the newly assigned literal to check whether there are new unit clauses or conflicts. Modern SAT solvers use the 2-literal watching technique to scan clauses. For each clause, only two unassigned literals are watched at the beginning; and with the updation of α, a falsified watched literal is updated to another unassigned literal (if it exists). The references of a clause can only be reached from the watched two literals when BCP. The advantage is that it can reduce some unnecessary clause visiting, and the watch table does not need to be updated when backtracking. But with this lazy data structure, we cannot dynamically update and quickly count the exact number of satisfied clauses, other than completely scan through all the clauses. A small example is presented below to illustrate the implementation details of the 2-literal watching.

Consider the clause:

$$C = (x_1 \vee x_2 \vee \neg x_3 \vee x_4)$$

Initially, we choose to watch the literals x_1 and x_2, and the clause C is satisfied. If x_1 is assigned the value *false*, the solver does not immediately traverse the entire clause. Instead, it lazily checks x_2 to determine whether the clause can still be satisfied. If x_2 is *true* or unassigned, no further action is needed, and the clause remains satisfied. Only when x_2 is also assigned the value *false* does the solver lazily check the remaining literals, $\neg x_3$ and x_4, to find a new literal to watch or to propagate an assignment. Although the 2-literal watching based on the lazy data structure is efficient, it cannot produce partial assignments.

Full-literal Watching Technique.

To address this issue, we propose to use the full-literal watching technique. In comparison, it is an earlier and simpler technique. In the full-literal watching model, every literal in a clause is monitored. Whenever the value of any literal changes, the solver re-evaluates the entire clause to determine if BCP is necessary. Algorithm 1 presents the pseudocode, with distinctions from the traditional approach marked by $_{pa:}$. We maintain a global counter for unsatisfied clauses (falsified and undecided clauses) n (line 2), and it returns current α if $n = 0$ under α (line 14). Simultaneously, we establish clause counters for each clause to track the number of satisfied literals, to monitor the satisfiability changes. For each clause, its reference can be accessed by all the internal literals. Upon assigning a value to a variable, it is necessary to examine the associated clauses that include this variable, and the variable n should be decremented by the number of clauses that have now become satisfied (lines 5, 16). Conversely, when a variable's assignment is canceled, n should be incremented by the number of clauses that have now become undecided (line 10). Moreover, if all the clauses, in which a variable v appears, are still satisfied by cancel v's assignment, then v can be assigned with 'X'.

The example in Fig. 5 illustrates how the full-literal watching technique can yield partial assignments with X bits. In this example, we assume that BCP is not taken into consideration. For a three-input AND gate, the CNF is encoded into four clauses: $\{C_1, C_2, C_3, C_4\}$ involving four variables: $\{x_1, x_2, x_3, x_4\}$. The branching order for variable assignments is from x_1 to x_4. The lower part of the figure shows tables for both 2-literal watching and full-literal watching techniques, detailing the clauses monitored by each variable. In the 2-literal watching technique, each clause can appear in the table a maximum of two times, allowing full-literal watching for x_1 and x_2 to monitor an additional clause, C_4. Suppose the current partial assignment is $\alpha : \{x_1 = 1, x_2 = 0\}$. If C_1 is not satisfied, a backtrack occurs, changing the assignment to $\alpha' : \{x_1 = 0, x_2 = 0\}$. At this point, with 2-literal watching, the lazy data structure requires that all variables be fully assigned to determine whether all clauses are satisfied. In contrast, with full-literal watching, it is already possible to check that the current assignment satisfies all clauses. Thus, the algorithm can return the partial assignment result, yielding $\{x_1 = 0, x_2 = 0, x_3 = X, x_4 = X\}$.

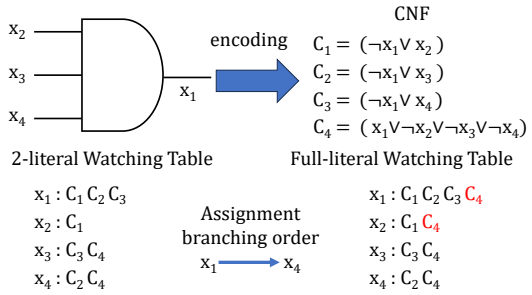


Fig. 5: The difference between 2- and full-literal watching.

With our full-literal watching method, CDCL is capable of yielding an assignment with 'X' values, which can help compact the test cubes better. However, the full-literal scans more clauses, leading to increased time consumption: First, our method scans clauses and updates counters not just throughout the BCP reasoning process, but also during the backtracking process. Secondly, in the process of scanning a literal's watched table, all clauses in which the literal appears should be scanned, which is the worst case of 2-literal watching.

To improve the efficiency, we only use the full-literal watching method for the original clauses. The original clauses are usually modeled short, which narrows the gap of scanning times between the two methods. The number of learnt clauses increased rapidly, and most of them are long clauses. As learnt clauses are implications of original clauses, i.e., learning clauses have no relation to evaluating whether the current assignment α satisfies all clauses. Thus, we keep the 2-literal watching method for learnt clauses.

Circuit-adapted Heuristic Branching Strategy.

In SAT solvers, *branching* involves selecting a variable to assign a truth value, crucial for guiding the search efficiently. The VSIDS heuristic [25] aids this process by assigning each variable a dynamic score. The key steps are:

- **Initialization**: All variables are initially assigned an activity score, typically set to zero.
- **Conflict-Based Updates**: After a conflict, variables involved in the conflict have their scores increased, prioritizing them for future branching.
- **Exponential Decay**: Scores are periodically reduced to give more weight to recent conflicts, preventing overemphasis on past ones.
- **Selection**: The variable with the highest score is chosen for branching.

The original VSIDS heuristic in MiniSat is based solely on conflict clauses during solving. We modify it by incorporating information from the circuit structure in the structural ATPG process to guide initialization. Each variable receives an initial score based on its gate's position within the circuit, creating a circuit-adaptive heuristic. As shown in Fig. 2, ATPG detects faults by propagating fault values through the output cone of the faulty circuit while justifying values in the good circuit. We sort variables by the logical depth of their corresponding gates, with the bad circuit sorted from input to output and the good circuit in reverse order. This sequence is mapped to the interval $[threshold, 0]$, with the threshold set to 10.

IV. Experimental Results

A. Experimental Setup

The structural ATPG components are derived from an open source ATPG program [26]. We compare PA-MiniSat with three popular SAT solvers: two solvers that have won championships in global SAT competitions CaDiCaL, Kissat, [27] and the original version of MiniSat [14]. Our benchmarks consist of combinational circuits, including large-scale circuits from ISCAS 85, ISCAS 89, and ITC 99, as well as an industrial circuits e203 [28], a RISC-V CPU core. To ensure a fair comparison, all circuits were synthesized into netlists using Design Compiler and subsequently processed by a well-known commercial ATPG tool C to generate fault lists. Our program runs on a Linux machine equipped with Intel Xeon processor.

B. Performance Evaluation

We conduct three experiments to validate the effectiveness of the proposed method. The first two experiments evaluate the test compaction effects and time overhead of pure PA-MiniSat SAT ATPG, while the latter one assess the performance of the whole hybrid framework PastATPG. To align with real-world scenarios applications, the first two experiments are performed on hard-to-detect fault sets, specifically faults with runtime exceeding the structural ATPG threshold. Due to the excessive time consumption associated with the invocation of pure SAT solvers for dynamic test compaction, in the second experiment we only run 30% of the hard-to-detect faults for benchmarks b17, b20, b21, and b22, while for b19 and e203, the limit is 10%. The final experiment is conducted on the entire fault set to evaluate the performance of the framework under real ATPG applications. The last two experiments are conducted on the complete ATPG system incorporating commonly used test compaction techniques from structural algorithms into SAT ATPG, including static [12] and dynamic [13] compaction methods, but differ in the configuration for fault simulation. In the second experiment, we eliminated the random 0/1 bit-filling for the final test cubes to exclude interference from random pattern fault simulation.

Evaluation of X bits Generation Effectiveness.

Fig. 6 illustrates the effectiveness of X bits generation by PA-MiniSat. For hard-to-detect faults of different circuits, we examine the proportion of X bits genetared by PA-MiniSat in the inputs of the transitive fanin cone for each fault's test cube without test compaction. It can be observed that the proportion of X bits varies significantly across different circuits. Overall, compared to a fully assigned SAT solver, our PA-MiniSat is able to generate, on average, 35.8% of the X bits for the transitive fanin cone portion of each test cube.

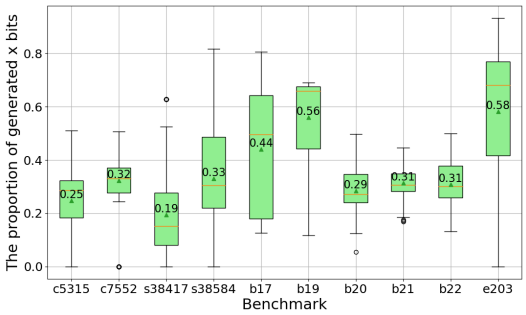


Fig. 6: The figure presents box plots illustrating the proportion of X bits in the inputs of the transitive fanin cone within the test cubes generated by the PA-MiniSat solver for various hard-to-detect fault sets in circuits; each box plot shows the median, interquartile range (IQR), and potential outliers (denoted by hollow circles), while the numbers indicate the average.

Evaluation of Solver Performance.

Left half of TABLE I presents the experimental results of the hard-to-detect fault set, a full SAT-only ATPG process is conducted with consistent test compaction parameters: the test cube bucket size for static compaction is set to 32, and the conflict limit for dynamic compaction is 100. TABLE I also shows the results without test compaction techniques in the right half. The results show that under test compaction, PA-MiniSat significantly reduces the number of patterns compared to other SAT solvers. In terms of time overhead, PA-MiniSat achieves significant acceleration on circuits such as c5378, c7552, s38417, and s38584 compared to other solvers with and without test compaction. This demonstrates the effectiveness of our proposed circuit-adaptive heuristic variable branching technique. However, for the ITC 99 circuits, PA-MiniSat incurs a higher time overhead. This is due to the presence of numerous redundant faults in these circuits, which substantially increase the time overhead of the full-literal watching technique men-

Bench	With test compaction								Without test compaction							
	PA-MiniSat		Kissat		CaDiCaL		MiniSat		PA-MiniSat		Kissat		CaDiCaL		MiniSat	
	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)
c5315	**122**	**65.48**	219	155.34	231	225.9	184	122.97	428	**0.75**	473	4.33	471	5.48	352	4.27
c7552	**134**	**68.44**	309	232.61	260	265.58	214	161.56	488	**1.66**	594	7.00	482	4.94	349	7.12
s38417	**101**	**59.08**	996	605.54	952	722.4	934	492.44	2904	**3.75**	2960	18.30	2554	18.31	2333	9.56
s38584	**169**	**27.89**	1394	718.33	1428	894.97	1578	623.13	5498	**6.36**	4721	24.75	4459	27.32	4057	17.07
b17	**257**	1668.44	411	713.33	418	969.15	439	941.98	968	166.05	949	96.76	939	145.46	858	62.1
b19	**320**	5893	459	2799.06	446	3346.75	438	3100.27	594	331.69	613	206.7	586	199.29	584	216.08
b20	**561**	2525.68	623	959.41	609	1385.00	592	1254.00	1268	488.01	1376	131.42	1339	149.36	1127	203.98
b21	**638**	2468.27	711	996.66	693	1370.31	660	1141.08	1440	315.14	1552	115.89	1503	139.56	1283	176.68
b22	**644**	2357.98	872	1232.29	850	1673.05	836	1587.07	1866	467.42	1938	164.50	1930	192.99	1631	243.90
e203	**333**	3729.7	436	2015.07	452	2663.89	483	2414.97	990	215.75	862	58.48	823	50.87	852	73.35
APRR (%)			41.46		40.49		36.80									

#Pat. is pattern count and APRR is average pattern reduction rate comparison between PA-MiniSat and three other solvers, consistent with the followings.

TABLE I: Comparison of pattern numbers and ATPG runtime across different hard-to-detect fault sets in circuits under different SAT solvers, with and without test compaction techniques.

Bench	PA-MiniSat		Kissat		CaDiCaL		MiniSat		Structural [26]			Commercial tool C		
	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	#Pat.	Time (s)	TC (%)	#Pat.	Time (s)	TC (%)	#Pat.	Time (s)
c5315	**462**	**23.13**	696	196.47	680	326.63	575	183.96	99.73	348	8.812	100	177	0.24
c7552	**605**	**24.14**	722	200.56	727	315.72	683	186.07	99.38	584	10.41	100	283	0.77
s38417	**1075**	**45.27**	1294	377.09	1216	555.24	1216	370.81	99.03	962	13.93	100	964	1.22
s38584	**873**	**17.15**	989	192.17	967	252.55	937	163.13	99.94	800	18.46	100	781	1.11
b17	**5321**	6184.4	5822	4699.8	5969	6060.9	5672	3870.7	96.57	3698	765.22	99.97	2804	2884.56
b19	**11570**	19793	12272	12446	12672	18221	12425	13468	97.668	10747	3210.86	99.95	9615	4876.55
b20	**2985**	**1071.79**	3196	6481.95	3074	7336.11	3029	6979.33	92.24	1383	573.03	99.02	1782	5848.13
b21	3317	**1227.44**	3495	7833.61	3521	9618.13	3208	7917.29	91.33	1409	644.23	99.73	2037	3524.37
b22	**2995**	**1867.76**	3520	7601.55	3459	8571.16	3361	7142.42	91.55	1426	579.87	99.30	2273	6201.65
e203	**2371**	7956.32	2666	3364.76	2813	5430.61	2750	4776.93	98.751	2017	889.82	99.97	2182	2989.89
APRR (%)			13.05		12.75		8.53							

TC represents test coverage, all test coverages of SAT solvers are 100%.

TABLE II: Comparison of test metrics across different circuits' full-fault-sets under different SAT solvers and the original structural method [26] with test compaction techniques, and commercial ATPG at limited parameters set.

tioned in Section III-B. In the best case, PA-MiniSat achieves a speedup of up to 82.53× compared to CaDiCaL for the hard-to-detect fault set in circuit s38584, and the number of patterns is reduced to 10.71% of MiniSat's result. This is because the generated X bits within the test cubes provide ample space for test compaction, significantly reducing the probability of test generation failures in dynamic test compaction due to fault incompatibility, thereby saving the wasted time.

Evaluation of Framework Performance.

TABLE II presents the overall performance of the PastATPG framework using different SAT solver engines, including the number of patterns, time overhead and test coverage. We also compared our approach with the original structural method [26] and the commercial tool C. The test compaction parameter settings are kept the same. We maximized the test compaction capability and ATPG abort limit of C to achieve the highest possible test coverage and the fewest patterns. The experimental results show that, in terms of the number of patterns, the framework incorporating PA-MiniSat consistently generates the fewest patterns compared to the other three solvers, with the exception of the b21, where it falls short of the original MiniSat. Although the test cubes generated by MiniSat contain fewer X bits, they demonstrate better fault compatibility in b21, as evidenced by the uncompacted results in TABLE I. Regarding runtime, the following trend is observed: PastATPG with PA-MiniSat demonstrates an advantage in small to medium-sized circuits and can even outperform commercial tools for medium-sized circuits (b20, b21, b22). However, for large circuits (b17, b19, e203), the overhead of full-literal watching results in a performance disadvantage.

V. Conclusion

This paper presents PastATPG, a hybrid ATPG framework that integrates structurally generated test cubes with those from a SAT solver into a test compaction process. Using a custom SAT solver, PA-MiniSat, which generates test cubes with more X bits for better compaction and speed, PastATPG outperforms other SAT solvers in most cases and commercial ATPG tools in some cases in comparative experiments.

Acknowledgement

This paper is supported in part by National Natural Science Foundation of China (NSFC) under grant No. 92473203, 92373206, and in part by the Chinese Academy of Sciences under grant No. XDB0660100.

REFERENCES

[1] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM journal of Research and Development*, vol. 10, no. 4, pp. 278–291, 1966.

[2] Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE transactions on Computers*, vol. 100, no. 3, pp. 215–222, 1981.

[3] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[4] K. Yang, K.-T. Cheng, and L.-C. Wang, "TranGen: a SAT-based ATPG for path-oriented transition faults," in *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, 2004, pp. 92–97.

[5] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloffel, "PASSAT: Efficient SAT-based test pattern generation for industrial circuits," in *IEEE computer society annual symposium on VLSI: new frontiers in VLSI Design (ISVLSI'05)*. IEEE, 2005, pp. 212–217.

[6] D. Tille and R. Drechsler, "A fast untestability proof for SAT-based ATPG," in *2009 12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*. IEEE, 2009, pp. 38–43.

[7] D. Tille, S. Eggersgluss, and R. Drechsler, "Incremental solving techniques for SAT-based ATPG," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1125–1130, 2010.

[8] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy, and B. Becker, "Thread-parallel integrated test pattern generator utilizing satisfiability analysis," *International Journal of Parallel Programming*, vol. 38, pp. 185–202, 2010.

[9] H. Chen and J. Marques-Silva, "A Two-Variable Model for SAT-Based ATPG," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1943–1956, 2013.

[10] J. Huang, H.-L. Zhen, N. Wang, M. Yuan, H. Mao, Y. Huang, and J. Tao, "Accelerate SAT-based ATPG via Preprocessing and New Conflict Management Heuristics," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 365–370.

[11] A. Czutro, I. Polian, P. Engelke, S. M. Reddy, and B. Becker, "Dynamic Compaction in SAT-Based ATPG," in *2009 Asian Test Symposium*, 2009, pp. 187–190.

[12] S. Jha, K. Chandrasekar, W. Wu, R. Sharma, S. Sengupta, and S. M. Reddy, "A Cube-Aware Compaction Method for Scan ATPG," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 2014, pp. 98–103.

[13] A. Czutro, S. M. Reddy, I. Polian, and B. Becker, "SAT-Based Test Pattern Generation with Improved Dynamic Compaction," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 2014, pp. 56–61.

[14] N. Eén, A. Mishchenko, and N. Amla, "A single-instance incremental SAT formulation of proof-and counterexample-based abstraction," in *Formal Methods in Computer Aided Design*. IEEE, 2010, pp. 181–188.

[15] S. Cai and X. Zhang, "Deep cooperation of CDCL and local search for SAT," in *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*. Springer, 2021, pp. 64–81.

[16] Z. Chao, X. Zhang, J. Huang, J. Ye, S. Cai, H. Li, and X. Li, "A fast test compaction method for commercial dft flow using dedicated pure-maxsat solver," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 503–508.

[17] Z. Chao, S. Wang, P. Tian, S. Yuan, H. Li, J. Ye, and X. Li, "A distributed atpg system combining test compaction based on pure maxsat," in *2023 IEEE 32nd Asian Test Symposium (ATS)*, 2023, pp. 1–6.

[18] S. Eggersgluss and R. Drechsler, "Improving test pattern compactness in SAT-based ATPG," in *16th Asian Test Symposium (ATS 2007)*. IEEE, 2007, pp. 445–452.

[19] S. Eggersglüß, S. Milewski, J. Rajski, and J. Tyszer, "A New Static Compaction of Deterministic Test Sets," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 411–420, 2023.

[20] S. Eggersglüß, R. Krenz-Bååth, A. Glowatz, F. Hapke, and R. Drechsler, "A new SAT-based ATPG for generating highly compacted test sets," in *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2012, pp. 230–235.

[21] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More constraints, better compaction," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 85–90.

[22] S. Eggersglüß, K. Schmitz, R. Krenz-Bååth, and R. Drechsler, "On Optimization-Based ATPG and Its Application for Highly Compacted Test Sets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 2104–2117, 2016.

[23] P. Wang, A. M. Gharehbaghi, and M. Fujita, "An Automatic Test Pattern Generation Method for Multiple Stuck-At Faults by Incrementally Extending the Test Patterns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2990–2999, 2020.

[24] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.

[25] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001, pp. 530–535.

[26] "OpenDACS: CASTest oCASatpg v1.0." [Online]. Available: https://gitlink.org.cn/opendacs/oCASatpg

[27] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," in *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Gurfinkel and V. Ganesh, Eds., vol. 14681. Springer, 2024, pp. 133–152.

[28] "Hummingbirdv2 E203: The Ultra-Low Power RISC-V Core." [Online]. Available: https://github.com/riscv-mcu/e203_hbirdv2

SAPO: Improving the Scalability and Accuracy of Quantum Linear Solver for Portfolio Optimization

Tianze Zhu
Zhejiang University
Hangzhou, China
tianzezhu@zju.edu.cn

Liqiang Lu*
Zhejiang University
Hangzhou, China
liqianglu@zju.edu.cn

Jiajun Chen
Zhejiang University
Hangzhou, China
cuiox@zju.edu.cn

Yuhang Chen
Peking University
Beijing, China
chenyuhang@pku.edu.cn

Hengrui Chen
Zhejiang University
Hangzhou, China
hengruichen@zju.edu.cn

Meng Xi
Zhejiang University
Hangzhou, China
ximeng@zju.edu.cn

Jinshan Zhang
Zhejiang University
Hangzhou, China
zhangjinshan@zju.edu.cn

Xiaoming Sun
Chinese Academy of Sciences
Beijing, China
sunxiaoming@ict.ac.cn

Jianwei Yin
Zhejiang University
Hangzhou, China
zjuyjw@cs.zju.edu.cn

***Abstract*—Portfolio optimization is one of the most important financial problem, suffering from huge computational pressure due to arithmetic complexity. Quantum computing offers polynomial or even exponential speedup that turns out to be a promising approach. However,existing quantum methods is fundamentally limited by either poor scalability or insufficient accuracy. In this paper, we propose SAPO, which formally articulates the quantum circuit that seamlessly integrates financial theory and historical data characteristics with quantum algebra. The circuit design is extended from the HHL algorithm incorporating mean-variance theory, which promotes scalability by equivalent transformation. Then, we present a min-max eigenvalue model that leverages historical financial information to refine parameter settings with high accuracy. Experiments conducted on market data demonstrate that SAPO can effectively reduce the complexity by 36.94% compared to basic HHL [1], [2] and improve the accuracy by 1.46× compared to hybrid HHL [3].**

I. Introduction

Portfolio optimization identifies the best asset allocations to balance the risk and return, which is grappling with computational bottlenecks [4]. Through qubit entanglment and superposition, quantum computing offers an alternative for portfolio optimization. Extending from the mean-variance portfolio theory [5], classical solutions introduce various models, including the standard model, Tobin-Sharpe-Lintner model, and Black's model [6]. It has been demonstrated in recent researches that quantum algorithms are particularly well-suited for optimizing portfolios subject to the simple equality constraints in Black's model [7], [8].

Quantum approaches can be categorized into the quantum Ising solver [9], [10], [11] and the Harrow-Hassidim-Lloyd (HHL) algorithm [1], [2], [3]. The quantum Ising solver handles a quadratic unconstrained binary optimization problem, where asset holdings are represented by superpositions of 0 or 1, with the final strategy determined by the qubit state. On the other hand, the HHL algorithm formulates Black's model as a linear system, computing allocations by projecting the vector space onto quantum states. Despite the significant parallelism of quantum computing, existing attempts listed in Table I suffer from either poor scalability or low accuracy. The quantum Ising solver only tell the strategy of "buy" or "not buy". In other words, it lacks the capability to quantify the weight for each asset, leading to low accuracy. Conversely, the HHL algorithm can provide concrete weights. However, it exhibits overwhelming computational complexity in terms of circuit scale. For example, the circuit for 2-assets portfolio optimization has a depth over 3.7×10^8 after being compiled by Qiskit [12]. Though there are hybrid HHL algorithms [3] attempt to reduce the circuit complexity by inserting classical operations, they still rely on numerical properties and require massive qubits to obtain precise eigenvalues.

*Corresponding author.

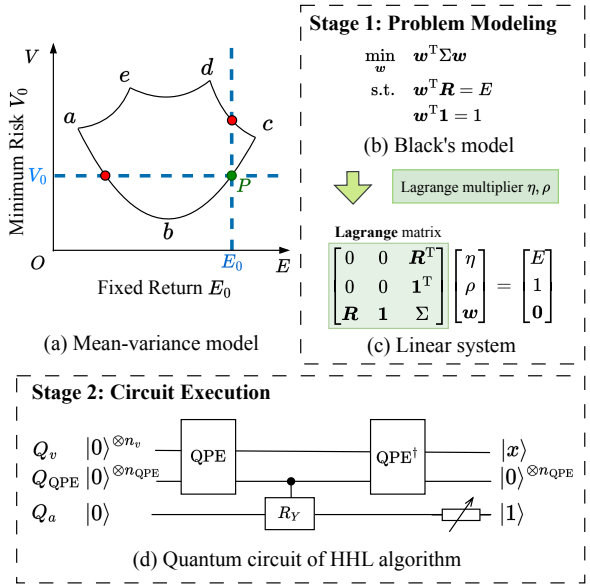


Fig. 1. Portfolio optimization using HHL algorithm.

These limitations originate from the absence of financial theory and historical financial information. When applied to portfolio optimization, the quantum Ising solver is hard to involve the intrinsic features. Consequently, the implementation on a quantum annealer [9] or parameterized quantum circuit [10], [11] is a coarse-grained coding scheme, resulting in overly simplistic recommendations for investment. On the other hand, the drawbacks of existing HHL algorithms are primarily attributed to the complexity of solving the linear system, fundamentally bounded by the large condition number of the matrix in it. Specifically, the condition number serves as an indicator when solving linear equations [13], which is a squared term in the asymptotic complexity of the HHL algorithm [1]. For 2-asset portfolio optimization, the condition number can reach an order of 10^5. Even with the hybrid method [3], which reduces the number of controlled rotation gates by inserting measurements in the midst of the HHL algorithm, it fails to mitigate the substantial complexity of the HHL circuit. Moreover, its accuracy hinges on the matrix's eigenvalue characteristics, making it difficult to estimate the necessary qubit number for the quantum phase estimation (QPE) module.

To overcome these challenges, we propose SAPO that exploits financial theory and data to improve both scalability and accuracy for portfolio optimization. Building upon the HHL algorithm, we observe that previous approaches are significantly limited by the large condition number, leading to poor scalability. To tackle this, we propose a novel constraint scaling technique with equivalent transformation to

TABLE I
EXISTING QUANTUM APPROACHES FOR PORTFOLIO OPTIMIZATION.

Methodology	**quantum Ising solver**			**HHL algorithm**		
	HyQSAT [9]	FrozenQubits [11]	CAFQA [10]	basic HHL [1], [2]	hybrid HHL [3]	**SAPO**
Parameter setting	✘	✘	Clliford simulation	✘	✘	eigenvalue predictor
Resource saving	subset identification	freeze hotspot nodes	✘	✘	middle measurement	condition number reduction
Financial theory	✘	✘	✘	✘	✘	Black's model & historical info.
Complexity level	low	low	low	high	medium	medium
Accuracy level	only tell "buy" or "not buy"			medium	medium	high

shrink it while adhering to the principles of Black's model. Then we develop an iterative algorithm to identify the optimal scaling factors for a given financial dataset. Moreover, with the economic understanding, we present a meticulous circuit parameterization to further improve the accuracy. This is grounded in prior knowledge derived from historical financial data, making it a static model that only depends on the dataset.

The contribution of this work is summarized as follows:

- This paper represents an exploratory endeavor aimed at an in-depth understanding of applying quantum computing to financial scenarios. The key is to profoundly leverage the intricacies of financial knowledge and data. We investigate the optimization opportunities of Black's model at both arithmetic level and quantum circuit level.
- We propose the constraint scaling technique for significant reduction of the condition number, which exploits the equality constraints in Black's model, leading to low complexity.
- We propose a prediction model, which estimates the minimum and maximum eigenvalues to delicately parameterize the quantum circuit, resulting in high accuracy.

In our experiments, SAPO achieves around 34.64%-36.94% complexity reduction compared to basic HHL [1], [2] with different settings of the absolute error. Furthermore, compared to Qiskit HHL [12] and hybrid HHL [3], SAPO improves the accuracy by $1.52\times$ and $1.46\times$ under fixed qubit number, and by $1.28\times$ and $1.39\times$ under fixed asset number, respectively.

II. BACKGROUND

A. Portfolio Optimization

H. Markowitz details the *mean-variance theory* for portfolio optimization [5], which aims to find the optimal asset allocation vector $\boldsymbol{w}$, with the objective of minimizing the risk while maximizing the expected return. First, we consider the returns of n assets across each of the T time periods, denoted as:

$$\boldsymbol{r}_t = \left[r_1^t, r_2^t, \cdots, r_n^t\right]^{\mathrm{T}}, \quad t = 1, 2, \cdots, n \tag{1}$$

Subsequently, the historical expected return vector, denoted as $\boldsymbol{R}$, and the covariance matrix, denoted as Σ, can be estimated using the data from the past T time slices as follows:

$$\begin{aligned} \boldsymbol{R} &\equiv \mathbb{E}(\boldsymbol{r}_t) \approx \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{r}_t \\ \Sigma &\equiv \mathbb{V}(\boldsymbol{r}_t) \approx \frac{1}{T-1}\sum_{t=1}^{T} (\boldsymbol{r}_t - \boldsymbol{R})^{\mathrm{T}} (\boldsymbol{r}_t - \boldsymbol{R}) \end{aligned} \tag{2}$$

Under a given $\boldsymbol{w}$, the expected return E and risk V can be defined:

$$\begin{aligned} E &\equiv \boldsymbol{w}^{\mathrm{T}} \boldsymbol{R} \\ V &\equiv \boldsymbol{w}^{\mathrm{T}} \Sigma \boldsymbol{w} \end{aligned} \tag{3}$$

For instance, as depicted in Fig. 1(a), the optimal $\boldsymbol{w}$ corresponds to the minimized risk V_0 for a specified expected return E_0. Typically, the problem can be formulated as a quadratic programming problem, where $\boldsymbol{w}$ constitutes the solution (Fig. 1(b)). In particular, Black's model [6] presents the problem with equality constraints. By employing the Lagrange multiplier method, the optimization can be converted into a linear system with a Lagrange matrix M_L (Fig. 1(c)).

B. HHL Algorithm

Considering A as a Hamiltonian matrix with eigenvalues $\{\lambda_j\}_{j=1}^n$ and eigenvector $\{|\mu\rangle_j\}_{j=1}^n$ and $\boldsymbol{b}$ as a normalized vector represented by $\sum_{j=1}^n \beta_j |\mu_j\rangle$. The solution to the linear system $A\boldsymbol{x} = \boldsymbol{b}$ can be represented as a quantum state:

$$|x\rangle = \sum_{j=1}^{n} \frac{\beta_j}{\lambda_j} |\mu_j\rangle \tag{4}$$

Fig. 1(d) illustrates the quantum circuit of the HHL algorithm [1]. The circuit is comprised of three primary components, including QPE, uniformly controlled R_Y gate, and QPE†. There are also three quantum registers for different functions: a) Q_v is used to load the vector $\boldsymbol{b}$ as part of the QPE module; b) Q_{QPE} stores the output of QPE, recording eigenvalue information; c) Q_a only contains an auxiliary qubit controlled by Q_{QPE} through the uniformly controlled R_Y gate.

- **QPE module.** In the HHL algorithm, the QPE part is used to extract eigenvalues of the input Hamiltonian matrix A. Specifically, QPE generates a superposition state of phases $\varphi \in [0, 1)$ and records this in the quantum register Q_{QPE}, establishing an entanglement with Q_v. Each phase and eigenvalue adhere to the relationship:
$$e^{2\pi i \varphi_j} = e^{i\lambda_j H_t} \tag{5}$$
where H_t is a key hyperparameter named *Hamiltonian simulation time*. Increasing the number of qubits for the QPE module can improve the preciseness of the phase, making the eigenvalue obtained by Eq. (5) more accurate. Thus, the number of qubits in Q_{QPE},represented as n_{QPE}, is an important hyperparameter. In addition, QPE† is the inverse operation of QPE, which is applied at the end of the quantum circuit to disentangle Q_v and Q_{QPE}.
- **Uniformly Controlled R_Y Gate.** The uniformly controlled R_Y gate is a multi-qubit controlled gate. In the HHL algorithm, according to the phase φ_j stored in Q_{QPE}, $R_Y(2\arcsin(Const/\lambda_j))$ gate is applied to the single qubit stored in Q_a so that the eigenvalue can be placed on the denominator of the amplitude of the quantum state. Here, $Const$ is a *normalization constant*, another critical hyperparameter in the HHL algorithm.

The result is obtained by measuring the auxiliary qubit stored in Q_a at the end of the quantum circuit. Ideally, if the result is 1, the quantum state collapses to the normalization of Eq. (4). For the sake of simplicity, this paper assumes that we can directly obtain

the quantum state vector after executing the circuit, which is post-processed to get the desired solution, denoted as $\tilde{\boldsymbol{w}}$.

III. SAPO Overview

A. Evaluation Metrics

As aforementioned, our goal is to improve the scalability and accuracy in portfolio optimization. To achieve this, we define two metrics, named *complexity* and *accuracy*, to evaluate quantum approaches.

First, we model the complexity in both spatial and temporal dimensions by counting the consumed quantum resources and the circuit depth. According to the HHL algorithm, the total number of qubits N^q is the summation of three quantum registers.

$$N^q = n_v + n_{\text{QPE}} + n_a \tag{6}$$

Besides, we use N^{CNOT} and N^d to represent the number of CNOT gates and the circuit depth after compilation. This is because the CNOT gate is the dominant source of errors on most existing NISQ devices [14], and the circuit depth is also an important indicator to evaluate the complexity of quantum algorithm [15]. Putting it all together, we formulate the complexity metric as follows.

$$\Theta = N^q + \lg\left(N^{\text{CNOT}}\right) + \lg\left(N^d\right) \tag{7}$$

Here, for simplicity, we employ lg function on N^{CNOT} and N^{d} since these two terms exhibit exponential growth with the number of qubits.

The accuracy is evaluated by quantifying the discrepancy between the calculated allocation $\tilde{\boldsymbol{w}}$ and the optimal choice $\boldsymbol{w}$. Thus, we formulate the accuracy in terms of cosine similarity:

$$\Delta = \frac{1}{2}\left(1 + \frac{\boldsymbol{w}^{\text{T}}\tilde{\boldsymbol{w}}}{\|\boldsymbol{w}\|\|\tilde{\boldsymbol{w}}\|}\right) \tag{8}$$

where Δ always falls within the interval $[0, 1]$. The closer it is to 1, the more accurate it is.

B. SAPO Advantages

The novelty of SAPO is that it leverages the knowledge of financial theory and data to optimize the design of the quantum approach. Extending from the Black's model, we observe that the condition number of the Lagrange matrix derived from real market data is typically large. This could markedly increase the circuit complexity, thereby limiting the scalability with respect to problem size. To address this, we propose a constraint scaling technique that adopts an equivalent transformation.

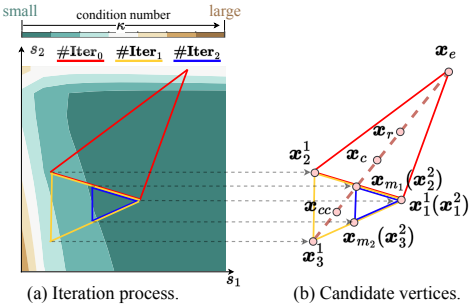
Next, we propose min-max eigenvalue model to enhance the accuracy of portfolio optimization. Our approach relies on the insight that knowing only the minimum and maximum eigenvalues of the Lagrange matrix is sufficient for setting the hyperparameters of the HHL circuit. We provide a quantitative formulation linking circuit hyperparameters to these eigenvalues obtained from a static predictor trained on historical data. In a word, we demonstrate that SAPO exhibits a lower end-to-end complexity compared to previous methods while achieving much higher accuracy for portfolio optimization.

IV. Equivalent Scaling of Lagrange Matrix

A. Equivalent Transformation

For the Lagrange matrix M_L featuring a set of eigenvalues $\{\lambda_j\}$, we first define min/max eigenvalue $|\lambda|_{\min} = \min\{|\lambda||\lambda \in \{\lambda_j\}\}$ and $|\lambda|_{\max} = \max\{|\lambda||\lambda \in \{\lambda_j\}\}$. We start by optimizing the condition number $\kappa = |\lambda|_{\max}/|\lambda|_{\min}$ of the Lagrange matrix.

Mathematically, the condition number indicates the difficulty of matrix calculus using approximation algorithms [13]. For example, when the condition number is exactly one, it means that the matrix is a scalar multiple of a linear isometry. In contrast, a larger condition number suggests that the linear system is hard to solve using an approximation solution, e.g., Jacobi method [16] and conjugate gradient method [17]. According to the HHL algorithm [1], the upper-bound complexity is $O(\log n \cdot \kappa^2/\epsilon)$, where n is the matrix size and ϵ is the given upper bound of absolute error. In the portfolio optimization problem, the condition number is usually in the scale of 10^5, which implies a huge complexity.



(a) Iteration process. (b) Candidate vertices.

Fig. 2. iterative method for searching the optimal scaling factor.

Conventional method to deal with large condition number involves rescaling the matrix by rows or columns [18]. However, such rescaling fails to keep the equivalence, reducing the expressiveness of the linear system. Thus, this method has to introduce various parameters that need to be tuned to compensate for the gap. Instead of directly scaling the matrix, we choose to modify Black's model by multiplying two scaling factors s_1, s_2 with the equations in Fig. 1(b).

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \boldsymbol{w}^{\text{T}}\Sigma\boldsymbol{w} \\ \text{s.t.} \quad & s_1\boldsymbol{w}^{\text{T}}\boldsymbol{R} = s_1 E \\ & s_2\boldsymbol{w}^{\text{T}}\boldsymbol{1} = s_2 \end{aligned} \tag{9}$$

In this manner, we have an equivalent Lagrange matrix as follows, which is an equivalent transformation with only two scaling factors.

$$\begin{bmatrix} 0 & 0 & s_1\boldsymbol{R}^{\text{T}} \\ 0 & 0 & s_2\boldsymbol{1}^{\text{T}} \\ s_1\boldsymbol{R} & s_2\boldsymbol{1} & \Sigma \end{bmatrix} \begin{bmatrix} \eta \\ \rho \\ \boldsymbol{w} \end{bmatrix} = \begin{bmatrix} s_1 E \\ s_2 \\ \boldsymbol{0} \end{bmatrix} \tag{10}$$

B. Search Algorithm of Scaling Factor

The scaling factor helps to adjust the condition number, providing the opportunity to reduce the difficulty of portfolio optimization. Empirically, we test that outstanding scaling factors can significantly reduce the condition number from 10^5 to ˜10. However, the optimal scaling factor varies in different Lagrange matrices. Therefore, we try to find a static scaling factor on a target dataset, which achieves optimal reduction on average in this dataset.

We adopt a iterative algorithm inspired by Nelder-Mead simplex algorithm [19] to search the optimal scaling factor. Here, we use a two-dimensional vector $\boldsymbol{x}$ to notate the tuple of (s_1, s_2) as shown in Fig. 2(a). First, given a set of Lagrange matrices derived from a specific dataset, we define a loss function that calculates the average of condition numbers after applying the scaling factor. The algorithm begins with the initialization of three scaling factors, labeled as $\boldsymbol{x}_1^0$, $\boldsymbol{x}_2^0$, and $\boldsymbol{x}_3^0$, drawing a triangle region in $s_1 - s_2$ plane. For each iteration, we generate multiple candidates according to the vertex of the current triangle. In Fig. 2(b), each candidate is the midpoint of its line segment (x_m , x_m , x_{cc}, etc.). After determining the candidates, we keep one vertex from the current three vertices for the next iteration, which shows the minimum loss for the given dataset in current iteration. The other two optimal vertices are selected from the remaining vertices and candidates. After several iterations, we choose

the best vertex as the scaling factor. As a result, the average condition number gradually decreases during iterations as Fig. 2 shown. Note that, to ensure quantum speedup, the scaling factor is pre-obtained by running the algorithm on the target dataset.

V. Min-Max Eigenvalue Model

A. Parameterization Using Min-Max Eigenvalue

As mentioned in Section II-B, the QPE module, taking up most of the qubits, is used to extract eigenvalues of the input matrix. Furthermore, since eigenvalues play an important role in characterizing the linear system, their preciseness significantly affects the final output accuracy of the HHL circuit. Thus, the prior knowledge of eigenvalues can naturally improve the output accuracy. However, profiling all eigenvalues requires a huge computational cost that destroys the quantum advantage. Considering that the eigenvalues are inherently related to the hidden information of the portfolio optimization problem, we predict $|\lambda|_{\min}$ and $|\lambda|_{\max}$ and leverage them to set hyperparameters. We choose these two eigenvalues rather than the largest two for two reasons:

- In the view of economics, $|\lambda|_{\min}$ captures the microeconomic details, which describes macroeconomic patterns of asset changes. While $|\lambda|_{\max}$ governs the long-term behavior of the system.
- The condition number κ, serving as the complexity metric, is calculated by $|\lambda|_{\max}/|\lambda|_{\min}$. Thus, the knowledge of these two eigenvalues helps to guide the qubit need of QPE, and then provides the opportunity to quantitatively evaluate the overall circuit.

The hyperparameter settings require that all eigenvalues of the input matrix are within $(-1, 1)$. To ensure this, we perform a standardization operation on the Lagrange matrix as a preprocessing.

$$M_L^{\text{std}} \equiv \frac{l}{|\lambda|_{\max}} M_L, \quad l \in (0, 1) \tag{11}$$

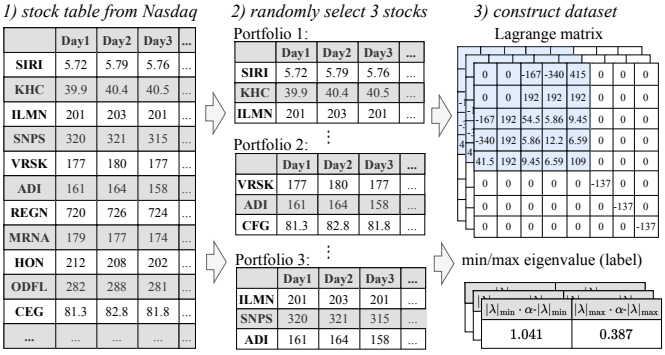
In this paper, we set $l = 0.5$ for simplicity, which makes eigenvalues of M_L^{std} lie in $[1/2\kappa, 1/2)$, but the value of κ does not change.

- **Hamiltonian Simulation Time: $H_t \equiv \pi$.** When $H_t = \pi$, the relationship between the output phase φ of QPE and the eigenvalue λ of M_L^{std} can be represented by a piece-wise function as follows

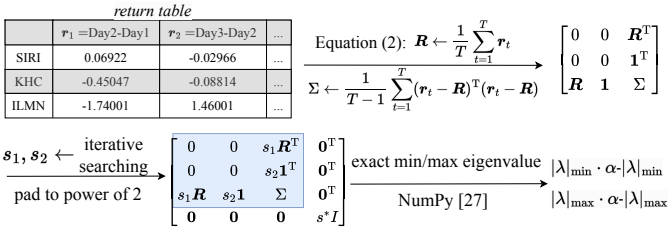
$$\lambda = \begin{cases} 2\varphi, & \varphi < \frac{1}{2}, \\ 2(\varphi - 1), & \varphi > \frac{1}{2}. \end{cases} \tag{12}$$

  Other methods usually set $H_t = 2\pi$ [12], [3] or randomly [20], which makes different value φ may map to the same eigenvalue, leading to low accuracy. In detail, through standardization by $|\lambda|_{\max}$, we ensure $|\lambda| < 1$ so that we can split the interval of $\varphi \in [0, 1)$ into two halves and map to the case of either the eigenvalues being positive or negative bounded by $1/2$. In a word, this one-to-one map results in more preciseness of all eigenvalues.
- **Number of Qubits for QPE: $n_{\text{QPE}} \geq 2 + \left\lceil \log_2 \frac{|\lambda|}{|\lambda| \cdot \epsilon} \right\rceil$.** The minimum n_{QPE} ensures that the absolute error of the solution output by HHL algorithm does not exceed ϵ. Essentially, this calculation uses the quotient of $|\lambda|_{\min}$ and $|\lambda|_{\max}$, which is the condition number. This also further illustrates the fact that the larger the condition number, the more difficult to solve portfolio optimization problems, because we need more qubits to perform QPE.
- **Normalization Constant: $Const \equiv |\lambda|_{\min}$.** The state before final measurement is

$$\sum_{j=1}^{n} \beta_j |\mu_j\rangle_{Q_v} \left(\sqrt{1 - \frac{Const^2}{\lambda_j^2}} |0\rangle_{Q_a} + \frac{Const}{\lambda_j} |1\rangle_{Q_a} \right) \tag{13}$$



(a) Constructing the dataset for 3-assets problem using Nasdaq maket data.The values in the Lagrange matrix need to multiply 10^{-5}.



(b) Details of calculating the dataset.

Fig. 3. Workflow of dataset construction.

For auxiliary qubit in Q_a, $Const \leq |\lambda|_{\min}$ must be satisfied. On the other hand, it is expected that the probability of measuring the auxiliary qubit is close to 1 as high as possible, which means the algorithm has a high success rate. Thus, we set $Const \equiv |\lambda|_{\min}$.

B. Min-Max Eigenvalue Predictor

Most conventional approaches deliver inefficient eigenvalue information. For example, [21] only predicts eigenvalues of Σ, and [22] only estimate the eigenvalue counts. On the contrary, [23] aims to figure out all eigenvalues, necessitating massive computational costs. We develop a predictor to quickly predict the min-max eigenvalue by leveraging historical financial information without any prior knowledge. More importantly, this prediction model is dataset-dependent and can be trained offline, which greatly reduces the computational pressure of the classical part.

We apply support vector regression (SVR) as our basic model. Starting from the matrix in Eq. (10), elements of $s_1\boldsymbol{R}$ are concatenated with the upper triangular portion of Σ into a vector and take this vector as the input of the SVR. The output is the predicted $|\lambda|_{\min}$ and $|\lambda|_{\max}$ multiplied by α-$|\lambda|_{\min}$ and α-$|\lambda|_{\max}$, respectively. Here, we introduce the tunable parameter α to mitigate the truncation error of floating-point.

The training data is collected from three representative stock exchanges listed in Table III. Fig. 3(a) illustrates the detailed workflow for constructing the 3-asset dataset from Nasdaq market. First, we collect the data of 3100 stocks within 251 days and randomly select three stocks to formulate a series of portfolio optimization problems. Then, we can derive the Lagrange matrix and label it with the exact min/max eigenvalues. To be specific, we build a return table by calculating the increments on adjacent days as shwon in Fig. 3(b). Using Eq. (2), we can obtain the historical expected return vector and the covariance matrix to construct the Lagrange matrix. Finally, we can compute the min/max eigenvalue using NumPy [24], regarding it as the label for SVR model training. In particular, since the HHL algorithm can only deal with matrices with size of the power of 2, the Lagrange matrix is padded with zeros and identity matrix. For

TABLE II
MODELS FOR PREDICTING MIN/MAX EIGENVALUES.

Model	# Parameter	#Operations	ℓ_2-loss of $\lvert\lambda\rvert_{\min}, \lvert\lambda\rvert_{\max}$
SVR	1640	1.6×10^3	0.033, 0.39
Decision Tree	1969	2×10^2	0.066, 1.66
MLP	2137	2×10^3	0.034, 0.024
LeNet	20411	2×10^4	0.088, 15.8

example, the 3-asset problem leads to a 5×5 Lagrange matrix, which is then padded by putting a scaled identity matrix at right-bottom.

We have also tried other methods like decision tree and deep learning. Table II evaluates four models for eigenvalue prediction. The other three show limited improvement yet are accompanied by a disproportionately high computational complexity. The effectiveness of SVR lies in its generalization performance, which is demonstrated by portfolio optimization based on the stock pre-selection [25]. Since the computational complexity of SVR is independent of the input space dimensionality [26], our method avoids the computational bottleneck caused by large-scale training to some extent and preserves the quantum advantages.

VI. EVALUATION

A. Experiment Setup

Implementation. SAPO is constructed based on Qiskit [12]. The QPE module and uniformly controlled R_Y gate is implemented by *PhaseEstimation* and *UCRYGate* in the framework. The optimization level of transpiler is set to 0. Scaling factor searching is extended from Nelder-Mead module in SciPy library[27]. The min-max eigenvalue predictor is implemented using Scikit-Learn [28]. Table III gives the configurations of SAPO on each dataset.

Comparison. We compare SAPO with three HHL implementations: basic HHL [1], [2], Qiskit HHL [12] and hybrid HHL [3]. Rebentrost et al. [2] images applying basic HHL [1] to the Black's model, but without any specific parameterizations and experiments. Thus, when compared to basic HHL, we assume eigenvalues have been pre-calculated, only comparing the complexity metric. In Qiskit HHL [12], the Hamiltonian simulation time is 2π, and the uniformly controlled R_Y gate is implemented using *ExactReciprocal*. Hybrid HHL [3] involves both classical and quantum computing. Both Qiskit HHL and hybrid HHL require specifying the number of qubits for QPE, which cannot adaptively balance the trade-off between complexity and accuracy. Thus, we only compare the accuracy metric with them, under the same number of qubits for QPE. Due to the huge complexity of simulating the HHL circuit, we consider the portfolio optimization problem ranging from 2 assets to 6 assets.

B. Scalability and Accuracy Improvement

Complexity analysis. Table IV compares the circuit complexity in terms of the number of necessary qubits, circuit depth, the number of CNOT gates, and the total complexity defined by Eq. 7. Overall, SAPO achieves around 34.64%-36.94% complexity reduction with the absolute error ϵ ranging from $1/16$ to $1/4$. In the case of $\epsilon = 1/8$, the detailed resource consumption is listed. SAPO shows a remarkable reduction, especially N^{CNOT} (achieving a maximum reduction of 99.996% for 2-asset portfolio optimization). According to the value of n^{QPE}, such resource-saving results from our scaling technique that shrinks the condition number. The complexity reduction lightly decreases for smaller ϵ, which comes from the logarithmic function when calculating complexity in Eq. (7).

We find that the complexity is significantly higher when handling 3-asset portfolio optimization problems. This may arise from the infinitesimally small or large condition number in several Lagrange matrices. To confirm this, we also give the median value of each metric in the footnote of Table IV, which turns out to be a much more reasonable value, following the trend of the other cases.

TABLE III
CONFIGURATIONS OF SAPO ON EACH DATASET.

Dataset	#Stocks	Scaling factor search			Min-max predictor	
		s_1	s_2	#iter	α-$\lvert\lambda\rvert_{\min}$	α-$\lvert\lambda\rvert_{\max}$
Nasdaq	3100	0.56	1.9e-3	119	3000	100
NYSE	400	0.25	3.2e-4	152	5000	800
AMEX	40	0.28	6.6e-4	125	4000	400

Accuracy analysis. Fig. 4 gives the accuracy improvement compared to hybrid HHL [3] and Qiskit HHL [12], where the accuracy Δ is defined by Eq. (8). Fig. 4(a) sets a fixed number of qubits, which shows that SAPO improves the accuracy by $1.52\times$ on average compared to Qiskit HHL, and $1.46\times$ on average compared to hybrid HHL. For limited available qubits, SAPO reduces the difficulty of solving linear systems and identifies more reasonable parameter settings for the HHL algorithm, thus, being able to solve more complex problems. In the case of the 5-asset portfolio optimization problem, Fig. 4(b) shows that SAPO improves the accuracy by $1.28\times$ on average compared to Qiskit, and $1.39\times$ compared to hybrid HHL. Particularly, the accuracy of SAPO, with a 4-qubit QPE circuit ($\Delta = 0.96$), outperforms all other approaches, even if they involve a 10-qubit QPE circuit. In a word, SAPO makes a better trade-off between accuracy and scalability by leveraging the features of Black's model and historical financial information.

We conduct an ablation study to quantitatively evaluate the advantages of scaling factor and eigenvalue prediction. Here, SAPO without scaling factor means we train the eigenvalue predictor using original Lagrange matrices. For SAPO without eigenvalue predictor, we adopt the same scheme as Qiskit HHL. Fig. 4(a) demonstrates that the scaling factor contributes most, improving the accuracy $1.74\times$-$1.96\times$ under the constraint of 5-qubits QPE module, and improves the accuracy $1.10\times$-$1.96\times$ for 5-asset portfolio optimization. This mainly benefits from our search algorithm. The accuracy of SAPO without the scaling factor is a little bit lower than hybrid HHL and Qiskit HHL. This is because a huge condition number makes it difficult to predict eigenvalues, leading to a less effective or even counterproductive eigenvalue predictor. However, the other two do not make any use of predicted eigenvalues. The gain from the eigenvalue predictor is $1.21\times$-$1.54\times$ and $1.05\times$-$1.37\times$ for fixed qubit number and fixed asset number, respectively. By leveraging prior knowledge, SAPO is able to capture the min-max eigenvalue more precisely, therefore, achieving high accuracy. Fig. 4(b) also suggests that the technique of scaling factor exhibits more improvement for small QPE modules.

C. Evaluation of the Scaling Factor

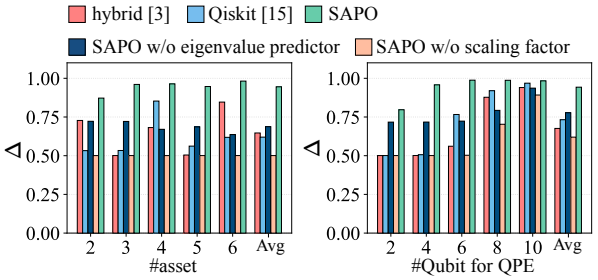
This section provides a detailed evaluation of the scaling factor that serves to squeeze the condition number. Fig. 5(a) draws the distribution of the optimal scaling factor under different datasets, where we randomly select a series of portfolio optimization problems. We can observe that most of their optimal scaling factors lie in a small region, demonstrating that the scaling factor can be static. The figure outlines the region and its edge point of AMEX dataset as an example. SAPO searches the point via the iterative algorithm, which is close to the central in this region.

We also compare the resultant condition number using the scaling factor from the edge point and the optimal scaling factor, as illustrated

TABLE IV
COMPLEXITY COMPARISON WITH BASIC HHL [1], [2].

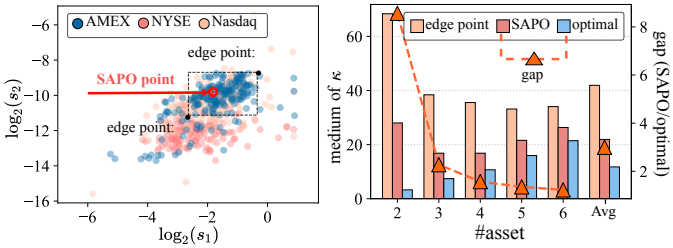
#Asset	The absolute error rate $\epsilon = 1/8$								$\epsilon = 1/16$		$\epsilon = 1/4$	
	N^q		N^d		N^{CNOT}		Θ		Θ		Θ	
	basic HHL	**SAPO**	basic HHL	**SAPO**	basic HHL	**SAPO**	basic HHL	**SAPO**	basic HHL	**SAPO**	basic HHL	**SAPO**
2	19.77	11.80	2.148×10^9	8.016×10^4	1.123×10^9	4.202×10^4	33.82	21.05	35.42	22.65	32.22	19.45
3	21.00	13.87	1.782×10^8	2.523×10^7	9.381×10^7	1.328×10^7	36.76	25.32	38.36	26.93	35.16	23.72
4	21.88	13.23	6.274×10^8	6.454×10^5	3.304×10^8	3.399×10^5	38.17	24.30	39.77	25.91	36.57	22.70
5	22.37	13.21	9.526×10^8	7.062×10^5	5.016×10^8	3.719×10^5	38.94	24.27	40.55	25.87	37.34	22.66
6	22.75	13.42	9.558×10^8	6.737×10^5	5.033×10^8	3.549×10^5	39.56	24.61	41.16	26.22	37.96	23.01
Reduction	39.16%		97.12%		97.12%		36.13%		34.64%		36.94%	

The medium value for #asset=3: $N^q = 13$, $N^d = 4.048 \times 10^5$, $N^{\text{CNOT}} = 2.132 \times 10^5$, $\Theta = 23.94$.



(a) Accuracy under 5-qubit QPE across different number of assets. (b) Accuracy under 5-asset problem across different number of qubits.

Fig. 4. Accuracy comparison with hybrid HHL[3] and Qiskit HHL [12].



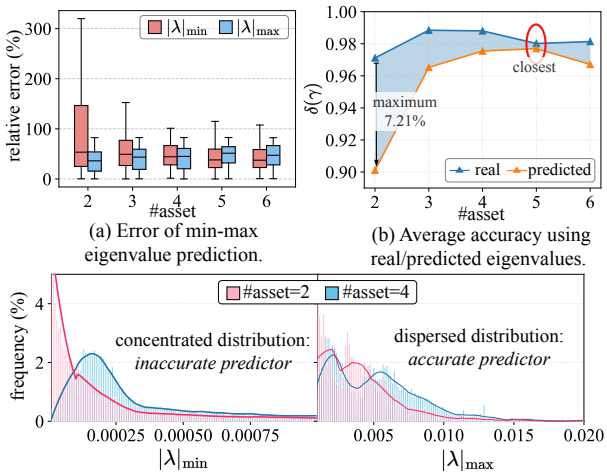
(a) The optimal scaling factor for different portfolios from each dataset. (b) Condition number across different scaling factor.

Fig. 5. Evaluation of the scaling factor.

in Fig. 5(b). Even choosing the edge point still shows a great reduction in the condition number. We also draw the gap between SAPO and the optimal setting, calculated as $\frac{\kappa \text{ of SAPO}}{\kappa \text{ of optimal}}$. The gap is narrowed when more assets are taken into account, from $8.56\times$ (2-asset) to $1.23\times$ (6-asset), demonstrating the validness of static scaling factors. This is because a large number of assets results in more combinations of assets (i.e., more Lagrange matrices), which makes the scaling factor fit the problem better. As a result, the condition number of the Lagrange matrix is dramatically reduced by at least 98.59% through a fixed scaling factor.

D. Evaluation of the Eigenvalue Predictor

In this section, we evaluate the preciseness of the eigenvalue predictor and investigate how it affects the accuracy of the final result. First, we collect the predicted min-max eigenvalues across 1500 cases. Then, we calculate the relative error compared to the real eigenvalues, as shown in the box plot of Fig. 6(a). Overall, the relative error of most predicted values is within 50%. The relative error of some samples is more than 100%, and even exceeds 300% when #asset=2. This is because some eigenvalues are close to zero. Fig. 6(b) depicts the gap of accuracy using the real eigenvalues and the predicted ones, calculated as $\left(1 - \frac{\Delta \text{ using predicted}}{\Delta \text{ using real}}\right) \times 100\%$. The maximum gap is 7.21% under the case of #asset=2, which is still an



(a) Error of min-max eigenvalue prediction. (b) Average accuracy using real/predicted eigenvalues.

(c) Distribution of min/max eigenvalues.

Fig. 6. Evaluation of the min/max eigenvalue predictor.

acceptable accuracy ($\Delta = 0.901$). Especially when #asset=5, the gap almost disappears, resulting from our precise prediction model.

We also picture the distribution of the real $|\lambda|_{\min}$ and $|\lambda|_{\max}$, as shown in Fig. 6(c). We observe that $|\lambda|_{\min}$ is concentrated in the interval $(0, 0.0001]$ when the number of assets is 2, which makes this eigenvalue hard to predict. That's why the relative error of $|\lambda|_{\max}$ is much lower than $|\lambda|_{\min}$ in Fig. 6(a).

VII. CONCLUSION

We propose SAPO to improve the scalability and accuracy for quantum portfolio optimization, indicating the idea that the algorithm design for quantum finance should be based on the deep integration of quantum computing and financial knowledge. SAPO is a novel design of HHL algorithm, which incorporates mean-variance theory and leveraging financial data characteristics. Depending on the equivalent transform, the condition number of the Lagrange matrix is extremely reduced, which improves the scalability. Considering the close connection between eigenvalues and the HHL algorithm, we propose a min-max eigenvalue model to guide parameter settings for highly accurate computation. Our experiments demonstrate that SAPO can achieve lower complexity and higher accuracy than conventional quantum methods.

ACKNOWLEDGEMENT

This work was supported by National Key Research and Development Program of China (No. 2023YFF0905200) and Zhejiang Provincial Natural Science Foundation of China under Grant (No.LR25F020002).

References

[1] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.

[2] P. Rebentrost and S. Lloyd, "Quantum computational finance: quantum algorithm for portfolio optimization," *arXiv preprint arXiv:1811.03975*, 2018.

[3] Y. Lee, J. Joo, and S. Lee, "Hybrid quantum linear equation algorithm and its experimental test on ibm quantum experience," *Scientific reports*, vol. 9, no. 1, p. 4778, 2019.

[4] A. Sedighi and D. Jacobson, "Computational challenges and opportunities in financial services," in *Smart Computing and Communication: 4th International Conference, SmartCom 2019, Birmingham, UK, October 11–13, 2019, Proceedings 4*. Springer, 2019, pp. 310–319.

[5] H. Markowitz, "Portfolio selection*," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[6] H. M. Markowitz and G. P. Todd, *Mean-variance analysis in portfolio choice and capital markets*. John Wiley & Sons, 2000, vol. 66.

[7] D. Qu, E. Matwiejew, K. Wang, J. B. Wang, and P. Xue, "Experimental implementation of quantum-walk-based portfolio optimisation," *Quantum Science and Technology*, 2024.

[8] D. Venturelli and A. Kondratyev, "Reverse quantum annealing approach to portfolio optimization problems," *Quantum Machine Intelligence*, vol. 1, pp. 17 – 30, 2018.

[9] S. Tan, M. Yu, A. Python, Y. Shang, T. Li, L. Lu, and J. Yin, "Hyqsat: A hybrid approach for 3-sat problems by integrating quantum annealer with cdcl," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 731–744.

[10] G. S. Ravi, P. Gokhale, Y. Ding, W. Kirby, K. Smith, J. M. Baker, P. J. Love, H. Hoffmann, K. R. Brown, and F. T. Chong, "Cafqa: A classical simulation bootstrap for variational quantum algorithms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 1*. New York, NY, USA: Association for Computing Machinery, 2022, p. 15–29.

[11] R. Ayanzadeh, N. Alavisamani, P. Das, and M. Qureshi, "Frozenqubits: Boosting fidelity of qaoa by skipping hotspot nodes," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2*, 2023, pp. 311–324.

[12] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross *et al.*, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, 2024.

[13] Wikipedia contributors, "Condition number — Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Condition_number

[14] G. Q. AI, "Quantum computer datasheet," May 2021. [Online]. Available: https://quantumai.google/hardware/datasheet/weber.pdf

[15] IBM, "Quantumcircuit — ibm quantum document," accessed: March 29, 2024. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/circuit

[16] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[17] J. R. Shewchuk *et al.*, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[18] R. D. Braatz and M. Morari, "Minimizing the euclidean condition number," *SIAM Journal on Control and Optimization (SICON)*, vol. 32, no. 6, pp. 1763–1768, 1994.

[19] F. Gao and L. Han, "Implementing the nelder-mead simplex algorithm with adaptive parameters," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 2012.

[20] R. Yalovetzky, P. Minssen, D. Herman, and M. Pistoia, "Nisq-hhl: Portfolio optimization for near-term quantum hardware," 2021. [Online]. Available: https://www.jpmorgan.com/content/dam/jpm/cib/complex/content/technology/publications/white-paper-nisq-hhl-portfolio-optimization-for-near-team-quantum-hardward.pdf

[21] X. Mestre, "Improved estimation of eigenvalues and eigenvectors of covariance matrices using their sample estimates," *IEEE Transactions on Information Theory*, vol. 54, no. 11, pp. 5113–5129, 2008.

[22] E. Di Napoli, E. Polizzi, and Y. Saad, "Efficient estimation of eigenvalue counts in an interval," *Numerical Linear Algebra with Applications*, vol. 23, no. 4, pp. 674–692, 2016.

[23] R. D. Somma, "Quantum eigenvalue estimation via time series analysis," *New Journal of Physics*, vol. 21, no. 12, p. 123025, 2019.

[24] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[25] N. F. Silva, L. P. de Andrade, W. S. da Silva, M. K. de Melo, and A. O. Tonelli, "Portfolio optimization based on the pre-selection of stocks by the support vector machine model," *Finance Research Letters*, vol. 61, p. 105014, 2024.

[26] M. Awad, R. Khanna, M. Awad, and R. Khanna, "Support vector regression," *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*, pp. 67–80, 2015.

[27] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

SSDL-ILT: Efficient ILT utilizing a self-supervised deep learning model

Rui Xu*
Zhangjiang Laboratory
Shanghai 201204, China
xurui@zjlab.ac.cn

Junqi Yang
Zhangjiang Laboratory
Shanghai 201204, China

Haoxiang Jiang
Shanghai Jiao Tong University
Zhangjiang Laboratory
Shanghai 200240, China

Ming Fang
Zhangjiang Laboratory
Shanghai 201204, China

***Abstract*—Inverse lithography technology (ILT) is an advanced resolution enhancement technique that achieves mask optimization at the pixel level. However, application of ILT is hindered by time-intensive physical simulation. Herein, we propose an efficient ILT algorithm leveraging a deep learning model. A novel loss function is constructed to guide the model training in a self-supervised manner, eliminating the requirement of labelled data that might be nontrivial to acquire. The trained model outputs final mask patterns without further ILT optimization. Sub-resolution assist features (SRAFs) are generated automatically, the complexity of which can be adjusted during the training process to control mask manufacturability. The model was trained and validated on ICCAD-2013 CAD contest dataset. Better pattern fidelity and up to 12,000 times speedup are observed compared to other SOTA models. The trained model also shows good generalization ability to geometrically-different design patterns from another dataset, via a few-shot learning approach.**

Keywords—ILT, self-supervised, deep learning, SRAF, few-shot learning

I. INTRODUCTION

Lithography is the optical process of creating pre-designed circuit patterns on a silicon wafer, and a mask is the template used to create these patterns. As the critical dimension of the design layouts gets increasingly smaller than the optical wavelength, strong imaging distortion/displacement effect occurs that has to be corrected in the process of mask design.

Optical proximity correction (OPC) is a major type of mask optimization technique. Mainstream OPC approaches include rule-based [1] and model-based OPC [2], [3]. Rule-based OPC corrects the mask pattern based on predefined rules or empirical formulas, which, however, are difficult to generalize to complex patterns. Model-based OPC simulates the imaging process of the optical system using physical models, allowing for more precise correction of distortions in mask patterns, but this method may suffer from limited solution space since the design patterns are optimized segment by segment [4]. Inverse lithography technology (ILT) [5] is an advanced mask optimization method used to address the challenges of shrinking feature size. ILT generates an optimized mask pattern by inversely solving the lithography process. Different from conventional OPC, ILT performs optimization at the pixel level, resulting in stronger flexibility and better fidelity. One of the major technical bottlenecks limiting the widespread application of ILT is the high computational demand and low efficiency.

With the development of artificial intelligence technology, deep-learning-based ILT algorithms reformulates the mask optimization problem as an image-to-image translation problem. Namely, pre-trained neural network models predict mask patterns from target layouts directly, eliminating the need for solving complex physical models. Additionally, by utilizing GPU hardware acceleration, the computational efficiency is improved. Several state-of-the-art (SOTA) deep learning based ILT methods have shown improved performance compared to conventional ILT approaches, such as GAN-OPC [6], Neural-ILT [7], DevelSet [8], and A2-ILT [9]. A common workflow for these methods requires the pretraining of a deep learning model in a supervised manner, providing the model with thousands of design layout/mask pattern data pairs to learn the underlying mapping relationship. The output of the trained model serves as an initial guess for further ILT optimization to produce the final mask. This workflow is not computationally efficient. On the other hand, the supervised model training process may have limited fidelity or generalization ability when the training dataset is not adequate.

In this paper, we propose a self-supervised deep learning based ILT model (SSDL-ILT) for mask optimization with improved fidelity and efficiency. The contribution of our work is fivefold:

(1) a novel self-supervised learning strategy is proposed utilizing a tailored loss function, which only requires design patterns as the training dataset;

(2) end-to-end mask generation is realized without the need of further ILT correction;

(3) sub-resolution assist features (SRAFs) are generated automatically, the complexity of which can be adjusted in the model training stage;

(4) significant speedup is achieved at both the model training and inference stages compared to other SOTA deep-learning-based ILT models;

(5) good generalization ability is demonstrated via a few-shot transfer learning approach.

The rest of the paper is organized as follows. Section II provides an introduction of the related lithography simulation background, mask optimization problem formulation, and the corresponding evaluation criteria. Section III describes in detail our proposed self-supervised deep learning based ILT algorithm. Section IV provides the experimental results, and conclusion is provided in Section V.

II. BACKGROUND

A. Forward lithography simulation

The forward lithography simulation computes the wafer image from the mask. It consists of an optical model and a photoresist model. The optical model produces an aerial image $\mathbf{I}$, which represents the light intensity distribution on the wafer surface. For a partially coherent optical system, the commonly used optical model is the Hopkins diffraction model [10]. By simplifying the model through singular value decomposition, a series of optical convolution kernels $\mathbf{h}$ can be obtained. The spatial image can be computed through the convolution operation between the mask image $\mathbf{M}$ and the optical convolution kernel $\mathbf{h}$:

$$\mathbf{I}(x,y) = \sum_{k=1}^{N} \omega_k |\mathbf{M}(x,y) \otimes \mathbf{h}_k(x,y)|^2, \quad (1)$$

where $\mathbf{h}_k$ is the kth optical kernel and ω_k is the corresponding weight; N is the total number of optical kernels used in the simulation, and here we set $N = 24$; $\otimes$ denotes the convolutional operation.

The photoresist model converts the continuous light intensity image $\mathbf{I}$ into a binary wafer image $\boldsymbol{Z}$. Here, we adopt a simplified threshold-based photoresist model. To smooth the wafer image allowing effective loss back-propagation in the deep learning model training process, a sigmoid function is further applied:

$$\boldsymbol{Z} = \frac{1}{1+\exp[-(\mathbf{I}-I_{th})]}, \quad (2)$$

where I_{th} is the light intensity threshold. Here, I_{th} is set to 0.225 consistent with other related works [6, 7].

B. Mask optimization problem formulation

Given a target layout $\mathbf{Z}_t$, the goal of mask optimization is to obtain an optimized mask $\mathbf{M}$ that could produce a wafer image as close to $\mathbf{Z}_t$ as possible, with acceptable runtime, low process variation band, and acceptable printability and manufacturability. The deep-learning-based mask optimization strategy is to replace the physics-based optimization process with a pre-trained surrogate model, which predicts the mask pattern directly from the target layout.

C. Evaluation metrics

***Definition** 1* (Squared L_2 error). Squared L_2 error is calculated by $\|\mathbf{Z}_t - \mathbf{Z}_{norm}\|_2^2$, given the target image $\mathbf{Z}_t$ and the wafer image $\mathbf{Z}_{norm}$. Here $\mathbf{Z}_{norm}$ denotes the wafer image under nominal dose and focus conditions.

***Definition** 2* (Process variation band, PVB). PVB denotes the summation of bitwise-XOR between two wafer images $\mathbf{Z}_{min}$ and $\mathbf{Z}_{max}$, which are generated under $\pm 2\%$ dose conditions, using the same settings as in the ICCAD-2013 CAD contest [11].

***Definition** 3* (Turnaround time, TAT): TAT is defined as the time elapsed during the model application for one mask optimization case, including the model prediction time and (if any) the ILT re-optimization time.

III. ALGORITHMS

A. Deep learning model for mask generation

We adopt a deep learning model to perform end-to-end mask optimization. An attention-based residual recurrent U-Net model (R2U-Net) is used which takes the target layout as input and outputs the corresponding mask pattern that would produce the target design. The attention R2U-Net model is a combination of R2U-Net [12] and attention U-Net [13]. The network architecture is shown in Fig. 1. In general, the U-Net model is composed of an encoder and decoder. The encoder down-samples the input target layout through convolutional and pooling operations to extract high-level features, whereas the decoder up-samples the extracted features and restore the image to the original resolution. Skip-connection [14] is applied at each up-sampling stage that fuses the features from the corresponding down-sampling stage to improve the model's memory of lower-level information. Attention mechanism [13] is added to the decoder so that the model learns the importance weights of each channel and attend to most related regions in the images to improve feature generation. Furthermore, the recurrent residual convolutional layers ensures better feature representation for training deep architecture. In the final layer of the decoder, a Sigmoid function is applied that restricts the output values to a range of 0-1, aligning with the value range of a binarized image.

B. Self-supervised learning strategy

Unlike traditional supervised deep learning models, the training process of our model is unsupervised, which is not guided by the data mismatch between predicted and referenced mask patterns. A novel loss function is constructed to incorporate physical guidance into the modeling training process. The loss function consists of three terms used to evaluate the quality of the generated masks. Given a series of layout patterns $(Z_{t_1}, Z_{t_2}, \ldots, Z_{t_{N_t}})$ similar in geometry and design to the target layout T, the neural network model ϕ_{NN} predicts the corresponding mask patterns

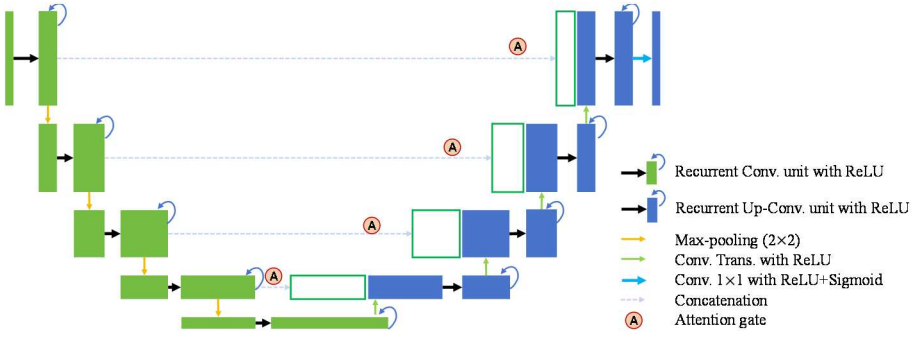


Fig. 1. Schematic of the attention R2U-Net model architecutre.

$(M_1, M_2, \dots, M_{N_t})$, which are then subject to lithographical simulation ϕ_{sim} described in Section II to obtain the corresponding wafer images $(Z_1, Z_2, \dots, Z_{N_t})$. The first term $Loss_{L_2}$ in the loss function quantifies the similarity between the simulated wafer images and the corresponding design patterns:

$$Loss_{L_2} = \frac{1}{N_t}\sum_{i=1}^{N_t}\left(Z_i - Z_{t_i}\right)^2, \tag{3}$$

where Z_t is the design pattern in the training set; N_t is the total number of design patterns used for model training; $\boldsymbol{w}$ is the parameter set of the neural network; Z is the simulated wafer image, and $Z = \phi_{sim}\left(\phi_{NN}\left(Z_{t_i}; w\right); \text{norm}\right)$. Here 'norm' indicates the dose and focus are at nominal conditions for the optical model.

The second term $Loss_{PVB}$ in the loss function evaluates the PVB of the generated mask:

$$Loss_{PVB} = \frac{1}{N_t}\sum_{i=1}^{N_t}[\phi_{sim}(\phi_{NN}(Z_t; \boldsymbol{w}); \min) - \phi_{sim}(\phi_{NN}(Z_t; \boldsymbol{w}); \max)]^2, \tag{4}$$

where 'min' indicates the wafer image generated under the nominal focus and −2% dose condition, and 'max' indicates the wafer image generated under nominal focus and +2% dose condition.

Theoretically, the mask complexity and manufacturability can be quantified by the number of shots performed during the mask creation process. However, the shots calculation can be cumbersome and time-consuming. Herein we define a new evaluation metric $Loss_{com}$, which is the third term in the loss function to heuristically quantify the mask complexity:

$$Loss_{com} = \frac{1}{N_t}\sum_{i=1}^{N_t}\left[\text{sum}(Z_{norm_i})/(H \times W)\right]^2, \tag{5}$$

where H and W indicate the image height and width in unit of total number of pixels. $Loss_{com}$ calculates the percentage of the image occupied by the mask pattern. Larger area of SRAF patterns will lead to higher value of $Loss_{com}$.

In the training process, the following total loss function is iteratively optimized:

$$\underset{\boldsymbol{w}}{\text{argmin}}\left\|\alpha Loss_{L_2} + \beta Loss_{PVB} + \gamma Loss_{com}\right\|_2^2, \tag{6}$$

where α, β, γ indicate the weight of the corresponding term. These weights can be tuned to generate mask patterns tailored to certain demand. Therefore, the model iteratively learns the physics behind mask optimization and generates masks that satisfy the defined evaluation criteria. The whole training process is in a self-supervised manner by providing the model with only certain amount of target layouts. The overall model training design is illustrated in Fig. 2.

During the training process, the backward calculation of the derivatives can be obtained as follows:

$$\begin{aligned}
\frac{\partial Loss_{L_2}}{\partial \boldsymbol{w}} &= \frac{\partial Loss_{L_2}}{\partial \mathbf{M}}\frac{\partial \mathbf{M}}{\partial \boldsymbol{w}} \\
\frac{\partial Loss_{PVB}}{\partial \boldsymbol{w}} &= \frac{\partial Loss_{PVB}}{\partial \mathbf{M}}\frac{\partial \mathbf{M}}{\partial \boldsymbol{w}} \\
\frac{\partial Loss_{com}}{\partial \boldsymbol{w}} &= \frac{\partial Loss_{com}}{\partial \mathbf{M}}\frac{\partial \mathbf{M}}{\partial \boldsymbol{w}}
\end{aligned} \tag{7}$$

$\frac{\partial Loss_{L_2}}{\partial \mathbf{M}}$, $\frac{\partial Loss_{PVB}}{\partial \mathbf{M}}$, and $\frac{\partial Loss_{com}}{\partial \mathbf{M}}$ can be calculated explicitly by deriving the lithographical model formula, the details of which can be referenced in [7]. $\partial \mathbf{M}/\partial \boldsymbol{w}$ can be obtained using the automatic differentiation feature provided by PyTorch [15]. The parameter set $\boldsymbol{w}$ of the neural network can then be updated by the following formula:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + lr\frac{\partial Loss_{L_2}}{\partial \boldsymbol{w}}, \tag{8}$$

where lr represents the learning rate of the optimizer, and n represents the n^{th} iteration step.

Additionally, an early-stop mechanism is used during the training process to prevent overfitting and improve training efficiency. The training process stops when the loss function does not further reduce over the last ten epochs.

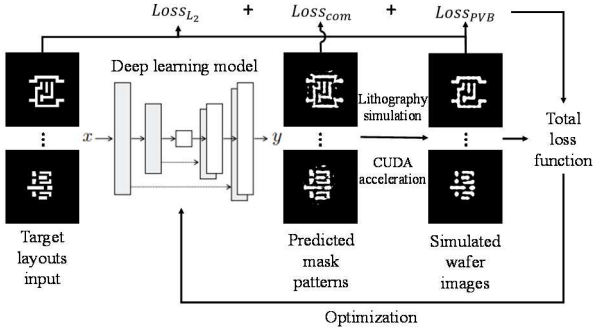


Fig. 2. Schematic of the self-supervised model training process.

C. Model inference

Once the model is adequately trained, it can be used for mask prediction given a target layout. If a target design is not in an image format (e.g. GDS files), it should first be converted into images, and the image size should be adjusted to match the input size of the model. The model's prediction result ϕ_{NN} needs to be converted into a binary image using a threshold value t.

$$\phi_{NN} = \begin{cases} 1, for\ \phi_{NN} \geq t \\ 0, for\ \phi_{NN} < t \end{cases} \tag{9}$$

Here we set $t = 0.5$. The entire self-supervised deep learning based ILT workflow is described in Algorithm 1.

Algorithm 1 The self-supervised deep learning based ILT workflow

Require: Target layout T for which a mask is to be generated, series of design patterns $\mathbf{Z}_t$ similar to T, lithography simulation model ϕ_{sim}, optical kernels $\mathbf{h}$, weights of the terms in the loss function α, β, γ, learning rate lr, maximum training epoch $E_{\max}$.

Model training:

1: Initialize deep learning model architecture ϕ_{NN} and parameters $\boldsymbol{w}$;

2: Set $itr = 0$;

3: **repeat**

4: Mask prediction: $\mathbf{M}_{pred} \leftarrow \phi_{NN}(\mathbf{Z}_t; \boldsymbol{w})$;

5: Lithography simulation under different dose/focus conditions: $\mathbf{Z}_{norm}, \mathbf{Z}_{min}, \mathbf{Z}_{max} \leftarrow \phi_{sim}(\mathbf{M}_{pred}, \mathbf{h};\ \text{norm}, \min, \max)$;

6: Loss function calculation: $Loss_{L_2} \leftarrow \frac{1}{N_t}\sum_{i=1}^{N_t}\left[Z_{norm_i} - Z_{t_i}\right]^2$, $Loss_{PVB} \leftarrow \frac{1}{N_t}\sum_{i=1}^{N_t}\left[Z_{min_i} - Z_{max_i}\right]^2$, $Loss_{com} \leftarrow \frac{1}{N_t}\sum_{i=1}^{N_t}\left[\mathrm{sum}(Z_{norm_i})/(H \times W)\right]^2$, $Loss \leftarrow \alpha Loss_{L_2} + \beta Loss_{PVB} + \gamma Loss_{com}$

7: Gradient calculation: $\frac{\partial Loss}{\partial \boldsymbol{w}} \leftarrow \left(\frac{\partial Loss_{L_2}}{\partial \mathbf{M}} + \frac{\partial Loss_{PVB}}{\partial \mathbf{M}} + \frac{\partial Loss_{com}}{\partial \mathbf{M}}\right)\frac{\partial \mathbf{M}}{\partial \boldsymbol{w}}$;

8: Network parameters update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + lr\frac{\partial Loss}{\partial \boldsymbol{w}}$;

9: Learning rate lr update based on cosine annealing;

10: $itr \leftarrow itr+1$;

11: **until** $itr \geq E_{max}$ or $\min(Loss_{itr-10:itr}) > \min(Loss_{0:itr})$

Model inference:

12: $M \leftarrow \phi_{NN}(T; \boldsymbol{w})$

13: mask binarization: $M = \begin{cases} 1, for\ \mathrm{M} \geq 0.5 \\ 0, for\ \mathrm{M} < 0.5 \end{cases}$

model training and evaluation is a single NVIDIA RTX 4070 GPU card, and a 13th Gen Intel(R) Core(TM) i9-13900K@3.00 GHz CPU. The training dataset is obtained from the GAN-OPC paper [6], including 4000 synthesized target layouts based on the existing 32nm M1 layout topologies. The lithography model used for model evaluation is obtained from ICCAD 2013 contest [11].

A. Baseline performance test

For each design layout image in the training set, a smallest rectangle enclosing the patterns is cropped from the original image and mapped to the center of a 2048×2048 image. This operation is to center the patterns and improve optimization efficacy. The image size is then downscaled to 256×256 via an average pooling operation to reduce the computational cost and GPU memory requirement without losing necessary details. The output image is then interpolated to the original 2048×2048 size for calculation of Squared L_2 error and PVB.

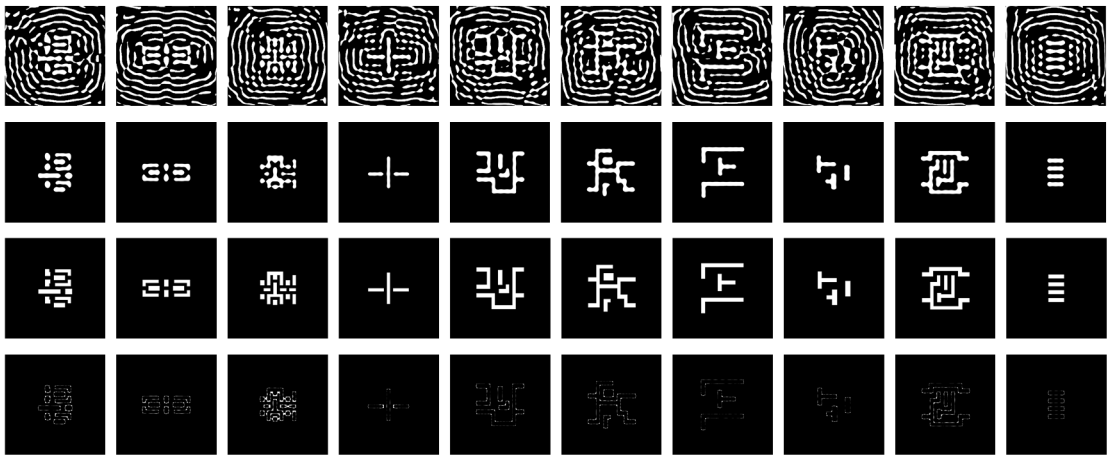


Fig. 3. The performance of our SSDL-ILT model on 10 test cases from ICCAD-2013 CAD contest. Rows from top to bottom indicate the predicted mask patterns, the corresponding simulated wafer images, the target layouts, and the absolute error between simulated wafer images and target layouts, respectively.

IV. EXPERIMENTAL RESULTS

The deep learning model is developed with PyTorch. The forward lithography model is adapted from the related module in OpenILT [16]. The entire workflow is deployed on CUDA to improve the solution efficiency. The hardware used for

α, β, γ in the loss function (6) is set to 1, 0.01, and 0, respectively. Adam is used as the optimizer and the learning rate is set to 0.0001. A cosine annealing learning rate decay scheduler is applied to improve loss convergence. The maximum training epoch is set to 50. The loss function converges after 32 epochs and the training takes about two

Table I. Performance comparison on 10 test cases from ICCAD-2013 CAD contest between our model and other SOTA models

Test case #	SSDL-ILT			Conventional ILT [17]			Re-optimization models								
							PGAN-OPC [6]			Neural-ILT 2.0 [7]			A2-ILT [9]		
	L2 (nm^2)	PVB (nm^2)	TAT (s)	L2 (nm^2)	PVB (nm^2)	TAT (s)	L2 (nm^2)	PVB (nm^2)	TAT (s)	L2 (nm^2)	PVB (nm^2)	TAT (s)	L2 (nm^2)	PVB (nm^2)	TAT (s)
1	34570	53793	0.031	49893	65534	1280	52570	56267	358	49817	55975	11	44477	49064	4.53
2	25991	42209	0.031	50369	48230	381	42253	50822	368	38174	52010	17	34031	40165	4.5
3	68138	84245	0.026	81007	108608	1123	83663	94498	368	89411	91357	10	88901	94850	4.54
4	7658	25760	0.032	20044	28285	1271	19965	28957	377	16744	29982	9	17464	24636	4.51
5	23597	57956	0.031	44656	58835	1120	44733	59328	369	45598	58900	11	35685	50664	4.53
6	27538	51164	0.031	57375	48739	391	46062	52845	364	43836	54969	10	41432	43516	4.52
7	11920	39441	0.031	37221	43490	406	26438	47981	377	20324	50542	16	21901	37688	4.51
8	8879	22001	0.032	19782	22846	388	17690	23564	383	13337	26353	15	15257	18548	4.48
9	32600	62556	0.031	55399	66331	1138	56125	65417	383	49401	68817	11	45885	56668	4.52
10	8259	16722	0.031	24381	18097	387	9990	19893	366	8511	20734	14	11121	14478	4.5
Average	**24915**	**45585**	**0.031**	44013	50900	789	39949	49957	371	37515	50964	12.4	35615	43028	4.51
Ratio	**1.000**	**1.000**	**1.000**	1.767	1.117	25634	1.603	1.096	12071	1.506	1.118	403	1.429	0.944	147

hours. 10 test cases provided by the ICCAD 2013 contest [11] are utilized to evaluate the model performance.

We show in Fig. 3 the predicted masks using our model, the simulated wafer images using the predicted masks, target layouts, and the absolute error between simulated wafer images and target layouts. The detailed performance results for each test case are shown in Table I, in comparison with conventional ILT [17] and other SOTA deep-learning-based ILT models. Table II shows the training time spent using our self-supervised training strategy and other supervised-learning-based models reported in the literature.

In general, deep learning based ILT models outperform conventional ILT by a large margin. Among deep learning ILT models, our model has the lowest L_2 error (on average 24915), achieving on average about 30% reduction compared to other SOTA models. The wafer images generated using our optimized masks show high fidelity to the target layouts, which indicates the effectiveness of our proposed self-supervised learning strategy. PVB reported by our model is also lower than all the other listed SOTA models (on average about 10% reduction) but A2-ILT (very close, though). Such PVB reduction results from the SRAF patterns generated by our model (Fig. 3). The most significant performance improvement using our model is the total TAT (on average 0.031s). Our model outputs the final optimized mask directly without further ILT optimization, achieving inference efficiency improvement of at least ~150 times compared to A2-ILT, and at most ~12,000 times compared to PGAN-OPC. Furthermore, as shown in Table II, the model training time is significantly reduced. Although the hardware used for training is not the same across different models, we expect a speedup of at least 5 times.

TABLE II. Model training time comparison

	SSDL-ILT	Re-optimization models		
		PGAN-OPC [6]	Neural-ILT 2.0 [7]	A2-ILT [9]
Training data size	4000	4000	4300	1000
Hardware	1 NVIDIA RTX 4070 GPU	1 NVIDIA Titan X GPU	1 NVIDIA Titan V GPU	1 NVIDIA RTX 3090 GPU
Training time (h)	2	10	19	25

B. Limited training data scenario

The baseline case in the previous section uses 4000 training samples to be roughly consistent with other SOTA models for comparison. Here we test the model's learning capacity in cases of limited training data, using 100, 500, and 2000 training samples, respectively, extracted randomly from the GAN-OPC training dataset. The results are shown in Table III. For comparison, we also train a supervised attention R2U-Net model, using the same network architecture and 4000 target layouts as training inputs, whereas the corresponding reference mask patterns are provided as well. As shown in Table III, our model's performance improves with the amount of the provided training data. The worst case scenario using only 100 training samples outperforms the supervised attention R2U-Net that uses 4000 training samples, and has comparable results to other SOTA models, indicating the strong generalization ability of our self-supervised learning strategy. The model trained using 500 samples achieves better performance compared to other SOTA models. 2000 samples are enough to provide decent results comparable to 4000 samples as shown in Table I. In summary, our self-supervised learning strategy guarantees fidelity and robustness in the case of limited training data.

TABLE III. Effect of training data size on the model performance

Test case #	SSDL-ILT						Supervised attention R2U-Net	
	100 training samples		500 training samples		2000 training samples		4000 training samples	
	L_2 (nm^2)	PVB (nm^2)	L_2 (nm^2)	PVB (nm^2)	L_2 (nm^2)	PVB (nm^2)	L_2 (nm^2)	PVB (nm^2)
1	54415	53640	40226	54386	37648	54272	70836	56619
2	37683	43490	28218	42218	25940	41852	51819	48729
3	92939	72340	74839	82291	70842	81377	105215	64178
4	23192	27948	16912	29313	9503	27059	33527	23707
5	39426	56633	30741	58797	25570	58315	52156	61309
6	41365	45483	29600	50018	27245	51469	47784	54128
7	20241	38718	16382	40443	13010	39386	33947	46260
8	15700	20274	12542	21896	9822	21741	17812	24382
9	47908	56574	38284	60842	33184	61978	66007	62066
10	15652	16985	8935	16957	9013	16647	19441	19454
Average	38852	43209	29668	45716	26178	45410	49854	46083

C. Mask complexity ajustment

The weight γ in (6) controls the complexity of the mask, especially the generation of SRAFs. The baseline case is trained without any constraint of SRAF generation (γ=0). Here we show two more cases trained using γ=0.005 and 0.06. The generated masks and corresponding wafer images are shown in Fig. 4. For γ=0.005, the third term in the loss function reduces the extent of SRAF generation, leaving only necessary patterns around the center design. The average L_2 error and PVB evaluated on the 10 cases are 25772 and 46167, respectively. For γ=0.06, SRAF generation is completely inhibited which improves the mask manufacturability. Similar L_2 error is reported (26636) but PVB is significantly larger (54477). Therefore, the major role of SRAF is the improvement of process window. In practical wafer production, the balance between PVB and mask manufacturability should be delicately determined.

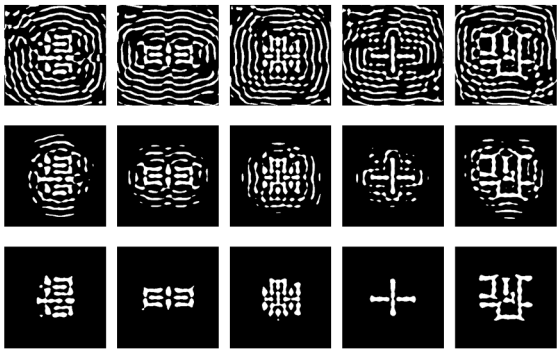


Fig. 4. Mask complexity adjustment demonstration. The rows from top to bottom are predicted masks using the trained model with γ=0, 0.005, and 0.06, respectively. For the sake of brevity, five test cases are shown.

D. Generalization ability test

We test the generalization ability of the model by applying the baseline model directly to 12 design patterns randomly extracted from the 2019 ISPD initial detailed routing contest, which are different in design and geometry from the training set. The results (5 out of 12 cases) are shown in Fig. 5. In general, good fidelity can be observed between simulated wafer images and design patterns (average L_2=61460). However, due to the existence of thicker and thinner pattern features than the training set, the wafer images obtained from the predicted masks show holes or missing/breaking thin lines in these regions. To further improve the model generalization ability, we adopt a few-shot transfer learning strategy using the pre-trained baseline model, by providing it with the remaining 7 design patterns from the 12 patterns as the training set. The network parameters are initialized to be consistent with the baseline model, and a smaller learning rate is applied. The model training is fast due to this small dataset, which completes in about two minutes after 300 epochs. The newly-trained model's performance is evaluated using the same five samples. As shown in Fig. 5, the holes in the previous wafer images are filled and feature resolution with small critical dimension is significantly improved compared to the original baseline model, which proves the strong adaptivity of our model (average L_2=40687). The few-shot transfer learning capability demonstrates potentially broad application scenarios of our model.

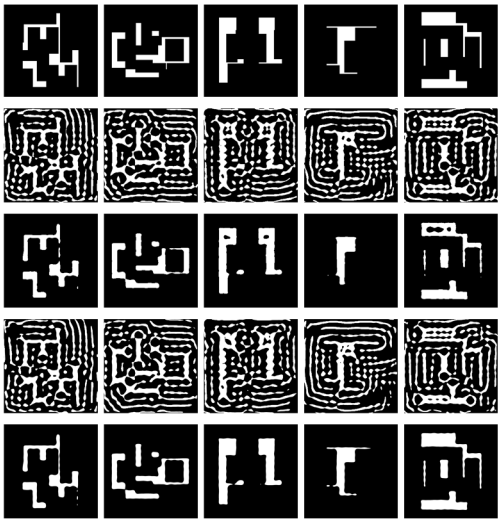


Fig. 5. Generalization ability test of the pre-trained model applied to design patterns extracted from ISPD 2019 initial detailed routing contest. The first row shows the five design patterns. The second and third rows are the predicted masks and simulated wafer images, respectively, obtained by the baseline model. The last two rows are results obtained by the few-shot transfer learning model.

V. CONCLUSION

In this paper, we propose a self-supervised deep learning model for direct mask prediction with high fidelity and efficiency. Critical lithography performance evaluation criteria is used to construct a novel physics-based loss function to guide the model training process. An attention-based R2U-Net architecture is applied to improve the model learning capability. Only hundreds of target layouts are required in the model training process instead of thousands of target/mask data pairs as in supervised learning based SOTA models to achieve comparable results, which effectively alleviates the data acquisition burden. SRAF patterns of varying complexity can be generated automatically during the inference process, which effectively improves PVB. Experimental results on ICCAD-2013 CAD contest dataset prove better fidelity and efficiency of our model compared to other SOTA deep-learning-based ILT models. A few-shot transfer learning strategy applied on the ISPD 2019 initial detailed routing contest dataset shows that the pre-trained model can be efficiently adapted to predict masks from different layout structures with high fidelity.

REFERENCES

[1] J.-S. Park *et al.*, "An efficient rule-based OPC approach using a DRC tool for 0.18 /spl mu/m ASIC," in *Proceedings IEEE 2000 First International Symposium on Quality Electronic Design (Cat. No. PR00525)*, Mar. 2000, pp. 81–85.

[2] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. Banerjee, "Fast lithographic mask optimization considering process variation," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2014, pp. 230–237.

[3] J. Kuang, W.-K. Chow, and E. F. Y. Young, "A robust approach for process variation aware mask optimization," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2015, pp. 1591–1594.

[4] B. Jiang, H. Zhang, J. Yang, and E. F. Y. Young, "A fast machine learning-based mask printability predictor for OPC acceleration," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. Jan. 2019, pp. 412–419.

[5] A. Poonawala and P. Milanfar, "Mask design for optical microlithography—An inverse imaging problem," *IEEE Trans. Image Process.*, vol. 16, no. 3, pp. 774–788, Mar. 2007,

[6] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Y. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, Jun. 2018, pp. 1–6.

[7] B. Jiang, L. Liu, Y. Ma, B. Yu and E. F. Y. Young, "Neural-ILT 2.0: Migrating ILT to domain-specific and multitask-enabled neural network," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2671-2684, Aug. 2022.

[8] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.

[9] Q. Wang, B. Jiang, M. D. F. Wong, and E. F. Y. Young, "A2-ILT: GPU accelerated ILT with spatial attention mechanism," in *ACM/IEEE Design Automation Conference (DAC)*, Aug. 2022, pp. 967–972.

[10] H. H. Hopkins, "The concept of partial coherence in optics," *Proc. R. Soc. Lond. Ser. Math. Phys. Sci.*, Aug. 1951.

[11] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2013, pp. 271–274.

[12] M. Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari, "Recurrent residual convolutional neural network based on U-Net (R2U-Net) for medical image segmentation," May, 2018, *arXiv*.

[13] O. Oktay *et al.*, "Attention U-Net: Learning where to look for the pancreas," May, 2018, *arXiv*.

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," May, 2015, *arXiv*.

[15] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," Dec. 2019, *arXiv*.

[16] S. Zheng, B. Yu, and M. Wong, "OpenILT: An open source inverse lithography technique framework," in *2023 IEEE 15th International Conference on ASIC (ASICON)*, Oct. 2023, pp. 1–4.

[17] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *ACM/IEEE Design Automation Conference (DAC)*, 2014, pp. 52:1–52:6.

ChatLS: Multimodal Retrieval-Augmented Generation and Chain-of-Thought for Logic Synthesis Script Customization

Haisheng Zheng[1†], Haoyuan Wu[2], Zhuolun He[2,3†]
[1]Shanghai Artificial Intelligence Laboratory [2]The Chinese University of Hong Kong [3]ChatEDA Tech

***Abstract*—Large Language Models (LLMs) have demonstrated significant potential in automating the Electronic Design Automation (EDA) process through effective integration with EDA tools. This paper targets the customization of logic synthesis scripts, which is crucial for accommodating the unique characteristics of each design in the EDA workflow. The proposed framework, called ChatLS, integrates multimodal retrieval-augmented generation (RAG) and chain-of-thought (CoT) reasoning, enabling LLMs to collaboratively analyze design features and precisely customize synthesis scripts. Experimental results demonstrate that ChatLS has achieved superior performance in customizing synthesis scripts with a commercial logic synthesis tool.**

I. Introduction

Electronic design automation (EDA) encompasses a suite of software tools designed for the creation, analysis, and verification of electronic systems. These tools have evolved to support intricate design flows and cater to the complex requirements of advanced semiconductor manufacturing. To enhance the user experience and development efficiency, application engineers are deployed by vendors to offer tailored on-site support. However, the significant costs associated with training and manpower have led to considerations of automating support services, eliminating the need for human involvement.

Recent advancements in large language models (LLMs) have prompted the proposal of various techniques that utilize LLMs to enhance the usability of EDA tools. For instance, addressing the challenge of navigating extensive and intertwined functionalities in EDA tool documentation, several studies [1]–[3] have developed customized retrieval-augmented generation (RAG) frameworks, which improve the accuracy of LLM-generated responses, enabling the efficient retrieval of relevant information to resolve user queries. Additionally, the complexity of numerous EDA commands can overwhelm users. In response, studies [4], [5] have fine-tuned LLMs specifically for the EDA domain, facilitating the automated generation of tool call scripts based on user requirements.

While these developments represent significant advances in harnessing LLMs to assist users with EDA tools, there remains room for further enhancement. As for logic synthesis tools, a crucial aspect is the integration of user designs and target libraries with insights derived from logic synthesis tool reports. Commercial logic synthesis tools [6], [7] offer numerous optional commands aimed at improving design quality, which need to be selected based on user preferences and the analysis of the design [8], [9]. Additionally, the constraint settings included in synthesis scripts must be based on the characteristics of the design and the target library [10]. Moreover, logic synthesis is generally an iterative process, not a one-time execution. After the initial synthesis, users may employ resynthesis strategies to resolve violations within the netlist or to enhance netlist quality, with strategy choices informed not only by the design and target library but also by insights from logic synthesis tool reports [10]. Therefore, the customization of synthesis scripts, informed by a comprehensive understanding of the designs, target libraries, and the feedback from logic synthesis tool reports, is essential.

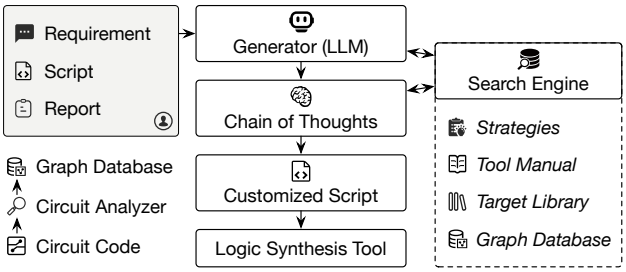


Fig. 1 An Illustration of ChatLS.

To achieve this goal, several challenges need to be addressed. The primary challenge involves a comprehensive analysis of design characteristics at both global and local levels to inform script creation. Global characteristics, such as the architecture and hierarchy of the design, guide the selection of appropriate compilation strategies [10], while local characteristics, including critical path analysis, inform constraint specifications in the scripts. Recent research [11]–[13] has demonstrated the capability of LLMs to understand register-transfer level (RTL) code snippets, suggesting potential applications in analyzing circuit design characteristics for customizing scripts. However, LLMs have significant limitations when processing long-context inputs, which affects their effectiveness in analyzing large codebases [14] and executing lengthy reasoning sequences [15].

Numerous logic synthesis algorithms [16]–[18] use graph-based representations of circuit designs to facilitate analysis and optimization. This graph-based representation allows for efficient exploration of design characteristics through a structured representation of the topology. Additionally, the graph-based representations of circuit designs have enabled numerous studies to leverage graph neural networks (GNNs) to extract meaningful characteristics from complex circuits [19]–[24]. Thus, leveraging graph-based analysis methods may help the LLMs overcome challenges in understanding complex circuit designs.

The second major challenge involves selecting synthesis commands within scripts that align with specific design characteristics. For instance, consider a scenario where a design requires timing optimization. Techniques such as retiming [25] and buffer balancing, both integrated into logic synthesis tools, can effectively reduce path delays. However, choosing the appropriate technique requires thorough consideration. Retiming is particularly effective for designs with extended critical paths that experience timing violations due to unbalanced register placement or excessively long combinational logic. Alternatively, buffer balancing is advantageous for mitigating timing issues in high-fanout nets. Therefore, a careful assessment of the strengths of each technique is essential for optimizing timing.

Techniques like retrieval-augmented generation (RAG) [26] and chain-of-thought (CoT) [27] reasoning have proven effective in assisting LLMs with complex decision-making tasks. In our context, RAG could help the LLMs retrieve relevant information and context

† Corresponding authors.

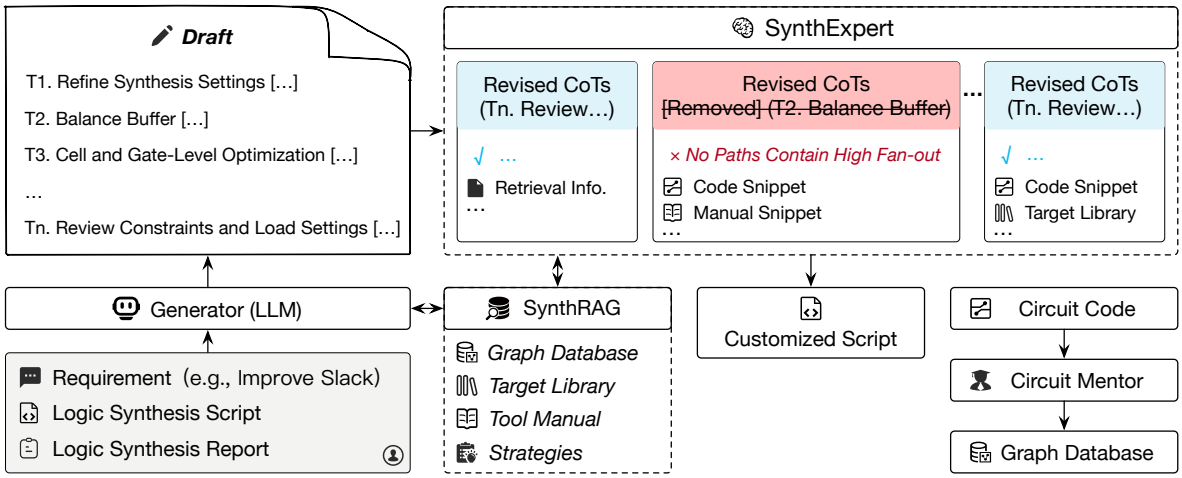


Fig. 2 The Overall Flow of ChatLS.

from circuit design snippets, target library, tool manuals, or past use cases, providing a well-informed foundation for decision-making. CoT then enables step-by-step reasoning to evaluate the trade-offs and select the most suitable solution from multiple options. It guides the LLMs through a logical progression of decisions, considering the impact of each command on design performance, ensuring that the final synthesis script is both optimal and aligned with user preferences.

In this paper, we present ChatLS, a framework for customizing logic synthesis scripts to meet specific requirements expressed in natural language. As illustrated in Fig. 1, ChatLS consists of four main components: a circuit analyzer for assisting LLM analyze and understand circuit designs, a search engine for retrieving relevant information, a CoT for decision-making during script refinement, and an LLM that interprets and generates synthesis scripts from natural language instructions.

Our major contributions are summarized as follows:

- We propose a graph-based method that enhances the ability of LLMs to analyze and understand circuit designs by transforming them into graph databases, with GNN supporting the analysis process.
- We introduce a domain-specific multimodal RAG framework that efficiently retrieves information to enhance LLMs for customizing logic synthesis scripts.
- We present a task-specific CoT mechanism, in tandem with the multimodal RAG framework, that enables iterative refinement of synthesis scripts by revising reasoning steps based on retrieved data.
- We conduct comprehensive experiments, demonstrating that ChatLS achieves superior performance in customizing logic synthesis scripts.

The remainder of this paper is structured as follows: Section II provides the preliminary background. Section III presents the problem formulation. The detailed description of the ChatLS framework is discussed in Section IV. The experimental results are presented and analyzed in Section V. Finally, conclusions are drawn in Section VI.

II. Preliminaries

Abstract Syntax Tree (AST) represents the hierarchical structure of source code, where each node corresponds to a specific construct or element in the program. It enables efficient analysis, transformation, and manipulation by abstracting the logical structure while omitting unnecessary syntactic details. ASTs are fundamental in compiler design, serving as an intermediate representation for syntax analysis, semantic validation, and optimization. They are also extensively used in static code analysis, automated reasoning, and code refactoring, effectively bridging raw code with its logical representation and enhancing software quality.

Retrieval-Augmented Generation (RAG) [26] targets the problem of enhancing natural language generation by combining the strengths of retrieval-based and generative approaches to improve both knowledge and contextual accuracy, making it effective for LLMs. RAG augments the generative process with a retriever module that dynamically retrieves relevant information $\{r_i\}_{i=1}^{k}$ from an external resource $\mathcal{R}$, such as a document corpus or knowledge base, based on an input query q. The retrieved information is then utilized by a generative model to produce a response y. In RAG, the objective is to maximize the conditional probability $P(y|q)$, which can be expressed as:

$$P(y|q) = \sum_{i=1}^{k} P(y|r_i, q)P(r_i|q), \tag{1}$$

where $P(r_i|q)$ represents the probability of retrieving a piece of information r_i given the query q, and $P(y|r_i, q)$ is the probability of generating the output y conditioned on both the query and a specific retrieved piece of information. By retrieving diverse and relevant information, RAG grounds the LLMs in external knowledge, thereby improving response quality and factual accuracy in complex tasks.

Chain of Thoughts (CoT) reasoning [27] enhances the performance of LLMs in tasks requiring complex reasoning, such as multi-step math word problems. Instead of generating the final answer directly, CoT prompts LLMs to generate intermediate steps, called thoughts, that serve as a scratch space guiding the model towards the solution. Given an input x, CoT introduces intermediate steps $s_1, s_2, \ldots, s_k$, forming a path from input to output:

$$x \rightarrow (s_1, s_2, \ldots, s_k) \rightarrow y, \tag{2}$$

this explicit reasoning framework helps LLMs break down complex tasks into manageable components, improving both accuracy and interpretability.

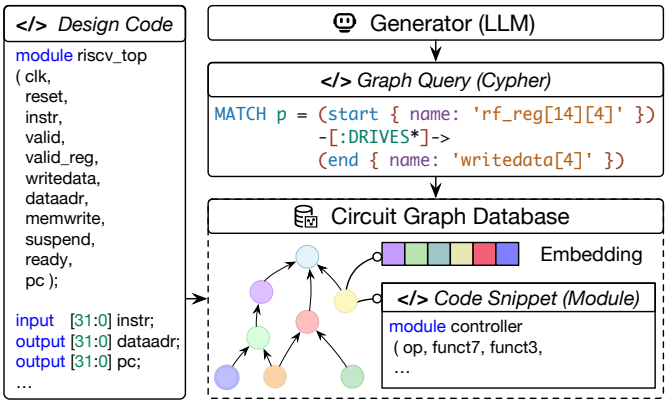


Fig. 3 Workflow Visualization of CircuitMentor.

III. The Scope of ChatLS

ChatLS is an innovative framework that provides a conversational interface, allowing users to employ natural language to customize synthesis scripts tailored to the specific requirements of each design. By interpreting both the design code and user requirements, ChatLS generates synthesis scripts that align with particular design attributes and user preferences.

Problem 1 (Objective of ChatLS)**.** *Design novel methodologies to enhance the capability of LLMs in customizing logic synthesis scripts to meet user preferences and address specific design characteristics.*

IV. Algorithms

Fig. 2 illustrates the overall workflow of ChatLS. The process begins with the user providing the requirements, circuit design code, corresponding logic synthesis scripts, and reports generated by the logic synthesis tool. Based on this input, the following steps are carried out to generate a customized logic synthesis script for the design:

- **CircuitMentor** is a circuit design analysis tool that transforms the circuit design into a graph representation and employs GNN to extract high-level features from the circuit design.
- **SynthRAG** is a multi-modal RAG framework used to retrieve relevant information for LLMs when customizing logic synthesis scripts.
- **Generator (LLM)** drafts a customized logic synthesis script based on user requirements, incorporating the reports from the logic synthesis tool, the compilation and optimization strategies retrieved by **SynthRAG**, and high-level information matched by **SynthRAG** using embeddings extracted from the circuit design by **CircuitMentor**.
- **SynthExpert** is a CoTs mechanism that uses **SynthRAG** to fetch additional information and iteratively evaluates the suitability of the drafted logic synthesis script for the circuit design. During this process, **SynthRAG** is prompted to provide code for netlist paths, high-level information for corresponding modules, usage details for commands in the logic synthesis manual, and specifics of the target library. These elements ensure the generated logic synthesis script complies with design requirements and command specifications.

A. CircuitMentor: Graph-Based Assistant for Circuit Analysis

Circuit Representation. LLMs encounter significant challenges in comprehending complex circuit code, as discussed in Section I. Given that circuits can naturally be represented as graphs, we introduce CircuitMentor, which exploits this graph-based representation to enhance the capacity of LLMs to customize logic synthesis scripts.

As visualized in Fig. 3, the circuit code is transformed into a graph, specifically an AST. This graph structure is hierarchical, with nodes representing different levels of the circuit design. At the highest level, the graph contains nodes representing the overall design, with subnodes corresponding to individual modules. Each module node stores its associated Verilog code, enabling LLMs to analyze the structure and functionality of paths when customizing scripts. Within each module, subnodes represent the circuit components, such as gates and registers. This hierarchical organization allows for efficient traversal and multi-level analysis of the design. Customizing logic synthesis scripts often requires analyzing critical paths within the circuit. Therefore, our goal is to identify an appropriate graph database to store the circuit graph, enabling efficient path extraction and analysis. Neo4j [28] is well-suited for this purpose, as it supports Cypher [29], a query language that can be leveraged to extract paths and subgraphs from circuit graphs, facilitating efficient graph traversal and analysis.

Global Circuit Feature Extraction. Understanding the characteristics of a circuit is crucial for customizing logic synthesis scripts. Different circuit components, such as arithmetic and memory-based modules, require distinct compilation strategies. Arithmetic components focus on optimizing speed and area, while memory components prioritize access time and power efficiency. Tailoring the compilation strategy to the function of each component is essential for achieving overall circuit optimization. Nevertheless, effectively conveying these characteristics through LLMs presents significant challenges, as previously outlined in Section I.

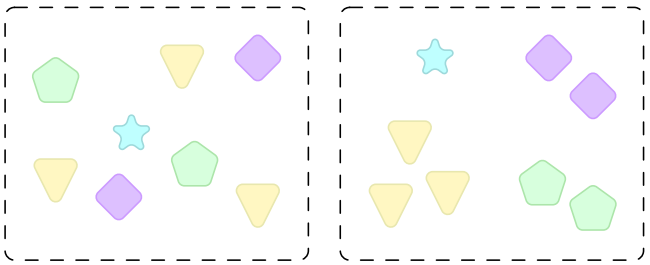
Since the circuit can be represented as a graph, GNNs can be employed to extract meaningful features from its design. We propose a hierarchical GNN based on GraphSAGE [30] to learn embeddings for the circuit. By treating each module as a subgraph, our approach captures the unique characteristics of each module. This hierarchical approach is crucial for capturing both local and global structural patterns within the circuit design. GraphSAGE aggregates features from neighboring nodes and their respective subgraphs, facilitating the extraction of hierarchical features. The key operation of GraphSAGE is defined as:

$$\boldsymbol{h}_v^{(k)} = \sigma\left(\boldsymbol{W}^{(k)} \cdot \text{Aggregator}\left(\{\boldsymbol{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\}\right)\right), \quad (3)$$

where $\boldsymbol{h}_v^{(k)}$ represents the embedding of node v at the k-th iteration, $\mathcal{N}(v)$ denotes the set of neighboring nodes of v, and Aggregator refers to the aggregation function. This aggregation step effectively encodes local circuit structural features into a compact and informative node representation.

The type of circuit design also plays a crucial role in customizing logic synthesis scripts. In cases where the design consists of a single module or has been flattened for synthesis, the graph representation collapses into a single node, rendering it ineffective for GNN-based operations. To address this, global pooling strategies are employed to obtain a global embedding. The global embedding is computed as the mean of all node embeddings in the design, given by $\boldsymbol{z}global = \frac{1}{N}\sum i = 1^N \boldsymbol{h}_i$. where N is the total number of modules in the circuit. This ensures that even when the design is flattened or consists of a single module, a meaningful global feature representation for the entire design can still be derived.

Metric Learning. Since our hierarchical GNN aims to extract



(a) (b)

Fig. 4 Illustration of the Metric Learning Process. (a) Initially, the embeddings are randomly scattered across the space. (b) After training, embeddings of similar data points converge, while embeddings of dissimilar data points diverge, forming distinct clusters.

embeddings from modules in the circuit graph, we need first address the challenge that topology-based approaches may fail to identify identical components due to structural variations. Given that GNNs are inherently topology-dependent, this poses a considerable challenge.

To address this, we adopt a metric learning approach, which focuses on learning a similarity function or distance metric that ensures similar designs are close in the embedding space, while dissimilar designs are placed farther apart. Various loss functions, such as contrastive loss [31] and multi-similarity loss [32], are commonly used to optimize the embedding space. These methods encourage the model to learn an embedding space where the geometry of the embeddings aligns with the desired similarity relationships between designs. Fig. 4 illustrates the evolution of embeddings during training. Initially, the embeddings are dispersed across the space, but after training, similar designs are pulled closer together, while dissimilar ones are pushed farther apart. This enhances the discriminative power of the embeddings, making them more effective for tasks such as design retrieval and similarity comparison.

By learning these discriminative embeddings, we capture the underlying structure of designs, enabling the efficient retrieval of relevant circuit designs based on their embeddings.

B. SynthRAG: Multimodal RAG for Domain-Specific Retrieval

The RAG framework substantially improves the generative capabilities of LLMs. Building on this advancement, we present SynthRAG, a domain-specific, multimodal RAG framework designed to enhance the ability of LLMs in customizing logic synthesis scripts. To support this goal, we first establish a database that stores the relevant information. TABLE I provides a summary of the different retrieval types, their corresponding representations, and the query methods employed in our approach.

Graph Embedding-Based Retrieval. In SynthRAG, graph embeddings serve as the foundation for information retrieval, linking related circuit designs with their respective compilation and optimization strategies. These embeddings are computed using CircuitMentor, which generates vector representations of circuit designs or modules. The resulting embeddings are used to query the graph database, retrieving relevant circuit designs with their respective compilation and optimization strategies.

Given a query embedding, $\boldsymbol{z}_{query}$, the retrieval process relies on nearest neighbor search over the database of graph embeddings. The nearest neighbor search is formalized as:

$$\boldsymbol{z}_{Retrieved} = \arg\min_{\boldsymbol{z}_{db}} \left(\text{sim}(\boldsymbol{z}_{query}, \boldsymbol{z}_{db})\right), \tag{4}$$

TABLE I Summary of Query Methods.

Category	Representation	Query Method	Retrieval Content
High Level Information of Circuit Design	Graph Embedding	Join	Compile Strategy Optimization Strategy
Code of Circuit Design	Graph Structure	Direct	The code of the module where the path is located
Target Library	Graph Structure	Direct	Gate Cell Information
Logic Synthesis Tool User Manual	LLM Embedding	Direct	Command Usage Command Requirement

where $\boldsymbol{z}_{db}$ represents the embeddings of designs in the database, and sim denotes the cosine similarity function. This ensures that the most semantically similar designs to the query are retrieved, facilitating efficient information retrieval for logic synthesis.

In RAG frameworks, reranking plays a crucial role in refining retrieved results to better fit the requirements of the target domain. Existing reranking methods [33], [34], commonly used in text-based retrieval, cannot be directly applied to our graph embeddings due to the unique characteristics of circuit designs. For instance, although both an arithmetic logic unit (ALU) and a systolic array are categorized as arithmetic components, they differ considerably in terms of scale and complexity. To address this, we introduce a domain-specific reranking function that incorporates additional criteria, such as timing, area, and power consumption, to reorder the retrieved graph embeddings.

Let $\boldsymbol{z}_{retrieved} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_K\}$ represent the set of Top-K retrieved embeddings, and let $c_1, c_2, \ldots, c_K$ denote their corresponding performance metrics (e.g., timing, area, power). The reranking function, $f(\boldsymbol{z}_{retrieved}, \boldsymbol{c})$, computes a new ranking score for each design by considering both the similarity of embeddings and the additional characteristics. The reranking is computed as:

$$\text{Score}(\boldsymbol{z}_i) = \alpha \cdot \text{sim}(\boldsymbol{z}_{query}, \boldsymbol{z}_i) + \beta \cdot c_i, \tag{5}$$

where $\text{sim}(\boldsymbol{z}_{query}, \boldsymbol{z}_i)$ is the cosine similarity between the query embedding $\boldsymbol{z}_{query}$ and the retrieved embedding $\boldsymbol{z}_i$, c_i is the domain-specific characteristic for design i, and α and β are weighting parameters that balance the contribution of similarity and characteristics in the final ranking.

The reranking process selects the designs with the highest scores, prioritizing those that meet the functional requirements and resource constraints of the query. This ensures that the retrieved designs are not only semantically similar to the query but are also optimized according to specific design criteria.

Graph Structure-Based Retrieval. In SynthRAG, the graph-based retrieval system utilizes Cypher queries, which can be generated by LLMs, to extract specific design details and information about the target library from the database. Each query retrieves a single result, thereby eliminating the need for reranking. The target library is integrated into the database to avoid inefficiencies associated with directly inputting it into the LLMs for processing.

LLM Embedding-Based Retrieval. Several studies, such as [1], have adapted RAG frameworks to retrieve information from EDA user manuals, which contain hybrid information. In our context, we focus exclusively on retrieving descriptions of logic synthesis commands within the user manual. Accordingly, SynthRAG utilizes the `text-embedding-3-large` model[1] to transform the user manual into text embeddings. Subsequently, to enhance the accuracy

[1]https://platform.openai.com/docs/guides/embeddings

TABLE II Overview of Hardware Designs in the Database.

Category	Components
Processor Core	Rocket [36], Sordor [37]
Machine Learning Accelerator	NVDLA [38], Gemmini [39]
Vector Arithmetic	SIMD [40]
Signal Processing	FFT [41]
Cryptographic Arithmetic	SHA3 [37]

of the retrieved information, SynthRAG employs `GPT-4o` [35] as a reranker.

C. SynthExpert: Iterative Customization with CoT and RAG

Although RAG and CoT techniques can significantly enhance LLMs by facilitating complex decision-making tasks, their integration into ChatLS demands careful consideration.

Despite **RAG** is effective in general applications, it faces limitations in handling complex reasoning tasks requiring multi-step reasoning. These tasks are difficult to encapsulate into simple search queries, often leading to challenges in retrieving relevant information, thus limiting the applicability of RAG. Specifically, RAG approaches typically retrieve all pertinent information in a single step, disregarding the dynamic and iterative nature of information requirements throughout the synthesis script customization process. In our context, logic synthesis script customization involves multi-dimensional data, including design characteristics, user manuals, and the target library. Retrieving all relevant information at once may overwhelm the LLMs, complicating the prioritization and application of the most pertinent data at each reasoning step. Consequently, relying solely on the initial task prompt often lacks the depth needed to effectively guide subsequent reasoning and generation, leading to suboptimal retrieval performance.

Similarly, **CoT** facilitates LLMs in breaking down complex tasks into manageable steps, which is crucial for generating accurate and optimized synthesis scripts. However, without explicit guidance, LLMs may introduce errors due to incomplete domain knowledge [42] or be influenced by hallucinations [43]. In our scenario, these inaccuracies may render customized logic synthesis scripts non-executable, as LLMs could generate nonexistent or incompatible commands for the circuit design. To address these challenges, we propose SynthExpert, which synergistically integrates CoT with RAG for logic synthesis customization. As shown in Fig. 2, SynthExpert mines each reasoning step to generate the final synthesis script.

Given a task prompt I, the LLM initially generates a series of thought steps, represented as $T := \{T_i\}_{i=1}^{n}$, where T_i is the i-th thought step. In SynthExpert, these steps correspond to intermediate reasoning stages, where the synthesis script is drafted based on user inputs. Recognizing potential inaccuracies in these steps, SynthExpert utilizes RAG to revise each step sequentially.

For each thought step T_i, the LLM formulates a query Q_i by incorporating the task prompt I to retrieve information R_i pertinent to this step, thereby refining T_i in the subsequent reasoning phase. Once Q_i is formulated, the retrieval mechanism fetches the relevant information $R_i = \text{Retrieve}(Q_i)$. The retrieved information R_i is then used to revise the current thought step, producing a revised thought step$T_i^\star$:

$$T_{1:i}^\star = p_\theta(\cdot|I, T_{1:i-1}^\star, T_i, R_i) \tag{6}$$

where $p_\theta(\cdot)$ represents the probability function of the LLM generating the revised thought $T_i^\star$ based on the integrated inputs and retrieved data.

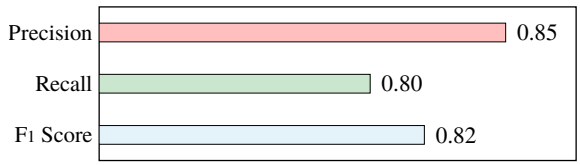


Fig. 5 Performance of SynthRAG.

This approach effectively mitigates the limitations of RAG and CoT by ensuring that each reasoning step is reinforced with the most pertinent and contextually relevant information. By revising each thought step individually, SynthExpert ensures that the LLM receives the appropriate context at every stage of the synthesis script customization process. This sequential retrieval and revision mechanism enhances the relevance and accuracy of the information used, providing richer contextual cues that align with the specialized demands of logic synthesis. Furthermore, focusing on revising individual thought steps minimizes the risk of introducing new errors into previously accurate reasoning steps, thereby preserving the integrity and reliability of the reasoning process.

As illustrated in TABLE III, integrating SynthExpert within ChatLS ensures that each aspect of synthesis script customization is informed by accurate and relevant data, thereby substantially improving the quality and reliability of the customized scripts.

V. Experimental Evaluation

ChatLS is implemented using PyTorch [49], PyTorch Geometric [50], FAISS [51], Neo4j [28], with GPT-4o (Version 2024-08-06) [35] employed as the generator. To construct the database for SynthRAG, numerous open-source designs were synthesized using the `Nangate 45nm`[2] library as the target technology, alongside the `5K_heavy_1k` wireload model, employing various optimization and compilation strategies. The corresponding logic synthesis scripts were then converted to the Design Compiler format, where they are treated as expert drafts. TABLE II presents a selection of hardware designs included in the database. Comprehensive experiments are conducted to evaluate the efficiency of ChatLS in customizing logic synthesis scripts.

A. Evaluation of SynthRAG

To evaluate SynthRAG, experiments were conducted using Chipyard [37] to generate SoCs with various configurations. These configurations provided a diverse set of circuit designs, enabling the assessment of the model's ability to retrieve relevant logic synthesis scripts and optimization strategies. The primary goal was to retrieve designs and components with high similarity to the query while considering resource usage to ensure effective customization of synthesis strategies. The F_1 score was selected for evaluation, as it balances precision and recall, making it well-suited for this task, where both false positives (irrelevant designs) and false negatives (missed relevant designs) can significantly impact performance:

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{7}$$

where Precision is $\frac{TP}{TP+FP}$ and Recall is $\frac{TP}{TP+FN}$. Here, true positive (TP) refers to correct predictions, false positive (FP) to incorrect predictions, and false negative (FN) to missed predictions. The results, summarized in Fig. 5, show that SynthRAG successfully retrieved relevant designs and modules, enabling effective customization of logic synthesis scripts.

[2]https://eda.ncsu.edu/freepdk/freepdk45

TABLE III Performance Comparison for Logic Synthesis Script Customization.

Design	GPT-4o [35]				Claude 3.5 Sonet [44]				ChatLS (Ours)			
	Timing (ns)			Area (μm^2)	Timing (ns)			Area (μm^2)	Timing (ns)			Area (μm^2)
	WNS	CPS	TNS		WNS	CPS	TNS		WNS	CPS	TNS	
aes	-0.17	-0.17	-31.64	16408.21	-0.17	-0.17	-30.70	16126.78	0.00	0.00	0.00	15919.04
dynamic_node	0.00	0.01	0.00	16327.08	0.00	4.06	0.00	20048.95	0.00	4.85	0.00	18907.28
ethmac	-0.55	-0.55	-75.89	80502.77	-0.54	-0.54	-128.99	81993.70	-0.47	-0.47	-55.72	80349.56
jpeg	0.00	0.00	0.00	57227.24	0.00	0.00	0.00	66106.85	0.00	0.00	0.00	67290.02
risv32i	0.00	0.59	0.00	10241.00	0.00	0.60	0.00	9711.66	0.00	0.74	0.00	9329.95
swerve	0.00	0.79	0.00	143557.54	0.00	1.86	0.00	158364.70	0.00	2.05	0.00	143545.30
tinyRocket	-0.21	-0.21	-42.22	35960.81	-0.33	-0.33	-484.71	41999.01	-0.09	-0.09	-14.74	36070.66

TABLE IV Performance Baseline of Various Designs.

Design	Timing (ns)			Area (μm^2)
	WNS	CPS	TNS	
aes [45]	-0.16	-0.16	-31.64	16577.12
dynamic_node [46]	-0.08	-0.08	-0.45	21155.51
ethmac [45]	-0.54	-0.54	-76.55	80533.36
jpeg [45]	-1.17	-1.17	-439.66	107612.16
risc32i [47]	0.00	0.59	0.00	10241.00
swerv [48]	0.00	0.81	0.00	161551.64
tinyRocket [36]	-0.88	-0.88	-1057.89	44231.28

B. Evaluation of ChatLS

Benchmark and Baseline. The evaluation uses design benchmarks from OpenROAD [47], as summarized in TABLE IV. The Design Compiler is employed as the logic synthesis tool. The original OpenROAD synthesis scripts are adapted to ensure compatibility with Design Compiler [6], and these adapted scripts serve as the baseline for evaluating ChatLS. This baseline provides a reference for assessing improvements or deviations in synthesis performance compared to the proposed methods.

Model Comparison. The performance of ChatLS is compared with two state-of-the-art LLMs: GPT-4o (Version 2024-08-06) [35] and Claude 3.5-Sonet (Version 2024-10-22) [44]. For this evaluation, ChatLS, GPT-4o, and Claude 3.5 each customize the logic synthesis script based on a baseline script, with only a single iteration of customization performed. To ensure a fair and consistent evaluation, all language model baselines are tested using identical prompt engineering techniques, and basic configurations, including the time period, are not allowed to change. The quality of each customized logic synthesis script was evaluated after this iteration. In cases where the design code exceeds the token limits of the models, it is partitioned into segments, each containing no more than 128,000 tokens.

The `Nangate 45nm` library is used as the target technology, along with the `5K_heavy_1k` wireload model. Timing optimization is a key focus, as it ensures sufficient slack that can be traded for improvements in area and power without compromising timing closure. Accordingly, the evaluation emphasizes ChatLS ability to customize logic synthesis scripts for optimal timing.

Evaluation Metrics. The evaluation employs the following key metrics to assess the synthesis script quality:

- **Timing**: Measures the ability of the design to meet operational speed requirements, including worst negative slack (WNS), critical path slack (CPS), and total negative slack (TNS).
- **Area**: Assesses the total hardware resources utilized by the synthesized circuit.

These metrics offer a thorough evaluation of the quality of logic synthesis scripts.

Evaluation Results The evaluation results for ChatLS are summarized in TABLE III. All models, including GPT-4o, Claude 3.5-Sonet, and ChatLS, improve design timing relative to the baseline. ChatLS consistently delivers the best timing performance, showing the most substantial improvements across all evaluated designs. This underscores the superior capability of ChatLS in customizing logic synthesis scripts for timing optimization. Both `ethmac` and `tinyRocket` exhibit timing violations, as only a single iteration was executed. However, logic synthesis is inherently an iterative process, not a one-time execution. Additional iterations are required to further resolve timing issues. While GPT-4o achieves the best area results in some cases, the primary objective of this evaluation is timing improvement, a goal that is not fully satisfied by GPT-4o. In contrast, ChatLS produces the highest-quality synthesis scripts, outperforming the baseline LLMs and demonstrating its effectiveness in customization.

VI. Conclusion

It is crucial to customize synthesis scripts, guided by a thorough understanding of design, target libraries, and feedback from logic synthesis tool reports. To address this need, we propose methods to enhance the ability of LLMs to customize synthesis scripts. To improve the ability of LLMs to understand circuit designs, we introduce CircuitMentor, which efficiently extracts the global characteristics of designs while assisting LLMs in understanding their local characteristics. Additionally, to enhance the ability of LLMs to customize high-quality logic synthesis scripts, we present SynthRAG, which supplies relevant information for script customization. Futhermore, to aid LLMs in selecting the appropriate commands within logic synthesis tools based on design and user specifications, we introduce SynthExpert, which works in tandem with SynthRAG. Experimental results demonstrate that the proposed framework, ChatLS, effectively and efficiently customizes synthesis scripts, aligning with user preferences and optimizing design performance. Future work could explore extending this framework to collaborate with a wider array of logic synthesis tools, such as PrimePower [52], to further advance the automation of the logic synthesis workflow.

Acknowledgment

This work is supported by Shanghai Artificial Intelligence Laboratory (JF-P23KK00071-1-DF). The authors would like to thank Professor Bei Yu at The Chinese University of Hong Kong for helpful discussions and comments.

REFERENCES

[1] Y. Pu, Z. He, T. Qiu, H. Wu, and B. Yu, "Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA," in *Proc. ICCAD*, 2024.

[2] L. Shi, M. Kazda, B. Sears, N. Shropshire, and R. Puri, "Ask-EDA: A Design Assistant Empowered by LLM, Hybrid RAG and Abbreviation De-hallucination," in *arXiv preprint*, 2024.

[3] A. Kaintura *et al.*, "ORAssistant: A Custom RAG-based Conversational Assistant for OpenROAD," in *arXiv preprint*, 2024.

[4] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "ChatEDA: A Large Language Model Powered Autonomous Agent for EDA," in *IEEE TCAD*, 2024.

[5] M. Liu *et al.*, "ChipNeMo: Domain-Adapted LLMs for Chip Design," in *arXiv preprint*, 2024.

[6] Synopsys, Inc., *Design Compiler: Concurrent Timing, Area, Power, and Test Optimization*. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.

[7] Cadence, Inc., *Genus Synthesis Solution: Delivering the Best Possible Productivity During RTL Design and the Highest Quality of Results (QoR) in Final Implementation*. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html.

[8] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, and L. P. Carloni, "A Synthesis-Parameter Tuning System for Autonomous Design-Space Exploration," in *Proc. DATE*, 2016.

[9] H. Bhatnagar, *Advanced ASIC Chip Synthesis: Using Synopsys® Design Compiler™ and PrimeTime®*. 2002.

[10] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. 1997.

[11] X. Yao, Y. Wang, X. Li, Y. Lian, R. Chen, L. Chen, M. Yuan, H. Xu, and B. Yu, "RTLRewriter: Methodologies for Large Models aided RTL Code Optimization," in *Proc. ICCAD*, 2024.

[12] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards LLM-Powered Verilog RTL Assistant: Self-Verification and Self-Correction," in *arXiv preprint*, 2024.

[13] F. Cui *et al.*, "OriGen: Enhancing RTL Code Generation with Code-to-Code Augmentation and Self-Reflection," in *arXiv preprint*, 2024.

[14] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "SWE-bench: Can Language Models Resolve Real-world Github Issues?" In *arXiv preprint*, 2024.

[15] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts," in *Transactions of the Association for Computational Linguistics*, 2024.

[16] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-Aware AIG Rewriting," in *Proc. DAC*, 2006.

[17] M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Optimizing Majority-Inverter Graphs with Functional Hashing," in *Proc. DATE*, 2016.

[18] W. Haaswijk, M. Soeken, L. Amarú, P.-E. Gaillardon, and G. De Micheli, "A Novel Basis for Logic Rewriting," in *Proc. ASPDAC*, 2017.

[19] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, Y. Huang, and B. Yu, "FGNN2: A Powerful Pre-Training Framework for Learning the Logic Functionality of Circuits," in *IEEE TCAD*, 2024.

[20] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "DeepGate3: Towards Scalable Circuit Representation Learning," in *Proc. ICCAD*, 2024.

[21] H. Zheng, Z. He, F. Liu, Z. Pei, and B. Yu, "LSTP: A Logic Synthesis Timing Predictor," in *Proc. ASPDAC*, 2024.

[22] C. Xu, P. Sharma, T. Wang, and L. W. Wills, "Fast, Robust and Transferable Prediction for Hardware Logic Synthesis," in *Proc. MICRO*, 2023.

[23] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie, "Gamora: Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks," in *Proc. DAC*, 2023.

[24] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph Neural Network Inference for Transferable Power Estimation," in *Proc. DAC*, 2020.

[25] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," in *Proc. FOCS*, 1981.

[26] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proc. NIPS*, 2020.

[27] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, H. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *Proc. NIPS*, 2022.

[28] J. J. Miller, "Graph Database Applications and Concepts with Neo4j," in *Proc. SAIS*, 2013.

[29] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An Evolving Query Language for Property Graphs," in *Proc. SIGMOD*, 2018.

[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning On Large Graphs," in *Proc. NIPS*, 2017.

[31] K. Sohn, "Improved Deep Metric Learning with Multi-class N-pair Loss Objective," in *Proc. NIPS*, 2016.

[32] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott, "Multi-Similarity Loss with General Pair Weighting for Deep Metric Learning," in *Proc. CVPR*, 2019.

[33] S. Robertson, H. Zaragoza, *et al.*, "The Probabilistic Relevance Framework: BM25 and Beyond," *Foundations and Trends® in Information Retrieval*, 2009.

[34] R. Nogueira, Z. Jiang, and J. Lin, "Document Ranking With a Pretrained Sequence-to-Sequence Model," in *arXiv preprint*, 2020.

[35] J. Achiam *et al.*, "GPT-4 Technical Report," *arXiv preprint*, 2023.

[36] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, *et al.*, "The Rocket Chip Generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[37] A. Amid *et al.*, "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," *IEEE Micro*, 2020.

[38] F. Farshchi, Q. Huang, and H. Yun, "Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim," in *Proc. EMC2*, 2019.

[39] H. Genc *et al.*, "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," in *Proc. DAC*, 2021.

[40] Intel Labs, *Vector Acceleration IP core for RISC-V**. [Online]. Available: https://github.com/IntelLabs/riscv-vector.

[41] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for Accelerator Design and Customized Architectures," in *Proc. IISWC*, 2014.

[42] A. Dubey *et al.*, "The Llama 3 Herd of Models," in *arXiv preprint*, 2023.

[43] V. Rawte, A. Sheth, and A. Das, "A Survey of Hallucination in Large Foundation Models," in *arXiv preprint*, 2023.

[44] Anthropic PBC, *Introducing Claude 3.5 Sonnet*, 2024. [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet.

[45] OpenCores, *OpenCores Hardware RTL Designs*. [Online]. Available: https://opencores.org.

[46] G. Tziantzioulis, T.-J. Chang, J. Balkind, J. Tu, F. Gao, and D. Wentzlaff, "OPDB: A Scalable and Modular Design Benchmark," in *IEEE TCAD*, 2021.

[47] T. Ajayi, D. Blaauw, T. Chan, C. Cheng, V. Chhabria, D. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça, *et al.*, "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain," *Proc. GOMACTECH*, 2019.

[48] T. Marena, "RISC-V: High Performance Embedded SweRV™ Core Microarchitecture, Performance and Chips Alliance," *Western Digital Corporation*, 2019.

[49] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. NIPS*, 2019.

[50] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," in *arXiv preprint*, 2019.

[51] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The Faiss Library," in *arXiv preprint*, 2024.

[52] Synopsys, Inc., *PrimePower: RTL to Signoff Power Analysis*. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html.

Hydra: Harnessing Expert Popularity for Efficient Mixture-of-Expert Inference on Chiplet System

Siqi He*†, Haozhe Zhu*†, Jiapei Zheng†, Lizhou Wu†, Bo Jiao†, Qi Liu†, Xiaoyang Zeng†, and Chixiao Chen†‡
† *State Key Laboratory of Integrated Chips and Systems (SKLICS), Fudan University, Shanghai, China*
‡ *Shaoxin Laboratory, Shaoxing, Zhejiang, China*
E-mail: zhuhz@fudan.edu.cn

***Abstract*—The rapid growth of model sizes in advanced artificial intelligence algorithms, particularly in Transformer-based large language models (LLMs), has led to significant computational overhead. Mixture-of-Expert (MoE) models offer a solution through their sparsely gating mechanism but introduce new challenges of extensive all-to-all communication and model computational inefficiencies. This paper presents Hydra, a software/hardware co-design aimed at accelerating MoE inference on chiplet-based architectures. In software, Hydra employs a popularity-aware expert mapping strategy to optimize inter-chiplet communication. In hardware, it incorporates Content Addressable Memory (CAM) to eliminate expensive explicit token (un)-permutation based on sparse matrix multiplications and a redundant-calculation-skipping softmax engine to bypass unnecessary division and exponential operations. Evaluated in 22 nm technology, Hydra achieves latency reductions of 14.2× and 3.5× and power reductions of 169.1× and 18.9× over GPU and state-of-the-art MoE accelerator, respectively, thereby offering a scalable and efficient solution for MoE model deployment.**

***Index Terms*—Mixture-of-expert model, chiplet, large language model**

I. Introduction

Transformer-based large language models (LLMs) have demonstrated exceptional performance in natural language processing tasks [1]–[3]. However, their advancements are primarily driven by the growing model sizes, which hinder efficient deployment on resource-constrained devices. Recently, Mixture-of-Expert (MoE) models have emerged as an effective solution to mitigate the computational overheads of LLMs [4]–[6]. As illustrated in Fig. 1, in an MoE model, the standard feed-forward network (FFN) layer in Transformers is replaced by an MoE FFN layer comprising multiple sparsely activated experts controlled by a gating network. This sparse gating mechanism enables further model expansion with only a slight increase in computational cost.

Despite their advantages, accommodating the entire MoE model on a single chip remains impractical due to limited on-chip memory. To overcome the cost and area limitations of monolithic chips, multi-chiplet modules (MCM) have emerged as a promising approach for building energy-efficient accelerators [7], [8]. By connecting multiple smaller chiplets through a silicon interposer or organic substrate, MCMs significantly expand system capacity [9], [10]. Additionally, their flexible interconnect supports various parallelism strategies commonly used in popular inference frameworks [11], especially model parallelism [6] and expert parallelism [12], to improve MoE inference efficiency [13]–[16].

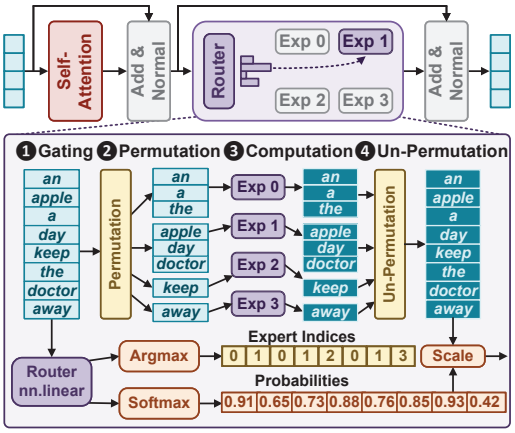


Fig. 1. Illustration of Mixture-of-Expert (MoE) mechanism in Transformer-based models.

However, deploying MoE models on chiplet-based architectures faces several challenges. First, as expert parallelism places different experts on different chiplets and executes them in parallel, extensive token dispatch and gathering operations across chiplets become necessary, resulting in extensive all-to-all communication. Previous studies have shown that this communication pattern can lead to network contention and congestion, limiting bandwidth utilization and ultimately becoming a bottleneck during inference [17]–[19]. Second, key operations in MoE layers remain computationally inefficient, including *permutation*, *un-permutation*, and *gating*. The (un)-permutation stage relies heavily on sparse matrix multiplications for token reordering, resulting in computational complexity of $O(S^2M)$, where S denotes the token length and M is the model hidden dimension. Moreover, the softmax function in the gating stage requires time-intensive exponential and division computations, incurring significant latency.

Prior approaches, including Pre-gated MoE [20] and FLAME [21], have focused on reducing the memory footprint of MoE models with an optimized expert prefetching strategy. However, these methods do not effectively address the all-to-all communication overhead or the intrinsic computational inefficiencies in MoE layers. To address these challenges, we propose Hydra, a hardware-software co-design. To our

* Siqi He and Haozhe Zhu are co-first authors. Haozhe Zhu is the corresponding author.

This work was supported by the National Natural Science Foundation of China (NSFC) under Grants 62304047 and 62322404.

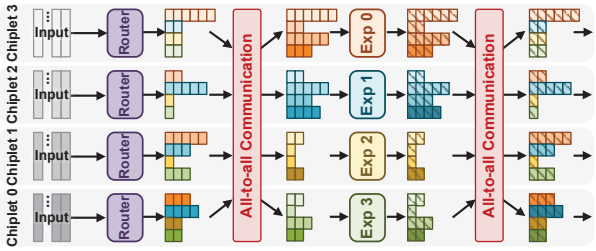


Fig. 2. MoE layer execution sequence, highlighting the all-to-all communication in an MCM, with the skewed routing decision favoring E_0 and E_1.

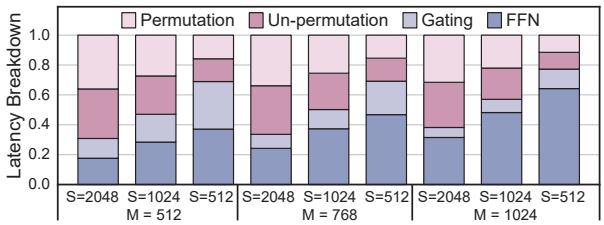


Fig. 3. Runtime analysis of four stages in an MoE layer (communication is excluded) for varying token lengths (S) and model hidden dimensions (M).

knowledge, Hydra is the first chiplet-based system specially designed for MoE deployment. The contributions of this work are as follows:

- We implement a popularity-aware dynamic expert mapping strategy derived from an analysis of conditional probabilities for expert selection across layers, significantly cutting all-to-all communication latency.
- We utilize CAM for parallel searching to establish a mapping between tokens and experts, eliminating the overhead of explicit (un)-permutation operations.
- We design a softmax engine with hybrid token/expert-wise calculation skipping, effectively removing redundant division and exponent operations in the gating stage.
- We develop the Hydra accelerator with 4×4 chiplets in 22 nm CMOS technology, achieving a 14.2× latency reduction and 169.1× power savings over the NVIDIA RTX 3090 GPU, positioning Hydra as a highly efficient solution for MoE-based LLM deployment.

II. Background and Motivation

A. MoE Execution in Chiplet-based System

As shown in Fig. 1, an MoE layer consists of four key steps: ❶ ***Gating***, where a gating network allocates input tokens to experts. Here, a linear layer first determines experts-token relevance, followed by softmax computation for probability assignment. An argmax unit then selects the target expert for each token and assigns the corresponding probability as the gate value to weight the output. Softmax computation, due to its division and exponential operations, dominates the computational cost of this stage. ❷ ***Permutation***, which groups tokens by their assigned experts, leveraging the *einsum* operator for sparse matrix multiplication $(S \times S) \times (S \times M)$, with a computational complexity of $O(S^2M)$. ❸ ***Computation***, where each expert processes their allocated tokens. ❹ ***Un-permutation***, which restores tokens to their original order, acting as the reverse of permutation stage. Together, these steps define MoE layer execution on a single device.

For multi-chiplet systems, MoE model deployment involves various parallelism strategies to improve throughput and reduce memory footprint, necessitating inter-chiplet communication. For attention layers, ***model parallelism*** partitions weight tensors across different chiplets, with all-reduce operations performed within model-parallel groups to aggregate results. For MoE layers, ***expert parallelism*** places different experts on distinct chiplets, reducing memory requirements per chiplet while enabling parallel execution. However, as each chiplet holds only a subset of experts, tokens assigned to experts on other chiplets must be dispatched to the appropriate chiplet and gathered back after step ❸ , resulting in two rounds of all-to-all communication. Fig. 2 illustrates this process, showing tokens of the same color aggregated onto their designated chiplets and subsequently retrieved.

B. Challenges and Motivations

1) All-to-all Communication: This communication pattern has been identified as a major bottleneck, contributing up to 74.9% of the total runtime of an MoE layer [17]. Although previous studies have introduced device-limited or topology-aware routing mechanisms along with token-dropping techniques to reduce communication costs and limit expert capacity, they risk degrading model performance and introducing additional training overhead [4], [22], [23].

As prior works suggest, the routing decisions in MoE models tend to be highly skewed due to the specialized knowledge domain of each expert [17], [19]. Usually, only a small subset of experts, such as E_0 and E_1 in Fig. 2, are heavily utilized. While these skewed routing decisions overload certain chiplets with the majority of tokens and lead to imbalanced data traffic that intensifies the all-to-all bottleneck, they also introduce correlation in expert choices across consecutive MoE layers, according to [18], [20]. However, this insight has not been fully explored for all-to-all communication optimization. ***By leveraging this intrinsic inter-layer correlation as a predictive basis for expert popularity on each chiplet***, we propose a dynamic expert mapping strategy to improve bandwidth utilization, as detailed in Section III.

2) Inefficient Computation: To identify computational bottlenecks in MoE inference, we analyze the runtime breakdown of the four key stages across different token lengths and model hidden dimensions. As shown in Fig. 3, excluding inter-chiplet communication, the (un)-permutation and gating stages together account for 62.7% of the runtime of an MoE layer on average, highlighting opportunities for further optimization. Although prior work has suggested using data replication for equivalent layout transformation [11], the complexity remains $O(S^2M/E)$, where E is the number of experts. Meanwhile, while optimizations for operators like argmax have been proposed [24], softmax remains unoptimized.

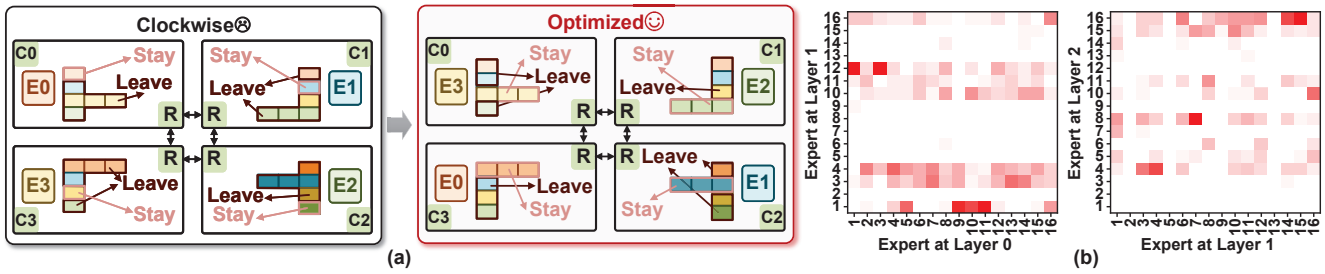


Fig. 4. (a) A 2×2-chiplet MCM example from a straightforward clockwise mapping scheme without considering expert popularity to an optimized mapping scheme that accounts for expert popularity across chiplets. (b) Heatmap of conditional probabilities for the Switch-base-16 model in the first two MoE layers.

To alleviate the computational burden, this paper introduces two key insights. First, ***a token-to-expert mapping established by parallel search*** can eliminate the need for sparse matrix multiplication. Second, the ***parallelism strategy combined with MoE's sparse gating mechanism introduces redundant computations*** in softmax operation. Based on these insights, we develop a CAM-based permutation engine for efficient parallel search and a softmax engine with hybrid token- and expert-wise calculation skipping to minimize inference latency. Further details are discussed in Section IV.

III. Software Optimization

To mitigate the all-to-all communication bottleneck in MoE inference, this section presents a predicted-popularity-aware expert mapping strategy, which is aimed at optimizing bandwidth usage and minimizing communication latency.

A. Expert Popularity Prediction

As illustrated in Fig. 4(a), for a 2×2 chiplet group with experts 0-3 arranged clockwise, each chiplet needs to send five tokens. However, with an optimized expert mapping scheme, each chiplet only sends three tokens, as more tokens remain on the original chiplet where their target experts reside. Thus, by considering the popularity of experts on each chiplet, an optimal mapping can be derived to either reduce inter-chiplet data transfers or minimize transmission hops, significantly lowering communication costs.

However, as routing decisions in MoE layers are dynamically determined at runtime, adjusting expert mapping based on the real-time gating results hinders time-overlapped weight loading, ultimately extending overall execution time. Therefore, predicting expert popularity in advance is essential for early expert mapping and prefetching. To this end, we track the routing decisions of 200,000 input tokens from the C4 dataset [25] as they traverse consecutive MoE layers and analyze the conditional probabilities of expert selection across various MoE models. Fig. 4(b) shows a heatmap of conditional probabilities for the Switch-base-16 model, revealing that expert selection across consecutive MoE layers is correlated. Therefore, expert popularity $\tilde{P}_c(E_{n,i})$ can be predicted as:

$$\tilde{P}_c(E_{n,i}) = \sum_{j=0}^{E-1} P_c(E_{n-1,j}) \times P(E_{n,i}|E_{n-1,j}) \qquad (1)$$

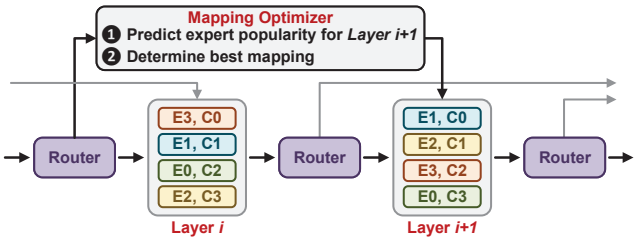


Fig. 5. Illustration of popularity-aware expert mapping.

In this equation, $P_c(E_{n-1,j})$ represents the real-time frequency of tokens assigned to expert j in layer $n-1$ for chiplet c, while $P(E_{n,i}|E_{n-1,j})$ represents the prior conditional probability of selecting expert i in layer n when expert j was selected in the previous layer $n-1$. This conditional probability-based prediction method enables the optimization of expert mapping.

B. Expert Mapping Algorithm

Building upon the expert popularity predictions presented in Section III-A, we develop an optimized expert mapping strategy, as shown in Fig. 5. Based on the prior conditional probability and the real-time routing decision in layer i, the expert mapping for layer $i+1$ can be determined. We first introduce a communication cost model tailored for multi-chiplet architectures, converting the problem into the following optimization task:

Given the expert popularity prediction $\tilde{P}_c(E_{n,i})$, the objective is to determine an optimal mapping between experts $\{E_{n,0}, E_{n,1}, ..., E_{n,E-1}\}$ and chiplets $\{C_0, C_1, ..., C_{C-1}\}$, where $n \in \{0, 1, ..., L-1\}$, $i \in \{0, 1, ..., E-1\}$ and $c \in \{0, 1, ..., C-1\}$. The mapping should minimize the following cost function:

$$\sum_{n=0}^{L-1} \sum_{c=0}^{C-1} \sum_{i=0}^{E-1} h(i,c) \times \tilde{P}_c(E_{n,i}) \qquad (2)$$

Here, L and C represent the number of MoE layers and chiplets, respectively, while $h(i,c)$ quantifies the communication cost between the chiplet hosting expert i and chiplet c. This cost is approximated as the sum of the minimum horizontal and vertical hops in the interconnect topology.

To solve the above combinatorial optimization problem, we employ a simulated annealing algorithm, as detailed in Algorithm 1. The algorithm is deployed on the mapping solver in Hydra chiplet. During inference for layer n, the external memory interface retrieves the experts for layer $n+1$

based on the pre-computed mapping scheme from the solver. This approach mitigates all-to-all communication overhead and overlaps computation with memory access, effectively reducing latency and improving overall system throughput.

Algorithm 1 Expert Mapping using Simulated Annealing

```
mapping ← AssignExpertsToChipletsRandomly()
bestMapping ← mapping
bestCost ← CalculateCost(mapping)
Temp ← InitialTemperature
while not StoppingCondition do
    newMapping ← SwapTwoExperts(mapping)
    newCost ← CalculateCost(newMapping)
    delta ← newCost - bestCost
    if delta < 0 then
        bestMapping ← newMapping
        bestCost ← newCost
    end if
    Temp ← Temp × CoolingRate
end while
return bestMapping, bestCost
```

IV. Hardware Design

In this section, we present the detailed hardware implementation of Hydra, designed to enhance computational efficiency for MoE inference.

A. Overall Architecture

Fig. 6(a) illustrates the architecture of the Hydra multi-chiplet system, composed of a 2×2 Hydra chiplet array coupled with high-bandwidth memory. Similar to Simba [8], these Hydra chiplets are organized in a 2D grid and connected via ground reference signals (GRS), supporting 100 Gb/s bandwidth in each direction. This scalable configuration allows for expansion to larger chiplet arrays, enabling the deployment of MoE models with varying sizes and diverse computational demands. Fig. 6(b) details the design of a single Hydra chiplet, which comprises 16 processing element (PE) arrays, a global SRAM buffer, an auxiliary unit, a top controller, a mapping solver, a softmax engine, a CAM-based permutation engine, and die-to-die interconnects.

B. CAM-based Token Permutation Engine

The (un)-permutation stage employs sparse matrix multiplication to rearrange data. While crucial for all-to-all communication and FFN computation, (un)-permutation incurs significant computational overhead. Therefore, we introduce a CAM-based solution to establish a mapping between the token and expert to achieve equivalent data layout transformation with reduced complexity.

We adopt a conventional 9T NAND-type CAM design [26], which is implemented by incorporating additional NAND logic outside the basic 6T SRAM cell to facilitate the search operation. When all bits in a row of the CAM array match the search key, the match result remains at a high level. As this

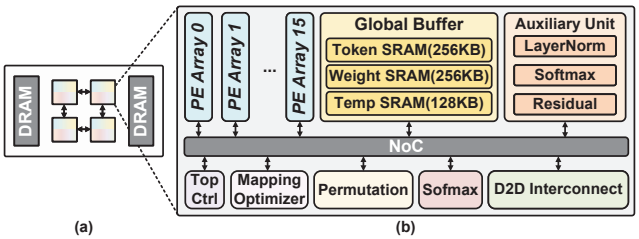


Fig. 6. (a) Overall architecture of Hydra MCM and (b) Hydra chiplet design.

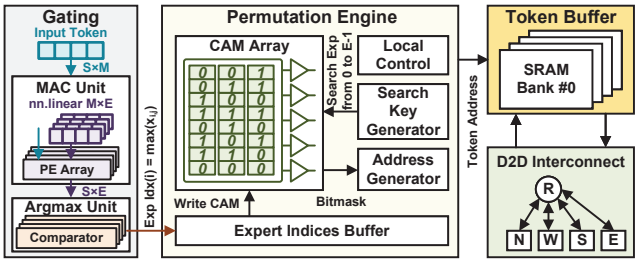


Fig. 7. CAM-based token permutation engine.

matching process is applied across all rows of the memory array, CAM enables rapid parallel searching.

The proposed CAM-based permutation engine is depicted in Fig. 7. Initially, the input tokens pass through a small linear layer. The output projection matrix is then passed to the *argmax* unit to identify the target expert, with corresponding expert indices stored in the CAM. The search key in the CAM is then sequentially set from 0 to $E-1$, with the match results forming a bitmask to indicate the positions of the relevant input tokens in memory. The address generator uses this bitmask to produce address indices for token retrieval, after which the tokens are fetched from the token buffer for participation in all-to-all communication. Following the completion of the FFN computation, the tokens are returned to the original chiplet through the interconnection network and stored back into the token buffer according to the bitmask.

The complexity of the parallel search operation in the CAM-based token permutation scheme is only $O(E)$. Given that CAM is far more energy-efficient in parallel search compared to conventional digital circuits, the overhead of this search is negligible relative to the extensive MAC operations in MoE inference. Thus, the token-to-expert mapping table is established at a minimal cost. Based on this mapping table, Hydra can complete the data layout transformation with just S data moves, achieving token permutation (or un-permutation) while eliminating the sparse matrix multiplication operations and minimizing intermediate results storage. This significantly reduces MoE inference latency and improves energy efficiency.

C. Redundant-Calculation-Skipping Softmax Engine

Fig. 8 illustrates the proposed softmax engine with hybrid token- and expert-wise calculation skipping. During probability calculation, exponential values for each element in the $S\times E$ output projection matrix of the linear layer are first computed and summed row-wise. The probability $S(x_{i,j})$ is subsequently derived through a division operation, which is implemented as a subtraction in the logarithmic domain.

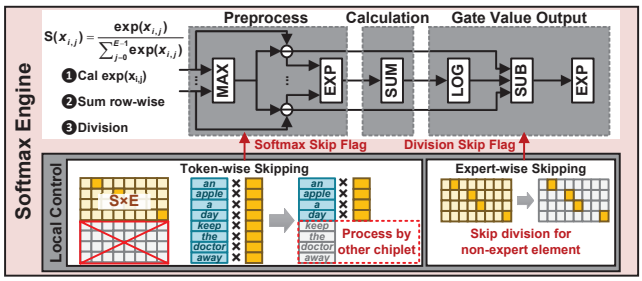


Fig. 8. Softmax engine with hybrid token-wise and expert-wise skipping.

The proposed token-wise skipping strategy targets redundant operations introduced by attention layers. Under model-parallelism, all-reduce communication occurs between chiplets that hold different weight tensor slices. As a result, input tokens across chiplets in the same model-parallel group are identical when entering the MoE layer. Thus, the local control unit instructs each chiplet to compute probabilities for only $1/M_P$ of tokens, with the remaining softmax operations assigned to other chiplet in the same group. Here, M_P is the degree of model parallelism. Expert-wise skipping strategy, on the other hand, aims at eliminating unnecessary division operations caused by the sparse gating mechanism. Once the *argmax* unit identifies the target expert, the local control unit skips the division operations for non-target experts, significantly reducing computational overhead.

The aforementioned token-wise skipping strategy is statically defined by the parallel configuration established during model deployment, while the expert-wise skipping strategy is dynamically determined by the routing decisions. As the proposed softmax skipping approach neither depends on data sparsity nor relies on approximate calculations, it offers an efficient, lossless computational optimization that eliminates redundancies arising from the inherent characteristics of the MoE algorithm and the parallelism framework.

V. Evaluation

A. Experimental Settings

Models and datasets. We evaluated the performance of Hydra using the Switch Transformer model [4], a state-of-the-art large-scale MoE model based on the encoder-decoder architecture. Specifically, we focused on switch-base-8 and switch-base-16 (with 8 and 16 experts per layer), as they are representative and highlight Hydra's scalability. The FP16 data type was employed. To verify that Hydra does not introduce any accuracy loss, we compared the negative log perplexity on a masked language modeling task against a GPU baseline using the C4 dataset.

Implementation. We evaluated a Hydra accelerator with a 4×4-chiplet configuration, using the Intel Xeon Gold 6242 and NVIDIA Geforce RTX 3090 as CPU and GPU benchmarks, respectively. The CAM array was implemented with custom circuits and simulated in Cadence Virtuoso. Power and area for inter-chip interconnects were scaled from previous 2.5D accelerator designs [8], consuming 2.4 pJ/bit. The rest of the digital modules were synthesized using Synopsys Design Compiler at 500 MHz under a 22 nm node. Table I shows the area and power breakdown. At 500 MHz, each Hydra chiplet occupies 10.49 mm^2 and consumes 934.65 mW of power.

TABLE I
Area and Power Breakdown of Hydra at 500MHz

Module	Area (mm^2)	Power (mW)
PE Arrays	5.77	643.98
Permutation Engine	0.01	0.86
Softmax Engine	0.14	19.36
Auxiliary Unit	0.26	35.43
Global Buffer	3.76	203.99
D2D Interface	0.06	4.38
Others	0.49	26.65
Total	**10.49**	**934.65**

Methodology. We used the s-tui tool [27] and NVIDIA System Management Interface to monitor the CPU and GPU, respectively. For the Hydra chip, we developed a behavioral-level simulator that was validated against the RTL implementation. At the chiplet level, we modified Noxim [28] to simulate inter-chiplet data transmission patterns, assessing both latency and power. For a thorough comparison, batch sizes of 1-4 were tested with various input-output token length combinations, where input lengths ranged from 64 to 1024 and output lengths from 16 to 256, reflecting typical user requests.

B. Performance

Inference latency and power consumption. Fig. 9(a) illustrates the normalized execution time of a single MoE layer across various input and output token lengths. The results show that Hydra achieves an average speedup of 14.2× over the GPU and 42.4× over the CPU. This significant performance improvement can be attributed to several key factors. First, Hydra optimizes expert mapping strategies, effectively reducing all-to-all communication latency. Additionally, Hydra leverages the CAM array and supports softmax skipping, minimizing the latency of the permutation operation and lowering the computational overhead of the gating stage. Fig. 9(b) presents the normalized power consumption of Hydra. The results demonstrate that Hydra achieves average power savings of 169.1× over the GPU and 357.5× over the CPU.

Multi-GPU implementation. Since multi-GPU systems also support expert parallelism, we deployed the MoE model on 4 to 8 NVIDIA A100 GPUs to compare the performance of Hydra with multi-GPU inference. Fig. 10 shows the normalized throughput and energy efficiency of Hydra under different batch sizes. The results indicate that, compared to implementations on 4 GPUs and 8 GPUs, Hydra achieves throughput improvements of 11.0× and 11.2×, respectively, along with energy efficiency improvements of 312.2× compared to 4 GPUs and 588.1× compared to 8 GPUs. The observed superiority of the 4 GPU implementation over the 8 GPU implementation may be attributed to increased communication overhead and resource contention, which can diminish the benefits of additional computational resources.

Ablation study. To evaluate the contribution of each proposed technique, we conduct ablation experiments. The work-

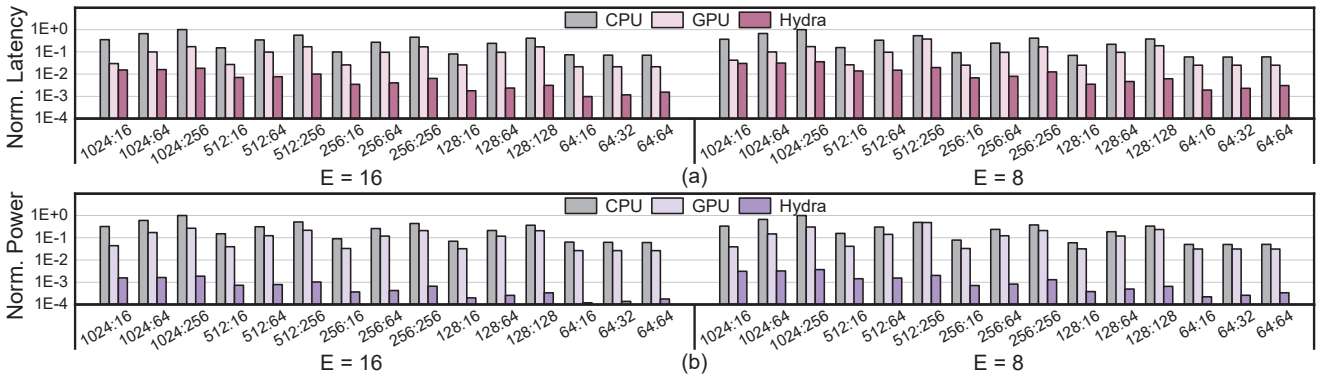


Fig. 9. (a) Inference latency and (b) power of Hydra compared to GPU and CPU. $E = 8$ refers to switch-base-8, and $E = 16$ refers to switch-base-16.

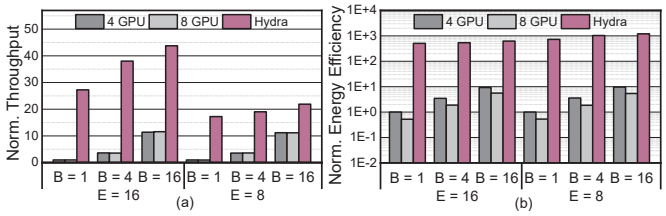


Fig. 10. Comparison with multi-GPU.

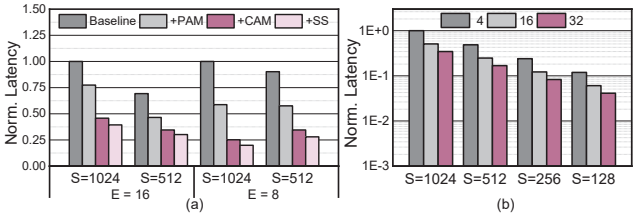


Fig. 12. (a) Ablation study. (b) Scalability evaluation.

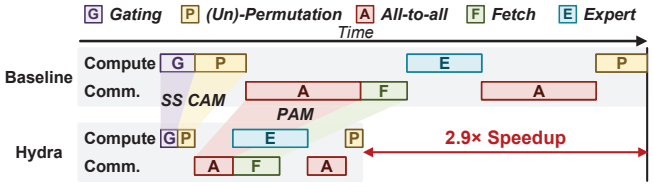


Fig. 11. Execution timeline. SS: softmax skipping; CAM: cam-based permutation; PAM: popularity-aware expert mapping.

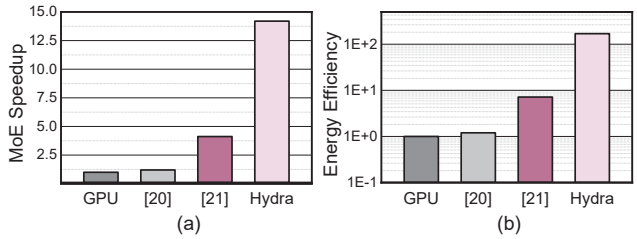


Fig. 13. (a) Inference speedup and (b) energy efficiency improvement of Hydra over other state-of-the-art works.

flow of Hydra with regard to the baseline is depicted in Fig. 11, and the results are displayed in Fig. 12(a). The optimized weight mapping strategy significantly reduces inference latency by 1.49× while introducing only a 1.2% increase in hardware overhead due to the mapping solver. Additionally, CAM-based permutation yields a 1.59× reduction by efficiently realizing token reordering, while the softmax-skipping technique further provides an additional 1.22× speedup. The above results demonstrate that each technology of Hydra has made significant contributions to inference acceleration.

Scalability. Fig. 12(b) demonstrates the performance scalability of Hydra when running the switch-base-16 model across different numbers of chiplets. When $C \leq E$, each chiplet holds E/C experts. Conversely, the MoE layer adopts a hybrid expert-model parallelism strategy, where each expert is distributed across C/E chiplets. While inter-chiplet communication costs generally limit the performance gains in chiplet-based systems, our expert mapping strategy efficiently optimizes bandwidth usage and minimizes all-to-all communication latency, thereby enabling Hydra to scale effectively.

Comparison with other accelerators. We compared Hydra to two state-of-the-art MoE acceleration schemes, Pre-gated MoE [20] and FLAME [21], both of which leverage memory offloading for single-device inference acceleration. Fig. 13 illustrates the speedup and energy efficiency results on the switch-base-8 model. Hydra achieves 14.2×, 11.8×, and 3.5× speedups over GPU-only, Pre-gated MoE, and FLAME, respectively, demonstrating the capability of a chiplet-based architecture in addressing MoE model memory requirements while alleviating latency bottlenecks from all-to-all communication through optimized expert mapping. Hydra also realizes energy efficiency improvements of 169.1×, 141.1×, and 18.9× over GPU-only, Pre-gated MoE, and FLAME, respectively.

VI. Conclusion

This work presents Hydra, a software/hardware co-design developed to address the communication and computational bottlenecks in MoE model deployment. By predicting expert popularity and dynamically adjusting expert mapping, Hydra significantly optimizes all-to-all communication. By incorporating a CAM array for (un)-permutation operations and eliminating redundant softmax calculations, Hydra further enhances computational efficiency. Hydra achieves significant reductions in latency and power, with average improvements of 14.2× and 169.1× over the NVIDIA RTX 3090 GPU, highlighting its effectiveness in accelerating MoE inference.

REFERENCES

[1] J. Achiam *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[4] W. Fedus *et al.*, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *JMLR*, vol. 23, no. 120, pp. 1–39, 2022.

[5] Y. Zhou *et al.*, "Mixture-of-experts with expert choice routing," in *NIPS*, vol. 35, 2022, pp. 7103–7114.

[6] C. Hwang *et al.*, "Tutel: Adaptive mixture-of-experts at scale," in *MLS*, vol. 5, 2023, pp. 269–287.

[7] S. Naffziger *et al.*, "2.2 amd chiplet architecture for high-performance server and desktop products," in *ISSCC*, 2020, pp. 44–45.

[8] Y. S. Shao *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *MICRO*, 2019, pp. 14–27.

[9] P. Vivet *et al.*, "Intact: A 96-core processor with six chiplets 3d-stacked on an active interposer with distributed interconnects and integrated power management," *JSSC*, vol. 56, no. 1, pp. 79–97, 2021.

[10] F. Li *et al.*, "Gia: A reusable general interposer architecture for agile chiplet integration," in *ICCAD*, 2022, pp. 1–9.

[11] R. Y. Aminabadi *et al.*, "DeepSpeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC*, 2022, pp. 1–15.

[12] D. Lepikhin *et al.*, "Gshard: Scaling giant models with conditional computation and automatic sharding," in *ICLR*, 2020.

[13] J. Cai *et al.*, "Gemini: Mapping and architecture co-exploration for large-scale dnn chiplet accelerators," in *HPCA*, 2024, pp. 156–171.

[14] Y. Feng *et al.*, "Heterogeneous die-to-die interfaces: Enabling more flexible chiplet interconnection systems," in *MICRO*, 2023, pp. 930–943.

[15] Z. Tan *et al.*, "Cocco: Hardware-mapping co-exploration towards memory capacity-communication optimization," in *ASPLOS*, 2024, pp. 69–84.

[16] Y. Feng *et al.*, "Evaluating chiplet-based Large-Scale interconnection networks via Cycle-Accurate Packet-Parallel simulation," in *USENIX ATC*, 2024, pp. 731–747.

[17] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating distributed MoE training and inference with lina," in *USENIX ATC*, 2023, pp. 945–959.

[18] J. Yao *et al.*, "Exploiting inter-layer expert affinity for accelerating mixture-of-experts model inference," in *IPDPS*, 2024, pp. 915–925.

[19] J. He *et al.*, "Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models," in *PPoPP*, 2022, pp. 120–134.

[20] R. Hwang *et al.*, "Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference," in *ISCA*, 2024, pp. 1018–1031.

[21] L. Xuanda *et al.*, "Flame: Fully leveraging moe sparsity for transformer on fpga," in *DAC*, 2024, pp. 1–6.

[22] A. Liu *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[23] C. Chen *et al.*, "Ta-moe: Topology-aware large scale mixture-of-expert training," in *NIPS*, vol. 35, 2022, pp. 22 173–22 186.

[24] X. Nie *et al.*, "Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system," *arXiv preprint arXiv:2203.14685*, 2022.

[25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[26] K. Pagiamtzis *et al.*, "Content-addressable memory (cam) circuits and architectures: a tutorial and survey," *JSSC*, vol. 41, no. 3, pp. 712–727, 2006.

[27] A. Manuskin, "The stress terminal ui: s-tui." [Online]. Available: https://github.com/amanusk/s-tui, 2019.

[28] V. Catania *et al.*, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *ASAP*, 2015, pp. 162–163.

CAE-DFKD: Bridging the Transferability Gap in Data-Free Knowledge Distillation

Zherui Zhang[1]*, Changwei Wang[3,4]*, Rongtao Xu[2], Wenhao Xu[1], Shibiao Xu[1]†, Yu Zhang[5], Jie Zhou[1] and Li Guo[1]
[1] School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing, China
[2] The State Key Laboratory of Multimodal Artificial Intelligence Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, China
[3] The Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center, Qilu University of Technology, Jinan, China.
[4] Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing, Shandong Fundamental Research Center for Computer Science, Jinan, China
[5] Tongji University, Shanghai, China

***Abstract*—Data-Free Knowledge Distillation (DFKD) enables the knowledge transfer from the given pre-trained teacher network to the target student model without access to the real training data. Existing DFKD methods focus primarily on improving image recognition performance on associated datasets, often neglecting the crucial aspect of the transferability of learned representations. In this paper, we propose Category-Aware Embedding Data-Free Knowledge Distillation (CAE-DFKD), which addresses at the embedding level the limitations of previous rely on image-level methods to improve model generalization but fail when directly applied to DFKD. The superiority and flexibility of CAE-DFKD are extensively evaluated, including: *i.)* Significant efficiency advantages resulting from altering the generator training paradigm; *ii.)* Competitive performance with existing DFKD state-of-the-art methods on image recognition tasks; *iii.)* Remarkable transferability of data-free learned representations demonstrated in downstream tasks.**

***Index Terms*—Knowledge Distillation, Representation Learning, Transfer Learning, Contrastive Learning.**

I. Introduction

KNOWLEDGE distillation (KD) transfers knowledge from high-capacity teacher networks to compact student networks, enabling comparable performance with reduced computational costs. Although effective for various computer vision tasks (e.g., image classification [1], object detection [2], [3], and semantic segmentation [4], [5]), traditional KD requires access to real training dataset (Figure 1 **(a)**), presenting privacy concerns. Data-free knowledge distillation (DFKD) [6] addresses this by reconstructing training data from the given pre-trained teacher network (Figure 1 **(b)**), are broadly categorized into optimization-based [7], [8] and generator-based views. This paper focuses on generator-based DFKD [9]–[12], where a trainable generator network synthesizes proxy dataset guided by the pre-trained teacher that as the non-updatable discriminator.

Despite advances in generator-based DFKD for image recognition [13], [14], a key research gap remains to understand and improve the transferability of learned representations to new downstream tasks and datasets. Furthermore, existing DFKD explorations also suffer from two key limitations: ***i)*** the teacher category prediction preferences introduce distributional shifts in the synthetic dataset, leading to inconsistent synthetic image quality across categories [15]–[17]; ***ii)*** the generator network training process can suffer from pattern collapse, leading to numerous noisy or worthless synthetic images. As shown in Figure 2 **(a)**, the synthetic images on CIFAR-100 exhibits significant difference across categories in low-confidence proportion from the teacher view ($\leq$ 0.1 highest probability), indicating inconsistent synthetic image quality and potential category imbalance. Figure 2 **(b)** visualizes the lack of clear semantic content in low-confidence synthetic images. Consequently, existing methods for improving model generalization or task adaptability, such as high-intensity data augmentation or auxiliary supervised objectives, directly applying methods such as Mixup or contrastive learning to synthetic images, can negatively impact student network performance (Table I), as transformations applied to semantically ambiguous synthetic images further degrade their usefulness (Figure 2 **(c)**).

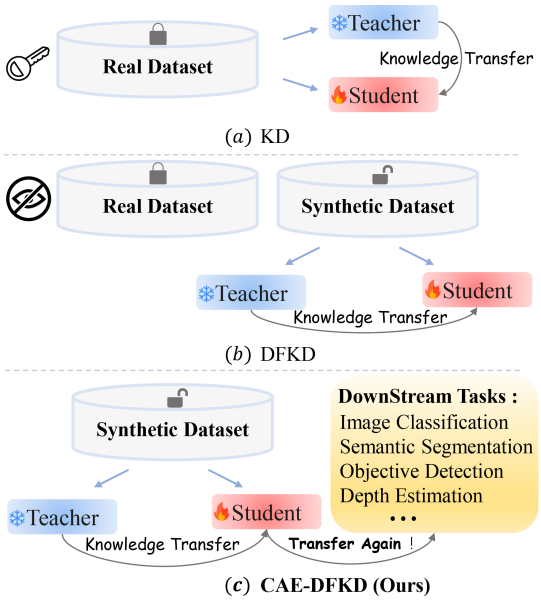


Fig. 1. **From KD to CAE-DFKD.** (a) The traditional paradigm of knowledge distillation (KD) has access to the real dataset. (b) Data-Free Knowledge Distillation (DFKD) is unaware of the read dataset and employs synthetic datasets for knowledge transfer. (c) The goal of CAE-DFKD is to continually transfer the knowledge acquired under data-free setting to downstream tasks.

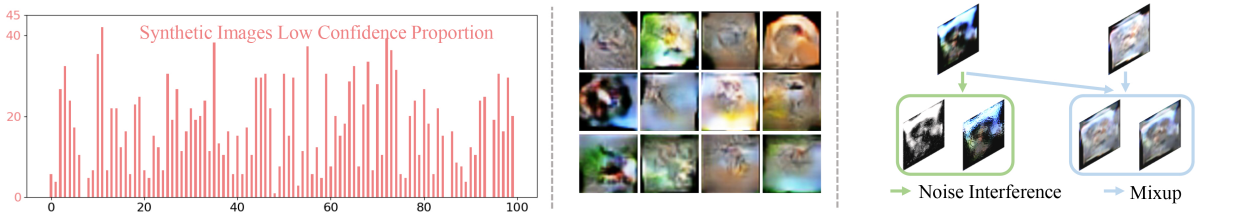
TABLE I
DIRECTLY APPLYING CONTRASTIVE LEARNING OR MIXUP, WHICH ARE DESIGNED FOR IMAGE-LEVEL, TO DFKD SYNTHETIC IMAGES YIELDS ADVERSE EFFECTS. SETTING: CIFAR-100, RESNET-34 → RESNET-18.

Method	**Top-1 Acc.(%)**
Vanilla	77.54
+ Mixup	77.13↓
+ Contrastive Learning	76.38↓

This paper addresses these limitations by shifting from image-level operation to embedding-level operation for DFKD, promoting more robust and transferable feature learning. By operating in the embedding space, our **Category-Aware Embedding Data-Free Knowledge Distillation (CAE-DFKD)** method mitigates the adverse effects of quality inconsistencies in synthetic images, and contributions are as follows:

- **Category Embedding Noise Diffusion (CEND) Layer**: Unlike

† Shibiao Xu is the corresponding author: shibiaoxu@bupt.edu.cn
* Zherui Zhang and Changwei Wang contributed equally.
This work was supported by the Beijing Natural Science Foundation (No.JQ23014), National Natural Science Foundation of China (No.62271074).



(***a***) CIFAR-100 Synthetic Dataset Quality Inconsistency (***b***) Low-Quality Synthetic Images (***c***) Negative Effects of Image-level Augmentation

Fig. 2. **Quality Difference in Synthetic Images.** (a) The proportion of low-confidence (≤ 0.1 highest probability) synthetic images within the corresponding categories. (b) The generator inevitably produces semantically ambiguous, low-quality synthetic images. (c) Applying image-level noise interference or high-intensity data augmentation to uncertain images renders their semantics more abstract and difficult to exploit.

native DFKD setting that sample generator embedding inputs from unstructured Gaussian noise, we introduce a structured embedding space with category priors using pre-trained language models. The proposed CEND layer applies multi-source noise to dynamically diffuse category embeddings, enriching the structured embedding space, and contributes to the fast convergence of the generator.

- **Category Noise Contrastive Learning (CNCL)**: CNCL is an embedding-level contrastive learning method designed for DFKD that exploits the generator network to *indirectly* produce positive-negative contrastive pairs, encouraging the student network to focus on domain-invariant category features with stronger generalization.
- **Effective Data-Free Transferability**: CAE-DFKD integrates the advantages of the CEND and CNCL components, enabling effective data-free knowledge transfer to various downstream tasks (Figure 1 (c)), surpassing the adaptability and practicality of existing DFKD methods.

II. Related Work

1) ***Data-free KD (DFKD):*** DFKD transfers knowledge from the given pre-trained teacher to the target student network using synthetic images, avoiding access to the real training dataset. Generator-based DFKD methods [13]–[15], [18], the focus of this paper, adversarially train a generator network and a student network under the guidance of the given teacher network that as the un-trainable discriminator. The generator synthesizes images to maximize the divergence between student and teacher predictions, while the student learns to align its predictions with the teacher in these synthetic images [9], [11], [18].

However, generator-based DFKD faces a key challenge: synthetic dataset quality. These generated images often do not fully capture the diversity and complexity of real data, and their quality is heavily influenced by the teacher network prediction preferences [15], [16]. Existing DFKD methods attempt to address this limitation through various strategies. Some optimize generator features to promote diversity (e.g. SpaceShipNet [14]), while others incorporate causal inference (KDCI [15]), leverage open-world data (ODSD [16]), implement curriculum learning (AdaDFKD [19]), or employ diffusion-based augmentation (DDA [17]). NAYER [13] periodically re-initializes the generator for improved diversity.

Although these methods improve image recognition performance on the corresponding real dataset by improving synthetic image quality, their effectiveness on broader downstream tasks remains largely unexplored. In other words, it remains unknown whether the representations learned following the data-free setting possess transferability. This paper fills this gap by introducing an embedding-level paradigm to DFKD, enabling data-free and privacy-preserving knowledge transfer for a wider range of real-world applications.

2) ***Contrastive Learning:*** Self-supervised learning exploits pretext tasks on unlabeled data to obtain robust and discriminative representations [20]–[23]. Among these pretext tasks, contrastive learning [24]–[27] outperforms many downstream tasks by bringing semantically similar data closer to the latent space while pushing apart differing data. However, the effectiveness of contrastive learning is highly dependent on how positive and negative pairs are defined [24]. Conventional methods generally produce these contrastive pairs at the image level [20], [22], but under the DFKD setting, this strategy risks introducing greater noise or more marked semantic ambiguity, thus increasing uncertainty in the optimization objective of the student network, as discussed in Section I.

In this paper, we exploit the synthesizing ability of the generator-based DFKD to address this issue. Instead of constructing positive-negative contrastive pairs at the image level, we introduce embedding-level design that avoids the potential degradation resulting from low-quality synthetic images, allowing the student model to focus on more generalizable domain-invariant features.

III. Method

A. Problem Formulation and Overview

1) ***KD $\rightarrow$ DFKD:*** Given a training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{\mathcal{I}}$, where $\mathcal{I}$ represents the length of $\mathcal{D}$, and each image $x_i \in \mathbb{R}^{3\times h\times w}$, where $h \times w$ are the input resolution, $y_i \in \{1, 2, \cdots, K\}$ denotes its corresponding hard label. Knowledge distillation (KD) aims to transfer knowledge from the given pre-trained teacher network $\mathcal{T}(x;\theta_T)$ to the smaller target student network $\mathcal{S}(x;\theta_S)$. The objective is for the student to approximate the teacher network output logits on $\mathcal{D}$ through the following optimization:

$$\min_{\theta_s} \mathbb{E}_{(x_i,y_i)\in\mathcal{D}} \Big[\mathcal{L}_{\text{KL}}\left(\mathcal{S}(x_i;\theta_S), \mathcal{T}(x_i;\theta_T)\right) + \lambda\mathcal{L}_{\text{CE}}\left(\mathcal{S}(x_i;\theta_S), y_i\right)\Big], \tag{1}$$

where λ is used as a hyperparameter to balance the Kullback-Leibler divergence ($\mathcal{L}_{KL}$) and cross-entropy loss ($\mathcal{L}_{CE}$).

By comparison, data-free knowledge distillation (DFKD) seeks to train the student network without access to the real training dataset $\mathcal{D}$. Generator-based DFKD achieves this by employing adversarial training, in which a generator G produces synthetic images used to train the student:

$$\min_{\theta_S}\max_{\theta_G} \mathbb{E}_{z\sim\mathcal{N}(0,1)}\left[\mathcal{L}_{KL}\left(\mathcal{S}(\hat{x};\theta_S), \mathcal{T}(\hat{x};\theta_T)\right)\right], \tag{2}$$

where $\hat{x} = G(z, y;\theta_G)$ is a synthetic image produced by the generator G, and z is embedding vector sampled from the Gaussian distribution.

2) ***Motivation:*** The goal of existing DFKD methods is to improve image recognition performance on the corresponding real dataset, yet there is insufficient exploration on whether the performance gains obtained through data-free paradigm can benefit other downstream tasks.

3) ***CAE-DFKD Framework:*** Figure 3 demonstrates how our proposed CAE-DFKD flexibly incorporates into the existing DFKD framework. During the generator network update, we replace the native unstructured Gaussian noise sampling with the initial category

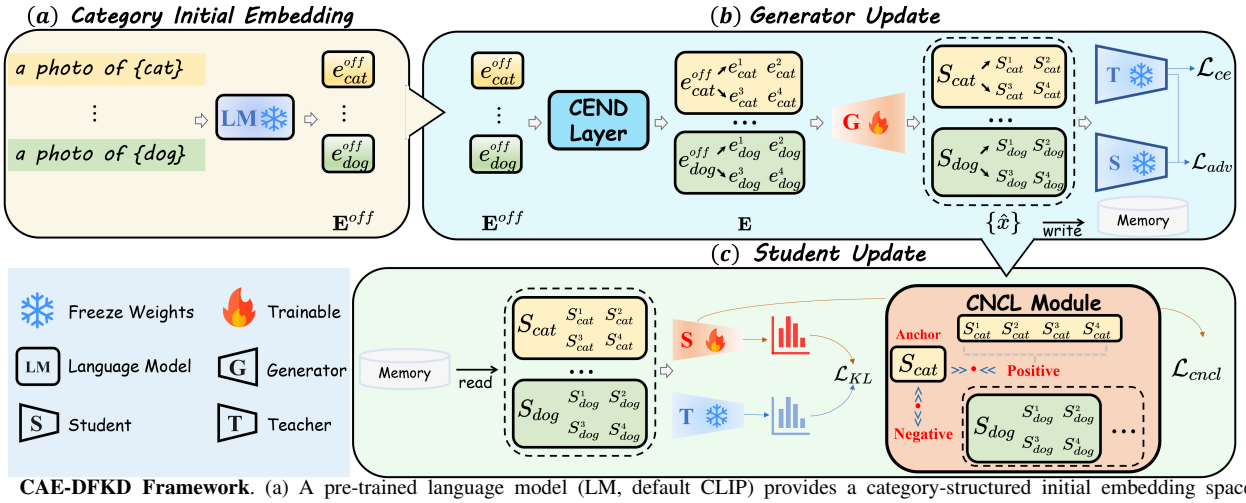


Fig. 3. **CAE-DFKD Framework**. (a) A pre-trained language model (LM, default CLIP) provides a category-structured initial embedding space, $\mathbf{E}^{\text{off}}$, contrasting with the un-structured Gaussian noise used in native DFKD, this process is performed offline and does not cause any training burden; (b) During generator updates, the Category Embedding Noise Diffusion (CEND) layer addresses $\mathbf{E}^{\text{off}}$ sparsity and lack of diversity. CEND samples from noise sources with distinct pre-set distributions, dynamically diffusing category embeddings to induce a diverse embedding space, $\mathbf{E}$. synthetic images, written to memory, are fed to teacher and student networks to compute cross-entropy ($\mathcal{L}_{ce}$), adversarial ($\mathcal{L}_{adv}$), and batch normalization ($\mathcal{L}_{bn}$) losses for generator updates; (c) During student updates, classic logit knowledge distillation ($\mathcal{L}_{kl}$) is performed on synthetic images read from memory. Further, Category Noise Contrastive Learning (CNCL) constructs embedding-level positive-negative pairs, optimizing towards $\mathcal{L}_{cncl}$.

embedding $\mathbf{E}^{\text{off}}$, and employ the Category Embedding Noise Diffusion (CEND) layer to construct a diffusible multi-source, category-structured embedding space $\mathbf{E}$. The synthetic images are then generated and stored in memory, with the generator network optimization details given in Section III-D. During the student network update, synthetic data from memory is fed into both the teacher and student networks. In addition to traditional logit knowledge alignment, the Category Noise Contrastive Learning (CNCL) module encourages the student network to focus on robust domain-invariant category features beneficial to strong generalization.

B. Initial Category Embedding and Noise Diffusion

Unlike native DFKD, which samples unstructured Gaussian noise as the generator embedding inputs, we employ a pre-trained language model (LM) to introduce structured initial category embeddings. Specifically, as illustrated in Figure 3 **(a)**, we construct prompts of the form "*a photo of* {*class*}" as input to the LM, where "{*class*}" can be the class name or class index (further explored in Section V-5). For example, for the class "cat", we input *"a photo of cat"* into the LM, obtaining the corresponding initial category embedding $\mathbf{e}^{\text{off}}_{\text{cat}} \in \mathbb{R}^D$ where D is the embedding dimension. This process is repeated for all K categories, forming the initial category embedding space $\mathbf{E}^{\text{off}} = \{\mathbf{e}^{\text{off}}_k\}_{k=1}^K$, where $\mathbf{E}^{\text{off}} \in \mathbb{R}^{K\times D}$ and "off" indicate that this operation takes place offline and is only performed once throughout the entire DFKD process, without introducing any additional training overhead.

Although $\mathbf{E}^{\text{off}}$ exhibits the initial category structure compared to Gaussian noise, it remains sparse and lacks the embedding space richness. To address this, we introduce the **Category Embedding Noise Diffusion (CEND)** layer, detailed in Figure 3 **(b)** or Figure 4. CEND takes $\mathbf{E}^{\text{off}}$ as input and samples from N noise sources (illustrated with $N = 4$). Each noise source, denoted as $\mathbf{NS}_n$ for $n \in \{1, 2, \ldots, N\}$, follows a distinct pre-defined noise distribution, generating the corresponding noise vectors $\mathbf{q}_n \in \mathbb{R}^D$, which guides $\mathbf{E}^{\text{off}}$ to diffuse into diverse latent embedding spaces. For the "cat" category, we sample noise vectors $\mathbf{q}_n$ from each $\mathbf{NS}_n$ and perform element-wise calculation to diffuse the initial category embedding $\mathbf{e}^{\text{off}}_{\text{cat}}$:

$$\mathbf{e}^{n}_{\text{cat}} = \mathbf{e}^{\text{off}}_{\text{cat}} \oplus (\mathbf{M}_n \odot \mathbf{q}_n), \quad n \in \{1, 2, \ldots, N\} \tag{3}$$

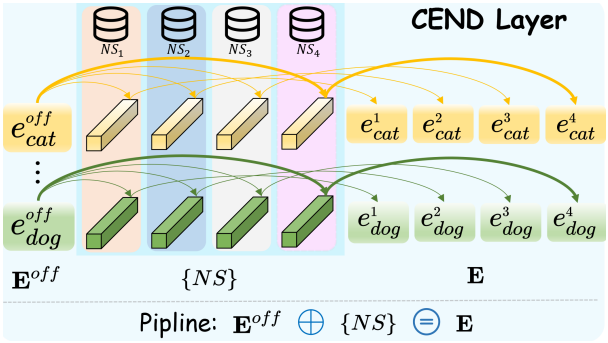


Fig. 4. **Category Embedding Noise Diffusion (CEND).** We leverage a pre-trained language model to initialize category embeddings ($\mathbf{E}^{\text{off}}$) with structure, in contrast to the unstructured Gaussian noise used in native DFKD. However, $\mathbf{E}^{\text{off}}$ suffers from sparsity and lacks diversity. Our proposed CEND layer introduces N noise sources ($\{NS\}$), each following a distinct pre-defined distribution (illustrated here with N=4), sampling from each $NS_n (n = 1, \cdots 4)$ and element-adding it to $\mathbf{E}^{\text{off}}$ enables dynamic diffusion of the initial embeddings, resulting in a richer embedding space $\mathbf{E}$.

where $\mathbf{M}_n$ is the magnitude of noise perturbation from source $\mathbf{NS}_n$, $\oplus$ denotes the element-wise addition and $\odot$ represents element-wise multiplication. The same process is applied for all K categories, producing diffused embedding space $\mathbf{E}$, as shown in Figure 4, which serves to enrich the category-structured embedding space.

C. Contrastive Pairs Construct: Image-Level → Embedding-Level

Traditional contrastive learning constructs positive-negative pairs at the image level, however, as analyzed in Section I (both quantitatively and qualitatively), this native design provides limited benefit to the student network in DFKD. We propose an embedding-level method that exploits the synthesizing ability of the generator network to indirectly construct positive-negative contrastive pairs, thereby avoiding direct comparisons among often lower-quality synthetic images.

We introduce the **Category Noise Contrastive Learning (CNCL)** module, which uses the initial category embeddings $\mathbf{E}^{\text{off}}$ and the diffused noise embeddings as generator input to produce anchor synthetic images and corresponding positive contrastive pairs. For example, as shown in Figure 3 **(b)**, the initial embeddings $\mathbf{e}^{\text{off}}_{\text{cat}}$ and $\mathbf{e}^{\text{off}}_{\text{dog}}$ generate anchor images $S_{\text{cat}} = G(\mathbf{e}^{\text{off}}_{\text{cat}})$ and $S_{\text{dog}} = G(\mathbf{e}^{\text{off}}_{\text{dog}})$,

TABLE II

SMALLER RESOLUTION EXPERIMENTS. COMPARISON OF OUR **CAE-DFKD** WITH OTHER DATA-FREE METHODS UNDER DIFFERENT CONFIGURATIONS. THE BEST RESULTS, SIGNIFIED IN BOLD, AND THE SYMBOLS [*] AND [$^\lambda$] ARE USED HERE TO INDICATE THAT THE RESULTS ARE FROM [14] AND NAYER [13], RESPECTIVELY.

Methods	Data Available	CIFAR-100					CIFAR-10				
		ResNet-34 ResNet-18	VGG-11 ResNet-18	WRN-40-2 WRN-16-1	WRN-40-2 WRN-40-1	WRN-40-2 WRN-16-2	ResNet-34 ResNet-18	VGG-11 ResNet-18	WRN-40-2 WRN-16-1	WRN-40-2 WRN-40-1	WRN-40-2 WRN-16-2
Teacher	✔	78.05	71.32	75.83	75.83	75.83	95.70	92.25	94.87	94.87	94.87
Student	✔	77.10	77.10	65.31	72.19	73.56	95.20	95.20	91.12	93.94	93.95
DAFL* [28]$_{ICCV'19}$	✗	74.47	57.29	22.50	34.66	40.00	92.22	81.10	65.71	81.33	81.55
ZSKT$^\lambda$ [29]$_{NIPS'19}$	✗	67.74	54.31	36.60	53.60	54.59	93.32	89.46	83.74	86.07	89.66
DeepInv$^\lambda$ [30]$_{CVPR'20}$	✗	61.32	54.13	53.77	68.58	61.34	93.26	90.36	83.04	86.85	89.72
DFQ$^\lambda$ [31]$_{CVPR'20}$	✗	77.01	66.21	51.27	54.43	64.79	94.61	90.84	86.14	91.69	92.01
CMI$^\lambda$ [32]$_{IJCAI'21}$	✗	77.04	70.56	57.91	68.88	68.75	94.84	91.13	90.11	92.78	92.52
FM$^\lambda$ [18]$_{AAAI'22}$	✗	74.34	67.44	54.02	63.91	65.12	94.05	90.53	89.29	92.51	92.45
SpaceShipNet* [14]$_{CVPR'23}$	✗	77.41	71.41	58.06	68.78	69.95	**95.39**	92.27	90.38	93.56	93.25
SSD-KD [33]$_{CVPR'24}$	✗	75.16	68.77	55.61	64.57	65.28	94.26	90.67	89.96	93.23	93.11
KDCI + CMI [15]$_{CVPR'24}$	✗	75.07	69.07	57.19	67.47	67.68	94.43	91.28	89.52	92.84	92.73
CCL-D [34]$_{TMM'24}$	✗	77.12	71.02	61.04	70.21	70.59	95.36	91.85	91.76	94.27	94.05
NAYER [13]$_{CVPR'24}$	✗	77.54	71.75	62.23	71.80	71.72	95.21	92.37	91.94	94.15	94.07
CAE-DFKD (Ours)	✗	**77.83**	**72.36**	**62.73**	**72.23**	**72.09**	95.09	**92.74**	**92.21**	**94.89**	**94.36**

respectively. The diffused embeddings $\mathbf{e}^n_{\text{cat}}$ generate $S^n_{\text{cat}} = G(\mathbf{e}^n_{\text{cat}})$, which serve as positive pairs for S_{cat}. As shown in Figure 3 **(c)**, anchor images of other categories (e.g., S_{dog}) and its corresponding noise-perturbed versions ($S^n_{\text{dog}} = G(\mathbf{e}^n_{\text{dog}})$) constitute negative pairs for S_{cat}. This leads to the CNCL loss:

$$\mathcal{L}_{cncl} = -\frac{1}{K}\sum_{k=1}^{K}\sum_{S_k^+ \in \mathcal{P}_k} \log \frac{\exp(\text{sim}(S_k, S_k^+)/\tau)}{\sum_{S' \in \mathcal{P}_k \cup \mathcal{N}_k} \exp(\text{sim}(S_k, S')/\tau)} \tag{4}$$

where K is the number of categories, τ is the temperature parameter, S_k is the anchor image for category k, $\mathcal{P}_k = \{S^n_k | n = 1, \ldots, N\}$ is the set of positive samples generated from the diffused embeddings of category k, $\mathcal{N}_k = \{S^n_{k'} | k' \neq k, n = 1, \ldots, N\}$ is the set of negative samples from other categories, and $\text{sim}(\cdot, \cdot)$ denotes the cosine similarity function.

D. Optimization

1) ***Generator Optimization****:* The generator network is trained using the following loss function (Figure 3 **(b)**):

$$\mathcal{L}_G = \mathcal{L}_{CE} + \lambda_{bn}\mathcal{L}_{BN} + \lambda_{adv}\mathcal{L}_{adv} \tag{5}$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss between the teacher network's predictions on synthetic images and its corresponding ground-truth labels, $\mathcal{L}_{BN}$ is a batch normalization loss for numerical stability (commonly used in DFKD), and $\mathcal{L}_{adv}$ (Eq. 2) is the adversarial loss. The hyperparameters λ_{bn} and λ_{adv} balance these losses.

2) ***Student Optimization****:* The student network is trained using a combination of the classical knowledge distillation objective and our proposed contrastive learning method (Figure 3 **(c)**). The loss function is:

$$\mathcal{L}_S = \mathcal{L}_{KL} + \alpha\mathcal{L}_{cncl} \tag{6}$$

where $\mathcal{L}_{KL}$ is the KL divergence between the teacher and student logits (for knowledge distillation), $\mathcal{L}_{cncl}$ is the embedding-level category noise contrastive learning loss (introduced in Section III-C), and α controls the contribution of the contrastive loss.

IV. EXPERIMENTS

A. Experimental Setup

Datasets: We comprehensively evaluate the performance of the proposed CAE-DFKD method across image recognition datasets with various input resolutions. For smaller resolutions of 32×32, we evaluate CAE-DFKD on CIFAR-10 and CIFAR-100, which contain 10 and 100 categories, respectively. For medium resolutions of 64×64, we employ Tiny-ImageNet, which comprises 200 categories. For larger resolutions of 224×224, we conduct evaluations on ImageNet-1K.

TABLE III

MEDIUM RESOLUTION EXPERIMENTS. COMPARISON OF RESNET-34 AS TEACHER AND RESNET-18 AS STUDENT AGAINST SOTA METHODS. SYMBOL [$^\lambda$] INDICATES THAT THE RESULT IS FROM [13].

Methods	Data Available	Tiny-ImageNet Top-1 Acc. (%)
Teacher	✔	66.44
Student	✔	64.87
CMI$^\lambda$ [32]$_{IJCAI'21}$	✗	64.01
PREKD$^\lambda$ [37]$_{AAAI'22}$	✗	49.94
MBDFKD$^\lambda$ [38]$_{ICCV'22}$	✗	47.96
MAD$^\lambda$ [39]$_{NIPS'22}$	✗	62.32
KAKR$_{MB}{}^\lambda$ [10]$_{CVPR'23}$	✗	47.96
KAKR$_{GR}{}^\lambda$ [10]$_{CVPR'23}$	✗	49.88
SpaceShipNet [14]$_{CVPR'23}$	✗	64.04
KDCI + DFND [15]$_{CVPR'24}$	✗	49.54
NAYER [13]$_{CVPR'24}$	✗	64.17
CAE-DFKD (Ours)	✗	**64.72**

Configuration Details: Evaluation procedure is performed across various teacher-student combinations, including ResNet [35], WideResNet (WRN), and VGG [36] models, without access to the real dataset. The generator network is optimized using the objective function defined in Eq. 5, with the Adam optimizer and an initial learning rate of 0.001. For the student network, we employ the objective function defined in Eq. 6 and optimize it using the SGD optimizer with an initial learning rate of 0.1 and training epochs of 300. Additionally, cosine annealing scheduling is applied to adjust the learning rate throughout the training process. All experiments are performed on the NVIDIA RTX 3090 GPUs.

B. Main Results

1) ***Comparison with State-of-the-art Methods****:* Tables II, III, and IV present the experimental results[1] conducted at progressively increasing resolutions: smaller (32×32), medium (64×64), and larger (224×224), respectively. "Teacher" and "Student" denote the results obtained using the data-accessible setting. In Table II, focusing on the smaller resolution corresponding

[1]Top three methods in descending order: 1_{st}, 2_{nd}, 3_{rd}

TABLE IV
LARGER RESOLUTION EXPERIMENTS. RESNET-50→RESNET-50. SYMBOL [$^\lambda$] INDICATES THAT THE RESULT IS FROM [13].

Methods	Data Available	ImageNet-1K Top-1 Acc. (%)
Teacher	✓	75.45
Student	✓	75.45
FM$^\lambda$ [18]$_{AAAI'22}$	✗	57.37
DeepInv$^\lambda$ [30]$_{CVPR'20}$	✗	68.00
NAYER [13]$_{CVPR'24}$	✗	68.92
CAE-DFKD (Ours)	✗	**69.33**

to CIFAR-10 and CIFAR-100, CAE-DFKD consistently outperforms other methods, except for the ResNet-34 → ResNet-18 combination on CIFAR-10. At medium resolution for Tiny-ImageNet, most current DFKD methods [13], [14], [37] select the ResNet-34 → ResNet-18 setting for testing, as shown in Table III, where our CAE-DFKD surpasses SOTA NAYER [13] (64.17% → 64.72%). In Table IV at the larger resolution for ImageNet-1K, due to efficiency constraints, existing DFKD methods rarely report results. We follow the classic ResNet-50 → ResNet-50 setting [13], [18], [30], using PyTorch-provided pre-trained weights for the teacher ResNet-50.

*2) **Generalization Performance Comparison**:* We conduct extensive experiments to validate the transferability of learned representations by student networks. Specifically, we perform the evaluation under both data-accessible and data-free settings across a range of downstream tasks, including semantic segmentation, depth estimation, surface normal prediction, and object detection. The evaluation metrics for each task are detailed as follows:

- NYUv2: A benchmark for scene understanding, NYUv2 employs distinct metrics for each subtask. Semantic segmentation evaluation via Mean Intersection over Union (mIoU) and Pixel Accuracy (pAcc). Depth estimation evaluation using absolute and relative error. For surface normal estimation, we consider the mean absolute error (Mean), median absolute error (Median), and percentages of pixels with angular errors within 11.25, 22.5, and 30 degrees.
- ADE-20K: Serving as a benchmark [40] to evaluate scene parsing capabilities, relies on semantic segmentation and employs pAcc and mIoU, consistent with NYUv2 evaluation. Training and evaluation are conducted using the open-source framework.
- COCO-2017: A large-scale image recognition dataset, COCO-2017 [41] includes various subtasks. We focus on evaluating transfer capabilities within the object detection subtask. Metrics include the mean Average Precision (mAP) at different IoU thresholds (50, 75) and detection behavior across object sizes, categorized as small (s), medium (m), and large (l).

In addition, CAE-DFKD and the other compared methods perform complete data-free knowledge distillation on CIFAR-100, and the obtained Student weights are then **fine-tuned** for downstream tasks.

Tables V and VI present the evaluation results for downstream tasks, comparing the proposed CAE-DFKD with state-of-the-art (SOTA) DFKD methods like CMI [32], SpaceShipNet [14], and NAYER [13], "Student" denotes data-accessible settings. Although SOTA DFKD methods might achieve comparable or marginally superior accuracy against the data-accessible setting on the real image recognition dataset, this advantage does not consistently extend to broader tasks, falling short of the data-accessible "Student". Notably, our method, in addition to maintaining comparable performance on the real image recognition dataset, prioritizes learning generalizable and transferable category representations. CAE-DFKD not only exceeds existing SOTA DFKD methods, but also outperforms data-accessible "Student" settings in certain scenarios, including absolute error (AErr) in depth estimation, the 11.25° setting in surface normal prediction, and COCO-2017 with small and medium object (mAP$^{bbox}_s$, mAP$^{bbox}_m$).

V. ABLATION EXPERIMENTS

*1) **Ablation Study of CEND and CNCL**:* Table VII uses CMI [32] as the base method to validate the effectiveness of our proposed CEND layer and CNCL module under the data-free setting. It is observed that the category-structured embeddings introduced by CEND, compared to the un-structured Gaussian noise embeddings commonly used in DFKD, introduce embedding-level richness and alleviate the convergence burden of the generator, which helps produce high-quality synthetic images. Additionally, CNCL constructs positive-negative contrastive pairs at the embedding level to mitigate the issue of significant difference in synthetic image quality, and encourages the student network to focus on category representations with strong domain generalization, thereby benefiting downstream tasks, as shown in Figure 5.

*2) **Effect of Noise Sources Number (N) on Generalization**:* In CEND, we introduce the noise diffusion method into the initial category embeddings to enrich the embedding space, which is influenced by the noise diffusion source number N. In addition, N also affects the construction of positive-negative contrastive pairs in CNCL. In Table VIII, we present the impact of different N on the performance of the student network in downstream tasks. When N is relatively small (< 4), CEND struggles to introduce sufficient richness to the embedding space $\mathbf{E}$. In contrast, when N becomes too large (> 4), an excessive number of positive and negative pairs is generated for CNCL, which decreases training efficiency and increases the difficulty of semantic contrast for noisy images, thus placing a heavier burden on the optimization process of the student network. Despite this, our CAE-DFKD significantly outperforms the base method under different values of N, with the most robust performance achieved when $N = 4$.

*3) **Efficiency Advantage of CEND**:* Under native DFKD generator network update setting, which samples Gaussian noise and is required to project into highly structured dataset structures ("**un-structured → structured**"), faces extreme difficulties, particularly in the data-free setting where no real images are available to provide discriminative information. In contrast, our CAE-DFKD approach introduces rich structured embeddings through CEND, transforming the generator update process into "**structured → structured**", significantly alleviating the convergence burden on the generator. Table IX reports the efficiency gains made by CEND.

*4) **LMs on Generalization**:* In Section III-B, we employ a pre-trained LM to provide structured category information for the generator embedding inputs. In Table X, we report the impact of different LMs on the downstream task of COCO-2017 object detection. It can be observed that, compared to the base, the specific choice of LM (doc2vec, CLIP, SBERT) shows little sensitivity, as all lead to significant performance improvements. However, CLIP yields superior performance.

*5) **Limitation**:* Data-free settings involve the absence of the real dataset. In scenarios demanding heightened privacy, access to class names might be restricted, or pre-trained language models may lack corresponding category priors. Unlike the discussion in Section III-B, which exploits *"a photo of {class name}"* as the default prompt, we explore a more general prompt design: *"a photo of {class index}"*. The experimental results in Table XI demonstrate the effectiveness of both prompt designs, with the more general

TABLE V
ON THE **NYUV2** DATASET, TRANSFERABILITY OF LEARNED KNOWLEDGE WITH OUR CAE-DFKD IS EVALUATED.

Method	Data Available	Semantic Segmentation		Depth Estimation		Surface Normal Prediction				
						Angle Distance		Within t°		
		mIoU ↑	pAcc (%) ↑	AErr ↓	RErr ↓	Mean ↓	MED ↓	11.25 ↑	22.5 ↑	30 ↑
Teacher	✔	37.61	64.03	0.5137	0.2194	27.82	20.92	28.99	52.78	64.13
Student	✔	34.09	61.28	0.5349	0.2219	29.68	24.16	26.77	50.14	59.94
NAYER [13]	✗	31.82	59.12	0.5678	0.2535	32.18	27.35	26.14	46.28	57.68
CAE-DFKD (Ours)	✗	**33.77**	**60.86**	**0.5211**	**0.2374**	**30.06**	**24.99**	**27.36**	**48.33**	**58.16**

TABLE VI
VALIDATE THE EFFECTIVENESS OF CAE-DFKD IN TRANSFER LEARNING ON THE **ADE-20K** AND **COCO-2017** DATASETS.

Methods	Data Available	ADE-20K		COCO-2017					
		pAcc (%)↑	mIoU↑	mAP^{bbox} ↑	mAP^{bbox}_{50} ↑	mAP^{bbox}_{75} ↑	mAP^{bbox}_{s} ↑	mAP^{bbox}_{m} ↑	mAP^{bbox}_{l} ↑
Teacher	✔	82.49	39.34	34.08	52.91	35.42	19.85	37.51	46.34
Student	✔	78.02	36.91	31.00	48.90	32.70	15.30	33.50	43.10
CMI [32]	✗	74.11	33.72	29.88	45.33	31.04	13.24	29.15	38.47
SpaceShipNet [14]	✗	75.87	34.09	29.47	46.90	30.60	13.52	31.29	41.06
CAE-DFKD (Ours)	✗	**77.14**	**35.66**	**30.43**	**48.26**	**32.18**	**16.79**	**34.43**	**42.35**

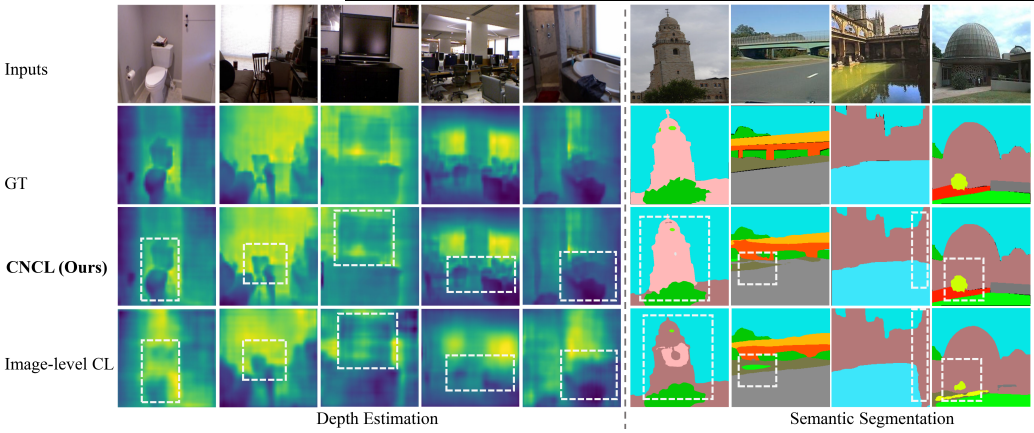


Fig. 5. **Downstream Task Performance Visualization Comparison**. The contrastive learning designed for image-level optimization proves to be less effective in enhancing the generalization ability of the student network in DFKD (Section I). In contrast, our embedding-level contrastive learning encourages the student network to learn domain-invariant class features, which benefits both depth estimation and semantic segmentation.

TABLE VII
ABLATION STUDY ON **CAE-DFKD** COMPONENTS. BASE METHOD: CMI [32]. DATASET: ADE-20K, FOLLOW SETTING IN SECTION IV-B2.

Methods	$\mathcal{L}_{KD}$	CEND	CNCL	pAcc. (%)	mIoU
ResNet-34 → ResNet-18					
Base	✔			$74.11^{0.0}$	$33.72^{0.0}$
+ CAE-DFKD	✔	✔		$75.38^{+1.27}$	$34.48^{+0.76}$
	✔	✔	✔	$\mathbf{77.14}^{+3.03}$	$\mathbf{35.66}^{+1.94}$
WRN-40-2 → WRN-40-1					
Base	✔			$67.52^{0.0}$	$29.47^{0.0}$
+ CAE-DFKD	✔	✔		$70.31^{+2.81}$	$30.97^{+1.50}$
	✔	✔	✔	$\mathbf{72.49}^{+4.97}$	$\mathbf{33.08}^{+3.61}$

TABLE VIII
ABLATION STUDY ON THE NOISE DISTURBANCES N FOR CATEGORY EMBEDDINGS. NYAER [13] AS THE BASE METHOD, FOLLOW SETTING IN SECTION IV-B2. DATASET: NYUV2 SEMANTIC SEGMENTATION (MIOU).

Methods	Effect of the NS Number (N)				
	2	3	**4**	5	6
ResNet-34 → ResNet-18					
Base			$31.82^{0.0}$		
+ CAE-DFKD	$32.14^{+0.32}$	$32.39^{+0.57}$	$\mathbf{33.77}^{+1.95}$	$33.29^{+1.47}$	$33.01^{+1.19}$
WRN-40-2 → WRN-40-1					
Base			$26.84^{0.0}$		
+ CAE-DFKD	$27.24^{+0.40}$	$27.28^{+0.44}$	$\mathbf{29.28}^{+2.44}$	$28.17^{+1.33}$	$28.87^{+2.03}$

TABLE IX
TRAINING EPOCH TIME COMPARISON IN MINUTES (m).

Methods	T → S	
	ResNet-34 → ResNet-18	WRN-40-2 → WRN-16-1
CAE-DFKD **w/o** CEND	2.76m	3.15m
CAE-DFKD w/ CEND	**2.01m**	**1.84m**
SpeedUp	**1.37**×	**1.71**×

design exhibiting a slight negative impact, but it still demonstrates

TABLE X
EFFECT OF LMS ON GENERALIZABILITY, FOLLOW SETTING IN SECTION IV-B2. DATASET: COCO-2017, mAP^{bbox}_{50}.

$T \rightarrow S$	Impact of LMs		
	doc2vec	**CLIP**	SBERT
ResNet-34 → ResNet-18	47.97	**48.26**	47.28
WRN-40-2 → WRN-40-1	44.37	**44.78**	43.12

generalization advantages over other SOTA methods.

TABLE XI
EXPLORING THE IMPACT OF VARIOUS PROMPTS, FOLLOW SETTING IN SECTION IV-B2. DATASET: NYUV2 SEMANTIC SEGMENTATION (MIOU).

T → S	Prompt	Semantic Segmentation	
		mIoU ↑	pAcc (%) ↑
ResNet-34 → ResNet-18	*"a photo of {class name}"*	**33.77**	**60.86**
	"a photo of {class index}"	33.16	60.41
VGG-11 → ResNet-18	*"a photo of {class name}"*	**31.38**	**57.76**
	"a photo of {class index}"	31.04	57.22

VI. CONCLUSION

In this paper, we propose **CAE-DFKD**, which rethinks the existing DFKD framework from the embedding-level view. For technical contributions: ***i)*** CAE-DFKD incorporates a category-aware prior, enriching the structured embedding space and alleviating the generator convergence burden; ***ii)*** Through embedding-level contrastive learning, CAE-DFKD addresses the ineffectiveness of image-level generalization methods for DFKD. For application contributions: CAE-DFKD extends the data-free knowledge transfer potential beyond widely focused image recognition to more diverse downstream tasks. Extensive experiments demonstrate the superiority and flexibility of CAE-DFKD.

REFERENCES

[1] Z. Guo, H. Yan, H. Li, and X. Lin, "Class attention transfer based knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 11 868–11 877.

[2] J. Wang, Y. Chen, Z. Zheng, X. Li, M.-M. Cheng, and Q. Hou, "Crosskd: Cross-head knowledge distillation for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16 520–16 530.

[3] G. Bang, K. Choi, J. Kim, D. Kum, and J. W. Choi, "Radardistill: Boosting radar-based object detection performance via knowledge distillation from lidar features," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 491–15 500.

[4] J. Yuan, M. H. Phan, L. Liu, and Y. Liu, "Fakd: Feature augmented knowledge distillation for semantic segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 595–605.

[5] M. Li, M. Halstead, and C. Mccool, "Knowledge distillation for efficient instance semantic segmentation with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5432–5439.

[6] R. G. Lopes, S. Fenu, and T. Starner, "Data-free knowledge distillation for deep neural networks," *arXiv preprint arXiv:1710.07535*, 2017.

[7] Z. Jiang, J. Gu, M. Liu, and D. Z. Pan, "Delving into effective gradient matching for dataset condensation," in *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2023, pp. 1–6.

[8] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4750–4759.

[9] Y. Wang, Z. Chen, J. Zhang, D. Yang, Z. Ge, Y. Liu, S. Liu, Y. Sun, W. Zhang, and L. Qi, "Sampling to distill: Knowledge transfer from open-world data," *arXiv preprint arXiv:2307.16601*, 2023.

[10] G. Patel, K. R. Mopuri, and Q. Qiu, "Learning to retain while acquiring: combating distribution-shift in adversarial data-free knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7786–7794.

[11] L. Frank and J. Davis, "Data-free knowledge distillation using adversarially perturbed opengl shader images," *arXiv preprint arXiv:2310.13782*, 2023.

[12] Z. Hao, Y. Luo, Z. Wang, H. Hu, and J. An, "Cdfkd-mfs: Collaborative data-free knowledge distillation via multi-level feature sharing," *IEEE Transactions on Multimedia*, vol. 24, pp. 4262–4274, 2022.

[13] M.-T. Tran, T. Le, X.-M. Le, M. Harandi, Q. H. Tran, and D. Phung, "Nayer: Noisy layer data generation for efficient and effective data-free knowledge distillation," *arXiv preprint arXiv:2310.00258*, 2023.

[14] S. Yu, J. Chen, H. Han, and S. Jiang, "Data-free knowledge distillation via feature exchange and activation region constraint," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 24 266–24 275.

[15] Y. Wang, D. Yang, Z. Chen, Y. Liu, S. Liu, W. Zhang, L. Zhang, and L. Qi, "De-confounded data-free knowledge distillation for handling distribution shifts," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 12 615–12 625.

[16] Y. Wang, Z. Chen, J. Zhang, D. Yang, Z. Ge, Y. Liu, S. Liu, Y. Sun, W. Zhang, and L. Qi, "Sampling to distill: Knowledge transfer from open-world data," in *ACM Multimedia 2024*, 2024. [Online]. Available: https://openreview.net/forum?id=zD9m7YE8Gj

[17] M. Li, D. Zhang, T. He, X. Xie, Y.-F. Li, and K. Qin, "Towards effective data-free knowledge distillation via diverse diffusion augmentation," in *ACM Multimedia 2024*, 2024. [Online]. Available: https://openreview.net/forum?id=bPu5rKozS7

[18] G. Fang, K. Mo, X. Wang, J. Song, S. Bei, H. Zhang, and M. Song, "Up to 100x faster data-free knowledge distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, 2022, pp. 6597–6604.

[19] J. Li, S. Zhou, L. Li, H. Wang, J. Bu, and Z. Yu, "Adadfkd: Exploring adaptive inter-sample relationship in data-free knowledge distillation," *Neural Networks*, vol. 177, p. 106386, 2024.

[20] Z. Li, C. Wang, H. Zheng, J. Zhang, and B. Li, "Fakeclr: Exploring contrastive learning for solving latent discontinuity in data-efficient gans," in *European Conference on Computer Vision*. Springer, 2022, pp. 598–615.

[21] Y. Ci, C. Lin, L. Bai, and W. Ouyang, "Fast-moco: Boost momentum-based contrastive learning with combinatorial patches," in *European Conference on Computer Vision*. Springer, 2022, pp. 290–306.

[22] J. Yang, H. Chen, Y. Liang, J. Huang, L. He, and J. Yao, "Concl: Concept contrastive learning for dense prediction pre-training in pathology images," in *European Conference on Computer Vision*. Springer, 2022, pp. 523–539.

[23] M. Yuan, Z. Li, Q. Jin, X. Chen, and M. Wang, "Pointclm: A contrastive learning-based framework for multi-instance point cloud registration," in *European Conference on Computer Vision*. Springer, 2022, pp. 595–611.

[24] Y. Liu, X. Yang, S. Zhou, X. Liu, Z. Wang, K. Liang, W. Tu, L. Li, J. Duan, and C. Chen, "Hard sample aware network for contrastive deep graph clustering," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 7, 2023, pp. 8914–8922.

[25] J. Denize, J. Rabarisoa, A. Orcesi, R. Hérault, and S. Canu, "Similarity contrastive estimation for self-supervised soft contrastive learning," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 2706–2716.

[26] W. Tu, S. Zhou, X. Liu, C. Ge, Z. Cai, and Y. Liu, "Hierarchically contrastive hard sample mining for graph self-supervised pretraining," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[27] W. Zhu, J. Liu, and Y. Huang, "Hnssl: Hard negative-based self-supervised learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4777–4786.

[28] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi, C. Xu, C. Xu, and Q. Tian, "Data-free learning of student networks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3514–3522.

[29] P. Micaelli and A. J. Storkey, "Zero-shot knowledge transfer via adversarial belief matching," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[30] H. Yin, P. Molchanov, J. M. Alvarez, Z. Li, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz, "Dreaming to distill: Data-free knowledge transfer via deepinversion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8715–8724.

[31] Y. Choi, J. Choi, M. El-Khamy, and J. Lee, "Data-free network quantization with adversarial knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 710–711.

[32] G. Fang, J. Song, X. Wang, C. Shen, X. Wang, and M. Song, "Contrastive model inversion for data-free knowledge distillation," *arXiv preprint arXiv:2105.08584*, 2021.

[33] H. Liu, Y. Wang, H. Liu, F. Sun, and A. Yao, "Small scale data-free knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 6008–6016.

[34] X. Li, L. Jiao, Q. Sun, F. Liu, X. Liu, L. Li, P. Chen, and S. Yang, "A category-aware curriculum learning for data-free knowledge distillation," *IEEE Transactions on Multimedia*, 2024.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[37] K. Binici, S. Aggarwal, N. T. Pham, K. Leman, and T. Mitra, "Robust and resource-efficient data-free knowledge distillation by generative pseudo replay," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, 2022, pp. 6089–6096.

[38] K. Binici, N. T. Pham, T. Mitra, and K. Leman, "Preventing catastrophic forgetting and distribution mismatch in knowledge distillation via synthetic data," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2022, pp. 663–671.

[39] K. Do, T. H. Le, D. Nguyen, D. Nguyen, H. Harikumar, T. Tran, S. Rana, and S. Venkatesh, "Momentum adversarial distillation: Handling large distribution shifts in data-free knowledge distillation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 10 055–10 067, 2022.

[40] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 633–641.

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

NeuralMesh: Neural Network For FEM Mesh Generation in 2.5D/3D Chiplet Thermal Simulation

1st Pengju Chen
Southeast University
Nanjing, China
pjchennnn@gmail.com

2nd Dan Niu
Southeast University
Nanjing, China
danniu1@163.com

3rd Dekang Zhang
Southeast University
Nanjing, China
zhdk2001@163.com

4th Wenhao Wang
ShanghaiTech University
Shanghai, China
wangwh2023@shanghaitech.edu.cn

5th Depeng Xie
BTD Technology, Inc
Nanjing, China
xiedepeng1997@gmail.com

6th Zhou Jin
China University of Petroleum
Beijing, China
jinzhou@cup.edu.cn

7th Wei W. Xing
The University of Sheffield
Sheffield, United Kingdom
wayne.xingle@gmail.com

8th Lei He
University of California, Los Angeles
Los Angeles, USA
lhe@ee.ucla.edu

***Abstract*—Advanced integrated circuit (IC) systems increasingly utilize chiplet-based packaging with complex 2.5D/3D structures and dense Through-Silicon Via (TSV) arrays. While the Finite Element Method (FEM) provides high-fidelity thermal simulation for these systems, its computational efficiency degrades significantly when generating and optimizing meshes for intricate geometries. To address these performance limitations while preserving simulation accuracy, we present NeuralMesh, a novel framework that accelerates thermal analysis of chiplet-based ICs. Our approach integrates deep learning and geometric analysis to optimize mesh generation without the need for iterative refinement steps. NeuralMesh first employs an enhanced segmentation model to predict thermal distributions based on geometric, material, and power parameters. These predictions, combined with key geometric features, guide the optimization of an initial coarse FEM mesh. By eliminating traditional iterative mesh refinement, our framework achieves up to 45.00× mesh generation speedup while maintaining thermal accuracy within 0.8% of commercial COMSOL simulations. It reduces the number of mesh elements in unimportant areas, which represents a speed improvement of the subsequent thermal simulation. This advancement enables rapid yet precise thermal analysis essential for modern IC package design.**

***Index Terms*—Chiplet, Thermal Simulation, Finite Element Method, Mesh Optimization, Segmentation Model**

I. Introduction

As semiconductor scaling approaches fundamental physical limits in the post-Moore era, traditional manufacturing process improvements yield diminishing returns in integrated circuit (IC) performance and power efficiency [1]. This challenge has driven the emergence of chiplet-based integration as a promising direction for the IC industry. This approach, illustrated in Fig. 1, employs advanced 2.5D/3D packaging to combine separately manufactured functional chiplets into integrated systems. By enabling the use of different manufacturing processes for different chiplets, this methodology overcomes single-process limitations while reducing defect rates and manufacturing costs [2], [3].

However, the increasing power density and decreasing feature sizes in modern chiplet-based designs have intensified thermal management challenges [4]. Localized hot spots significantly impact chip performance and reliability [5]. Furthermore, temperature gradients combined with mismatched thermal expansion coefficients generate thermal stress that can cause mechanical failures such as delamination, warping, and peeling in chiplet structures. These challenges necessitate precise and efficient thermal simulation methods for optimizing integrated chip designs.

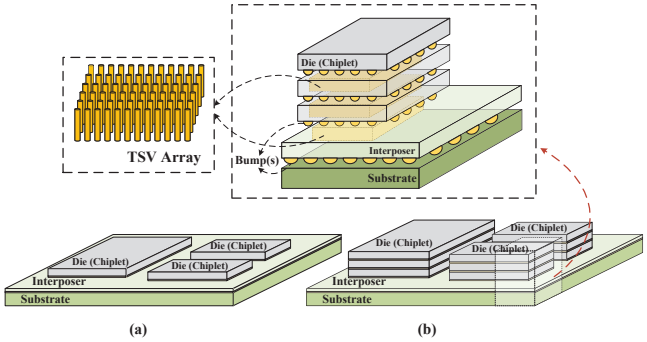


Fig. 1. The structure of integrated chips. **(a)** is the 2.5D packaging and **(b)** is the 3D packaging. The differences between them are: In 3D structure, the lower dies are also embedded with TSVs, and there are micro-bumps between the lower dies and the upper dies.

Current thermal simulation approaches fall into two categories: analytical and numerical methods [6]. While analytical methods offer closed-form solutions, they are generally limited to simple geometries and inadequate for complex integrated chip structures. Numerical methods, particularly the finite difference method (FDM) and finite element method (FEM) discretize heat conduction equations into systems of algebraic equations. Advanced-standard tools implementing these approaches include the FDM-based Hotspot [7] and 3D-ICE [8], and the FEM-based solutions like Manchester Thermal Analyzer (MTA) [9], Celsius [10] and COMSOL [11].

The geometric complexity of modern integrated chips, characterized by intricate stacking structures, redistribution layers (RDL), and dense through-silicon via (TSV) arrays, poses significant challenges for traditional FEM approaches [12]. *Mesh generation and optimization consume a substantial portion of the simulation time when applying FEM to integrated chip thermal analysis*. Recent research has explored

neural networks to accelerate the solution of discrete thermal equations [6], [13]. However, these deep learning approaches struggle with generalization across varied chip structures, and their end-to-end thermal predictions often fall short of simulation requirements [14].

FEM theory demonstrates that properly constructed meshes converge to exact solutions of the original differential equations through progressive refinement [15]. Traditional FEM mesh optimization relies on computationally intensive iterative refinement based on posterior error analysis. Recent active optimization strategies employing graph convolution networks (GCN) [15], machine learning [16], and reinforcement learning [17] have shown promise in improving efficiency. However, these methods focus primarily on 2D mesh optimization and fail to adequately address crucial regions of high temperature and large thermal gradients in integrated chip thermal simulation.

Recognizing that thermal gradient predictions can guide efficient mesh optimization for accurate simulation, we present NeuralMesh, a novel FEM mesh optimization framework that significantly reduces mesh generation time for integrated chip thermal simulation while maintaining accuracy. Our framework leverages segmentation model characteristics for rapid thermal distribution prediction and combines this information with critical geometric features to guide mesh optimization. Our research makes three primary contributions to the field:

- We pioneer a novel mesh optimization framework, NeuralMesh, that for the first time leverages deep learning-based thermal predictions to guide FEM mesh generation to replace traditional iterative refinement with neural network-guided mesh optimization.
- We develop an enhanced segmentation network architecture that achieves relatively high accuracy in thermal distribution prediction on the fly while maintaining real-time performance. The unique structure of our network preserves critical thermal gradient information essential for identifying regions requiring precise mesh refinement, enabling targeted optimization impossible with conventional approaches.
- We introduce an adaptive mesh refinement parameter that unifies thermal gradient data with geometric boundary information to precisely control element density. This innovation enables automatic identification and refinement of thermally critical regions, reducing mesh element count by 44% while maintaining simulation accuracy within 0.8% difference of commercial tools.

II. Background

A. Thermal Simulation of Integrated Chip by FEM

The thermal behavior of integrated chips is governed by the heat conduction equation, which simplifies to a Poisson equation for steady-state analysis:

$$k_x \frac{\partial^2 T}{\partial x^2} + k_y \frac{\partial^2 T}{\partial y^2} + k_z \frac{\partial^2 T}{\partial z^2} + Q_s = 0, \tag{1}$$

where Q_s represents volumetric power dissipation, $\frac{\partial T}{\partial \mathbf{n}} | \mathbf{n} = x, y, z$ denotes temperature gradients along principal axes, and $k_{\mathbf{n}}$ represents directional thermal conductivity. FEM discretizes the solution domain into three-dimensional elements, typically tetrahedra or hexahedra. Within each element, temperature is interpolated using shape functions $N_i(\mathbf{x})$, yielding the approximation $T(\mathbf{x}) = \sum_{i=1}^{s} N_i(\mathbf{x}) T_i$.

At chip-air interfaces, convective heat transfer is modeled using the Robin boundary condition:

$$-k \frac{\partial T}{\partial n} = h(T - T_\infty), \tag{2}$$

where h denotes the convective heat transfer coefficient and T_∞ represents ambient temperature. This convection term contributes to the heat source vector through the boundary integral:

$$F_i = \int_{\Gamma_h} N_i h (T - T_\infty) \, d\Gamma, \tag{3}$$

where Γ_h represents the convective boundary surface.

B. FEM Mesh Optimization

The quality and efficiency of FEM analysis depend critically on mesh characteristics, which are determined by model geometry, feature relationships, and thermal behavior [18], [19]. In integrated chips, geometric features such as device boundaries, inter-feature distances, and small structures like TSVs directly impact mesh requirements. Regions of high temperature or steep thermal gradients demand locally refined meshes to ensure simulation accuracy. For each tetrahedral element e, mesh quality is quantified using the aspect ratio metric:

$$Q_e = \alpha \frac{V_e}{h_e^3} \tag{4}$$

where V_e is the element volume, h_e is the longest edge length, and α is a normalization constant. The a posteriori error estimator η_e for thermal analysis is defined as:

$$\eta_e^2 = h_e^2 |R(T_h)|0, e^2 + \frac{1}{2} h_e |J(T_h)|0, \partial e^2 \tag{5}$$

where $R(T_h)$ is the element residual and $J(T_h)$ represents the jump in temperature gradient across element boundaries. Element subdivision uses the refinement criterion:

$$h_{new} = h_{old} \sqrt{\frac{\eta_e}{\eta_{target}}}, \tag{6}$$

whereas mesh reconstruction generates new elements based on the size function:

$$h(\mathbf{x}) = \min(h_{max}, \max(h_{min}, \beta |\nabla^2 T|^{-1/p})), \tag{7}$$

where β controls refinement intensity and p is the polynomial order. r-refinement optimizes node positions by minimizing the energy functional:

$$E(\mathbf{x}) = \int_\Omega |\nabla T|^2 d\Omega + \gamma \sum_e Q_e^{-2} \tag{8}$$

The selection of the optimization method depends on the specific requirements for accuracy, computational efficiency, and mesh quality constraints in the thermal simulation.

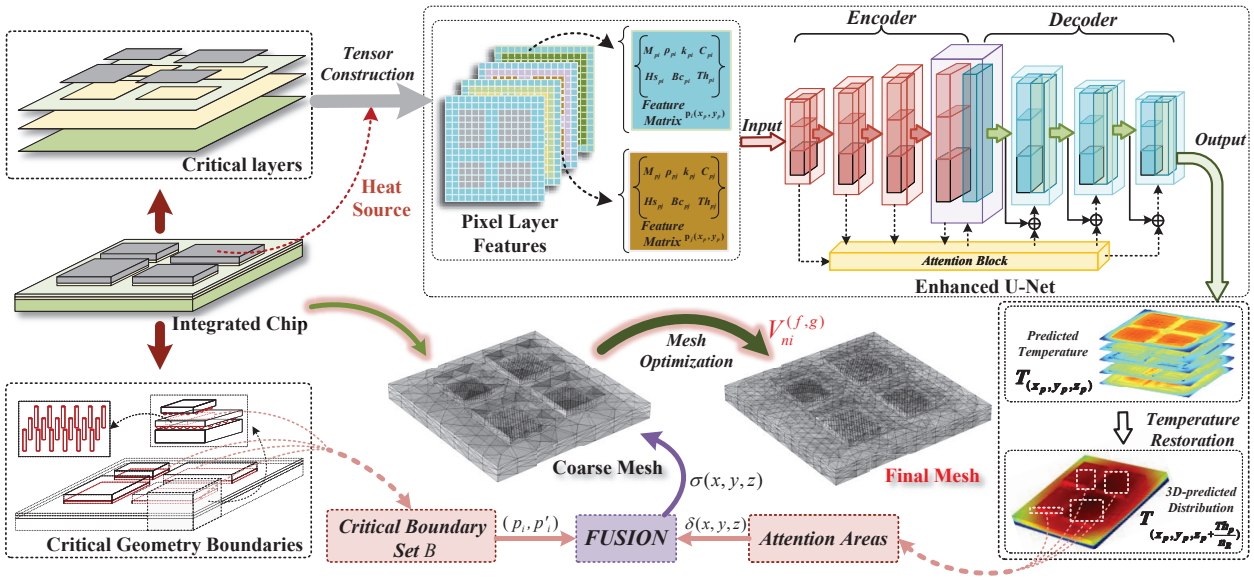


Fig. 2. The mesh optimization framework NeuralMesh for getting final refined FEM mesh.

III. Proposed Methodologies

Our NeuralMesh framework accelerates FEM mesh generation for integrated chip thermal simulation while maintaining accuracy. As illustrated in Fig. 2, the framework comprises three key stages: (1) geometric and thermal feature extraction through tensor construction, (2) temperature distribution prediction via an enhanced U-Net architecture, and (3) intelligent mesh optimization guided by predicted thermal gradients and geometric constraints.

A. Model Input and Tensor Construction

The framework integrates two critical information sources: geometric boundaries and predicted thermal distributions. For geometric boundaries, we define the edge set $E = \{e_i(p_i, p_i')\}, i = 1, 2, ..., N$, where each edge e_i connects points $(p_i, p_i') = [(x_i, y_i, z_i), (x_i', y_i', z_i')]$. Component volumes form the set $V = \{v_i^{(comp)}\}, i = 1, 2, ..., M$, and $comp = \{TSV, die, ..., substrate\}$. Critical boundaries are

$$f_{crit}(e) = e \in [v_i^{comp(i)} \wedge v_j^{comp(j)}], i \neq j \tag{9}$$

where $v_i^{comp(i)}$ and $v_j^{comp(j)}$ are the volumes of component i and j respectively. Yielding the boundary set $B = \{e_i\}$ where $f_{crit}(e_i) = True$. For thermal prediction, we decompose the 3D structure into critical layers based on material transitions. Each layer forms a 2D plane P_i with feature tensors encoding local physical properties:

$$\mathbf{p}(x_p, y_p) = \left[M_p, \rho_p, k_p, C_p, Hs_p, Bc_p, Th_p\right]^\top \tag{10}$$

where M_p, ρ_p, k_p, C_p, Hs_p, Bc_p, and Th_p represent material properties, density, thermal conductivity, heat capacity, heat source, boundary conditions, and Z-axis thickness respectively. The complete plane tensor is:

$$\mathbf{P} = [\mathbf{p}(x_p, y_p)]_{x=1,...,W;,y=1,...,H} \in \mathbb{R}^{W \times H \times C_f}, \tag{11}$$

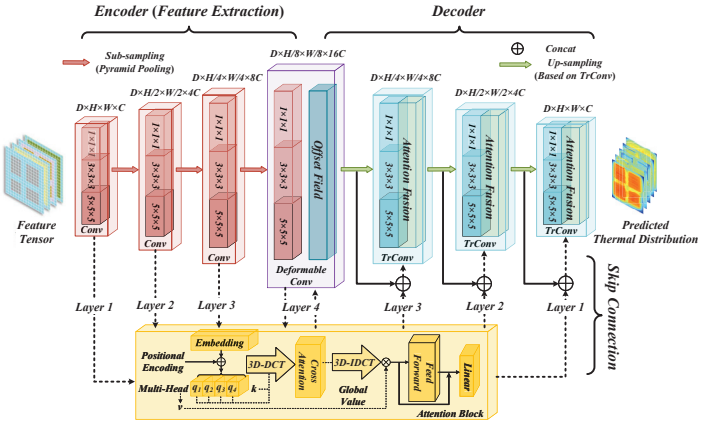


Fig. 3. The Enhanced U-NET for Temperature Prediction

where C_f is the number of channels of the feature. The tensor P is used as the input of the enhanced U-Net network proposed in this paper, which will be completely described in the next section, to predict the temperature distribution.

B. Temperature Prediction based on Enhanced U-NET

Our method first aims to predict the accurate temperature distribution of integrated chips based on the feature tensor constructed in the previous section. This segmentation model is implemented based on the enhanced U-Net architecture (illustrated in Fig. 3), which extends the traditional encoder-decoder structure with multi-scale convolution and pyramid pooling to capture diverse structural features. The network comprises three key components:

1) Encoder Structure: The encoder is the key feature extraction component in the U-NET, which is designed to capture the thermal characteristics of the integrated chip from local to global. In the encoder component, the following strategies are made to improve the extraction of the plane feature:

- **Multi-scale Convolution Layer:** Since the structure of the integrated chip is layered into plane tensors, it is

necessary to capture the local detail and wide-area thermal features of the integrated chip in a single layer. The multi-scale convolution strategy with various sizes convolution kernels (1×1×1, 3×3×3 and 5×5×5) in parallel can capture structural information from fine (TSV) to coarse (substrate). This design allows the U-Net to more comprehensively understand the thermal characteristics of the integrated chip, hereby improving the representation ability of the features.

- **Pyramid Pooling:** After the Multi-scale convolution layer, the sub-sampling based on pyramid pooling can enhance the spatial capture capability of U-Net by applying different sizes of pooling windows. It not only improves sensitivity to different-scale features but extracts more global thermal information.
- **Deformable Convolution:** Since different areas of a material respond differently to heat conduction, after the 3 layers of multi-scale convolution and pyramid pooling, the deformable convolution enables the U-Net more flexible to the heterogeneity of integrated chips.

2) Attention Block-based Skip Connection: Skip connection can effectively avoid the gradient disappearance in the enhanced U-Net training process. The attention block with multi-head attention mechanism-based skip connection can enable the decoder to reconstruct and refine the specific details of the temperature distribution in the large temperature gradient changed area. The 3D discrete cosine transform (3D-DCT) is applied to convert spatial features into the frequency domain, and the 3D inverse discrete cosine transformation (3D-IDCT) transforms the frequency branch back into the spatial domain to enrich the semantic content. This method effectively improves the efficiency of feature reuse and increases the accuracy of the enhanced U-NET.

3) Decoder Structure: The decoder is responsible for up-sampling the chip features, which are extracted by the decoder, to the original resolution. And then generating the temperature distribution prediction map of the integrated chips.

- **Transposed Convolution (TrConv):** The transposed convolution is the up-sampling strategy of the decoder part. It enables the output details of each layer to gradually closer to the high-resolution temperature map of the integrated chips.
- **Attention Fusion:** After each up-sampling, the features of the corresponding encoder layer are concat with the output of the current decoder layer after passing through the skip connection based on the attention block. By fusing the attention feature from the Attention block, the attention fusion module in the decoder performs weighted processing on each feature, which can give priority to focusing and refining the areas with large temperature gradient changes and concentrated hot spots.

Through the design of the above structure, the enhanced U-Net can effectively extract key features from the 2D tensor structure slices of complex integrated chips and perform precision temperature distribution prediction.

C. Mesh Optimization

While our U-Net provides accurate thermal predictions, further refinement is needed for simulation-grade accuracy, which is essential for an industrial standard of thermal simulation. To this end, we utilize the predicted temperature distribution to optimize the coarse FEM mesh based on the critical boundaries and the temperature distribution prediction.

1) 3D Temperature Field Reconstruction: The initial thermal predictions exist on discretized planes corresponding to material interfaces. To obtain a continuous 3D temperature field, we must first reconstruct the full volumetric distribution. Within regions of consistent material properties, the temperature field satisfies the steady-state heat equation:

$$\nabla \cdot (k \nabla T) = -q \tag{12}$$

where k is the thermal conductivity tensor and q represents volumetric heat generation. Given this governing equation and the assumption of piece-wise constant material properties between layers, we employ a physics-informed interpolation scheme:

$$T_{(x_p,y_p,z_p+\frac{Th_p}{n_R})} = (1-\frac{1}{n_R})T_{(x_p,y_p,z_p)} + \frac{1}{n_R}T_{(x_p,y_p,z_p+Th_p)} \tag{13}$$

where n_R determines the interpolation resolution along the z-axis. This approach preserves thermal continuity while respecting material boundaries.

2) Thermal-Geometric Refinement Parameter: The effectiveness of mesh refinement depends critically on identifying regions requiring higher resolution. We propose a novel refinement parameter that couples thermal behavior with geometric features. The thermal contribution is captured through a normalized parameter:

$$\lambda(x_p, y_p, z_p) = \frac{T_{(x_p,y_p,z_p)} - T_{min}}{T_{max} - T_{min}} \cdot \frac{|\nabla T_{(x_p,y_p,z_p)}|}{|\nabla T_{(x_p,y_p,z_p)\max}|} \tag{14}$$

This formulation balances absolute temperature magnitude with local gradients, addressing both hot spots regions and areas of rapid thermal variation. The inverse transformation:

$$\delta(x_p, y_p, z_p) = \frac{1}{1-\lambda(x_p, y_p, z_p)} \tag{15}$$

provides appropriate scaling for mesh refinement decisions. To incorporate geometric features, we transform the thermal parameter from discretized coordinates to the continuous geometric domain:

$$\delta(x, y, z) = \delta(x_p, y_p, z_p) \cdot \wp[\frac{\mathbf{n}p}{\mathbf{Ln}}|\mathbf{N}|\mathbf{n}_{p-1} : \mathbf{n}_{p+1}] \tag{16}$$

where $\wp[\cdot]$ represents a neighborhood interpolation function, $\mathbf{L_n}$ denotes directional pixel counts, and $|\mathbf{N}|$ gives physical dimensions. The final refinement parameter synthesizes thermal and geometric information:

$$\sigma(x, y, z) = \frac{e \cdot \delta(x, y, z)}{1 + [\sqrt{|(x, y, z) - p_i|^2 + |(x, y, z) - p_i'|^2}]_{min}} \tag{17}$$

TABLE I
DETAILS OF BENCHMARKS AND THE COMPARISON OF DIFFERENT MESH GENERATION METHODS

Integrated Chips	Number of			Number of Mesh Elements			Running Time of Mesh Generation (s)			Time Speedups	
	Chiplets	Micro-bumps	TSVs	COMSOL	Gmsh	NeuralMesh	COMSOL	Gmsh	NeuralMesh	vs COMSOL	vs Gmsh
Chip_2.5D_10_2	2	200	1800	2,430,598	1,733,437	1,037,232	60.014	124.620	19.944	3.00×	6.25×
Chip_2.5D_10_3	3	180	4200	3,902,809	2,980,222	2,229,312	90.997	195.151	26.456	3.42×	7.38×
Chip_2.5D_10_4	4	160	4200	3,714,610	2,930,852	2,391,776	100.614	165.218	25.715	3.91×	6.43×
Chip_2.5D_20_3	3	400	8400	7,252,363	5,021,381	4,385,152	423.204	475.272	33.242	12.73×	14.30×
Chip_2.5D_20_4	4	360	9600	7,334,624	5,926,821	5,096,576	458.872	492.657	30.441	15.07×	16.18×
Chip_3D_10_6	6	600	12600	33,901,129	24,215,092	17,133,312	2116.153	2401.833	47.025	**45.00**×	**51.07**×
Chip_3D_20_12	12	1200	15000	36,380,226	27,409,756	18,164,713	2364.808	2648.585	55.181	42.86×	48.00×
Average	-	-	-	-	-	-	-	-	-	18.00×	29.91×

This formulation ensures enhanced resolution near material interfaces and in regions of significant thermal activity while maintaining smooth transitions in element size.

3) Adaptive Tetrahedral Refinement: For tetrahedral elements, we employ an edge-based refinement strategy. Given an element m_{i0} with vertices $[(x, y, z)_f, f = 1, 2, 3, 4]$, the number of new vertices along edge (f, g) is determined by:

$$V^{(f,g)}ni = \Lambda[\sigma(x, y, z)_f + \sigma(x, y, z)_g] \tag{18}$$

where $\Lambda[\cdot]$ represents the ceiling function. The positioning of new vertices follows an exponential distribution governed by local σ values, ensuring smooth size transitions while maintaining element quality.

This approach satisfies several key theoretical properties including: (1) Element quality bounds are preserved through controlled vertex placement, (2) Mesh size transitions remain gradual, preventing numerical artifacts, (3) Refinement naturally concentrates in regions of high thermal gradient, (4) Material interfaces receive appropriate resolution enhancement.

The validation demonstrates that this algorithm achieves significant computational efficiency while maintaining accuracy. For complex 3D integrated chips, our framework demonstrates up to 51× acceleration compared to conventional approaches, with thermal prediction errors remaining below 0.8% relative to commercial FEM solvers.

IV. EXPERIMENTS AND RESULTS

In this section, We evaluate the NeuralMesh design and present our evaluation results. We first introduce the experimental setup. And then, in order to assess the efficiency of the NeuralMesh framework, we compare the mesh generate time and the number of mesh elements among the NeuralMesh, COMSOL [11] and the wild-used open source software Gmsh [20]. Finally, we verify the thermal simulation accuracy of different mesh refine algorithm by comparing the heat distribution from SOTA commercial software COMSOL.

A. Experimental Setup

Benchmark: Our evaluation employs seven industrial-grade integrated circuit test cases derived from Cadence reference designs, encompassing both 2.5D and 3D packaging architectures. The vertical stack comprises a substrate layer, C4 bumps, TSV-embedded interposer layer, micro-bumps, and dies. Test cases feature varied chiplet configurations randomly distributed across different substrate scales. TSV and micro-bump arrays follow local-uniform distribution patterns beneath chiplets, with specifications detailed in Table I.

Methodology and Platform: We conducted all experiments on an AMAX X12SPA-TF server equipped with an NVIDIA RTX 3090 (24GB VRAM) and Intel Xeon Gold 6336Y CPU (2.40GHz, 256GB RAM). The enhanced U-Net was pre-trained using thermal distributions from COMSOL simulations as ground truth. We evaluate mesh generation efficiency through element count and generation time metrics, comparing NeuralMesh against COMSOL and Gmsh implementations, which implement the SOTA mesh generation methods.

B. Effectiveness of Final Mesh Generation

We conducted a comprehensive evaluation comparing the proposed NeuralMesh against advanced-standard COMSOL and open-source Gmsh across diverse test cases which are mentioned above. The number of mesh elements and the mesh generation time of the final optimized FEM mesh are presented in Table I.

The number of mesh element reductions achieved by NeuralMesh ranges from 30.5% (Chip_2.5D_20_4) to 57.3% (Chip_2.5D_10_2) compared to COMSOL, with average reduction rate of about 44%, and is reduced averagely by about 25% compared to Gmsh. This efficiency stems from the ability of our framework to selectively refine thermally critical regions while maintaining coarser meshes elsewhere. The reduction in the number of elements directly translates to computational savings during the subsequent thermal simulation.

The running time of mesh generation based on NeuralMesh achieves significant improvement both compared to COMSOL and Gmsh, especially in the complex integrated chips. According to the improvements in mesh generation time, there is a super-linear scaling with the complexity of test cases. For simpler 2.5D structures (Chip_2.5D_10_2), NeuralMesh achieves 3.00× speedup versus COMSOL. And for the complex 3D structures (Chip_3D_10_6), this advantage increases dramatically to 45.00×. It is indicated that our segmentation model guided approach becomes increasingly effective as geometric complexity grows. And the performance gain primarily derives from eliminating redundant refinement iterations through predictive thermal distribution.

C. Accuracy of Thermal Simulation

In order to validate the thermal simulation accuracy of the FEM mesh obtained by NeuralMesh, we performed thermal simulation using COMSOL as the solver across FEM meshes generated by NeuralMesh, Gmsh and COMSOL. All the simulations are equipped with identical power distributions

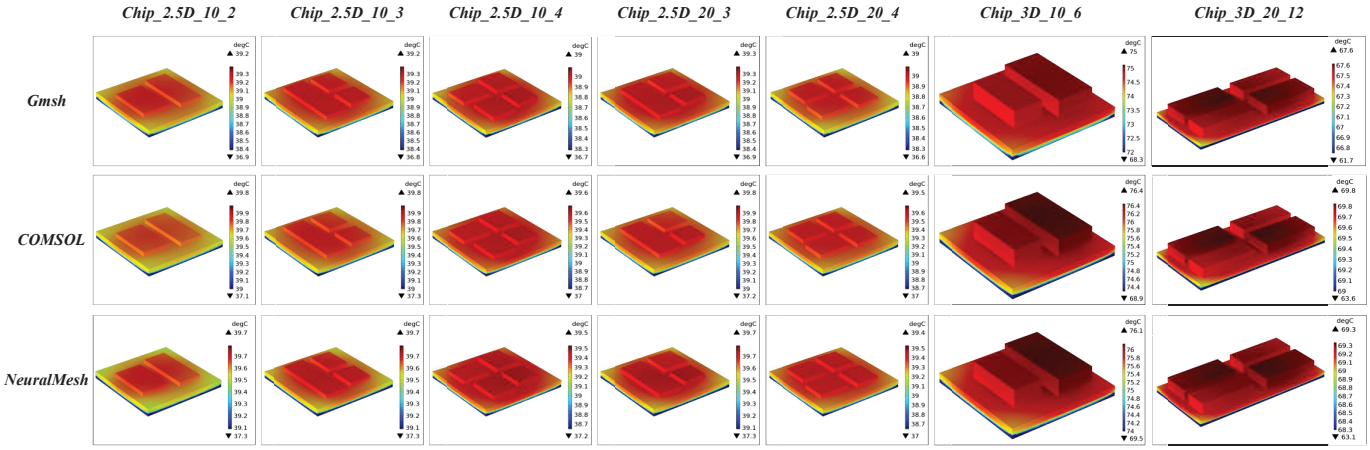


Fig. 4. Thermal Maps of Integrated Chips from Different FEM Mesh Generation Method

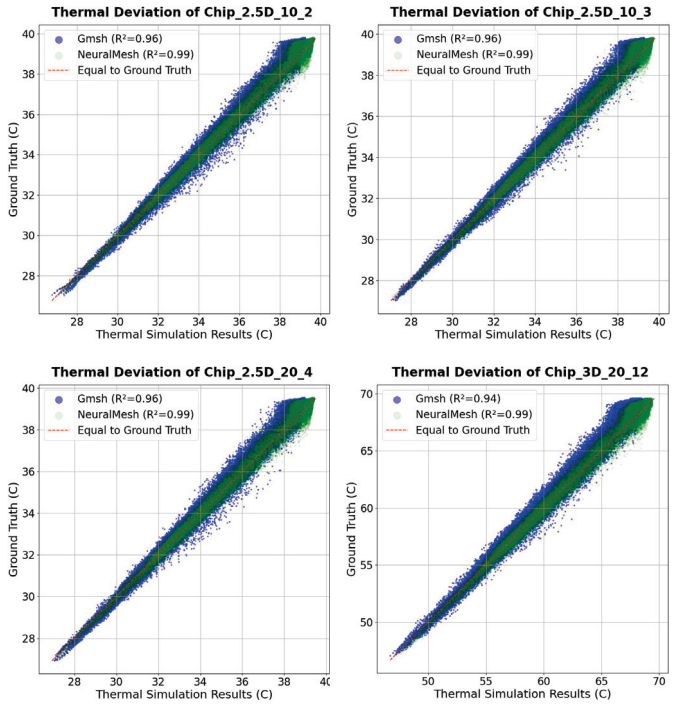


Fig. 5. Thermal Deviation of Representative Benchmarks

and boundary conditions. The Fig.4 presents the thermal maps of benchmarks from different FEM mesh generation methods, with the results of COMSOL serving as the golden reference.

The thermal distribution of the integrated chip main body with the maximum and minimum values of the temperature in Fig.4 directly illustrates the agreement of NeuralMesh and golden reference (COMSOL). The maximum difference in temperature ranges from 0.2 degC to 0.5 degC, which is from the simplest test case (Chip_2.5D_10_2) to the most complex test case (Chip_3D_20_12), and the error percentages are kept within 0.8%. Moreover, according to the comparison between the thermal results from Gmsh and the proposed NeuralMesh, it can be concluded that the thermal simulations based on NeuralMesh are more accurate than the Gmsh-based.

Furthermore, in order to more intuitively compare the accuracy of thermal simulation results based on NeuralMesh and Gmsh, we plot the thermal deviation images of four representative benchmarks including the temperature of the surrounding air field, which are illustrated in Fig. 5. According to the Coefficient of Determination R^2, it can easily obtain that the thermal distribution results based on Gmsh are more scattered than those based on NeuralMesh because their index R^2 are more deviated from *1*.

All these analysis above demonstrate that NeuralMesh achieves substantial computational efficiency without compromising thermal accuracy. The preservation of thermal gradients, particularly in critical regions, validates our approach of coupling neural prediction with geometric refinement criteria.

V. Conclusion

This paper presents NeuralMesh, a novel finite element mesh optimization framework that accelerates the thermal simulation of integrated circuits by combining neural network-based thermal prediction with geometric analysis. Our approach achieves up to 45× speedup in mesh generation while maintaining thermal accuracy within 0.8% of commercial solutions, demonstrating particular efficiency gains for complex 3D packaging structures. While the framework requires significant computational resources for initial training and currently focuses on steady-state analysis, these limitations are outweighed by the dramatic reduction in mesh generation time. The effectiveness of the proposed framework in selectively refining thermally critical regions while minimizing element count represents a significant advancement in thermal-aware IC design, though future work should address dynamic thermal analysis and broader packaging architectures. Despite these constraints, NeuralMesh establishes a new paradigm for efficient thermal simulation of integrated circuits, enabling rapid design iteration without compromising accuracy.

VI. Acknowledgment

Corresponding authors are Dan Niu and Wei W. Xing. This work is partially supported by NSFC (Grant No. 62374031), Natural Science Foundation of Jiangsu Province (BK20240173), NSFC (Grant No. 62234010, 92473107, 62204265, U23A20301).

REFERENCES

[1] X. Ma, Y. Wang, Y. Wang and Y. Han. Survey on chiplets: interface, interconnect and integration methodology. *CCF Transactions on High Performance Computing*, vol. 4, pp. 43-52, 2022.

[2] Y. Li, A. Louri and A Karanth. Scaling deep-learning inference with chiplet-based architecture and photonic interconnects. in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 931-936, 2021.

[3] Y. Feng and K. Ma. Chiplet actuary: A quantitative cost model and multi-chiplet architecture exploration. in *2022 59th ACM/IEEE Design Automation Conference (DAC)*, pp. 121-126, 2022.

[4] D. Rich, A. Kasperovich, M. Malakoutian, R.M. Radway, S. Hagiwara and T. Yoshikawa. Thermal Scaffolding for Ultra-Dense 3D Integrated Circuits. in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2023.

[5] J. Park, A. Kanani, L. Pfromm, H. Sharma, P. Solanki, E. Tervo, J.R. Doppa, P.P. Pande and U.Y. Ogras. Thermal Modeling and Management Challenges in Heterogenous Integration: 2.5 D Chiplet Platforms and Beyond. in *2024 IEEE 42nd VLSI Test Symposium (VTS)*, pp. 1-4, 2024.

[6] L. Chen, J. Lu, W. Jin and S.X.D. Tan. Fast full-chip parametric thermal analysis based on enhanced physics enforced neural networks. in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1-8, 2023.

[7] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron and M.R. Stan. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 14, pp. 501-513, 2006.

[8] F. Terraneo, A. Leva, W. Fornaciari, M. Zapater and D. Atienza. 3D-ICE 3.0: efficient nonlinear MPSoC thermal simulation with pluggable heat sink models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, pp. 1062-1075, 2021.

[9] S. Ladenheim, Y.C. Chen, M. Mihajlović and V.F. Pavlidis. The MTA: An advanced and versatile thermal simulator for integrated systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 3123-3136, 2018.

[10] R. Mathur, C.J. Chao, R. Liu, N. Tadepalli, P. Chandupatla and S. Hung. Thermal analysis of a 3D stacked high-performance commercial microprocessor using face-to-face wafer bonding technology. in *2020 IEEE 70th Electronic Components and Technology Conference (ECTC)*, pp. 541-547, 2020.

[11] R.W. Pryor. Multiphysics modeling using COMSOL®: a first principles approach. *Jones & Bartlett Publishers*, 2009.

[12] T. Lu, F. Zhang and J.M. Jin. Multiphysics simulation of 3-D ICs with integrated microchannel cooling. *IEEE transactions on components, packaging and manufacturing technology*, vol. 6, pp. 1620-1629, 2016.

[13] Z. Liu, Y. Li, J. Hu, X. Yu, S. Shiau, X. Ai, Z. Zeng, and Z. Zhang. DeepOHeat: operator learning-based ultra-fast thermal simulation in 3DIC design. in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2023.

[14] C. Nie, Q. Xu, C. Wang, H. Cao, J. Liu and Z. Li. Efficient transient thermal analysis of chiplet heterogeneous integration. *Applied Thermal Engineering*, vol. 229, 2023.

[15] M. Kim, J. Lee and J. Kim. GMR-Net: GCN-based mesh refinement framework for elliptic PDE problems. *Engineering with Computers*, vol. 39, pp. 3721-3737, 2023.

[16] P.F. Antonietti, F. Dassi and E. Manuzzi. Machine learning based refinement strategies for polyhedral grids with applications to virtual element and polyhedral discontinuous Galerkin methods. *Journal of Computational Physics*, vol. 469, 2022.

[17] J. Yang, T. Dzanic, B. Petersen, J. Kudo, K. Mittal, V. Tomov, J.S. Camier, T. Zhao, H. Zha, T. Kolev, R. Anderson and D. Faissol. Reinforcement learning for adaptive mesh refinement. in *International Conference on Artificial Intelligence and Statistics (PMLR)*, pp. 5997-6014, 2023.

[18] W. Xu, D. Jiang, C. Zhang, H. Li, S. Qiang, Y. Li, M. Yuan and C. Zhang. An adaptive mesh refinement strategy for 3D phase modeling of brittle fracture. *Engineering Fracture Mechanics*, vol. 284, 2023.

[19] G. Gibert, B. Prabel, A. Gravouil and C. Jacquemoud. A 3D automatic mesh refinement X-FEM approach for fatigue crack propagation. *Finite Elements in Analysis and Design*, vol. 157, pp. 21-37, 2019.

[20] C. Geuzaine and J.F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, vol. 79, pp. 1309-1331, 2009.

EPOC: An Efficient Pulse Generation Framework with Advanced Synthesis for Quantum Circuits

Jinglei Cheng[¶*], Yuchen Zhu[†*], Yidong Zhou[†], Hang Ren[‡], Zhixin Song[§], Zhiding Liang[†]
[¶]University of Pittsburgh [†]Rensselaer Polytechnic Institute [‡]University of California, Berkeley [§]Georgia Institute of Technology
[*]Authors contributed equally to this research. Corresponding to: jic373@pitt.edu, liangz9@rpi.edu

***Abstract*—In this work, we aim to address the computational overhead challenge in quantum optimal control while reducing circuit latency. We propose a novel approach combining ZX-Calculus, circuit partitioning, and circuit synthesis for pulse generation. By implementing finer granularity in pulse generation and exploring equivalent circuit representations, we achieve increased parallelism and decreased latency. Our method demonstrates a 31.74% reduction in latency compared to previous work and a 76.80% reduction compared to gate-based pulse generation methods, while minimizing computational overhead.**

***Index Terms*—quantum optimal control, quantum compilation, quantum circuit optimization**

I. Introduction

Quantum computing has emerged as a promising paradigm with the potential to outperform classical computers in various domains [1], including chemistry [2], machine learning [3], and physical simulations [4]. However, the current Noisy Intermediate-Scale Quantum (NISQ) era [5] presents significant challenges, such as limited qubit count, short coherence times, and high error rates [6]. To exploit the power of NISQ devices effectively, it is crucial to develop efficient quantum circuit optimization techniques that mitigate the impact of noise and errors [7]. The coherence time determines the duration and depth of quantum circuits that can be successfully executed on real quantum hardware. The term "quantum volume" has been proposed to quantitatively describe this property, which takes into account both the number of qubits and the depth of the quantum circuit that can be reliably executed on a given quantum device [8].

A typical quantum program compilation workflow starts with implementing quantum algorithms as quantum circuits. As shown in Figure 1, these circuits undergo compilation passes to become compatible with the hardware topology, including decomposition into basis gates and mapping to the target architecture [9]. Once expressed in basis gates, the gates are converted into physical signals applied to the qubits [10]. For superconducting quantum computers, these physical signals are modulated microwave pulses sent to transmons [11]. The modulated pulses are generated with an envelope that determines their function and a carrier signal.

In the final step of the compilation process, where basis gates are translated to pulses, quantum optimal control (QOC) [12] is used to find optimal control pulses that implement desired unitary operations, minimizing noise and errors [13]. However, QOC has significant computational overhead, limiting its scalability [14]. Previous works have proposed methods to accelerate QOC, such as utilizing GPUs, pre-compilation techniques, and improved circuit libraries [15], [16], [17]. These pulse generation frameworks focus on generating pulses from unitary matrices directly derived from quantum circuits without exploring equivalent representations, thus overlooking optimization opportunities by not adopting coarse-grained methods.

By coarse-grained methods, we refer to existing frameworks such as AccQOC [18] and PAQOC [19], which perform circuit partitioning at the gate level. These methods first group the quantum gates in the circuit and calculate the corresponding unitary matrices for each

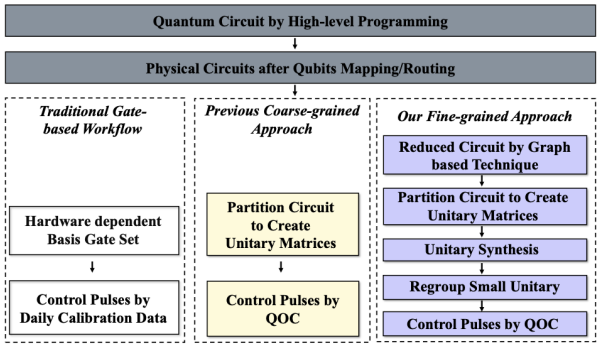


Fig. 1: The diagram compares the traditional gate-based workflow for creating quantum circuits with our novel fine-grained approach.

group. QOC is then applied to these unitary matrices to generate the optimized control pulses. While this approach can provide some benefits in terms of pulse optimization, it is limited by the inherent gate structure of the quantum circuit. If the depth of the circuit blocks does not match, leading to mismatched pulse latencies, the utilization rate of the qubit lines, where the pulses are applied, may not be optimal. This can result in suboptimal performance and longer overall execution times.

In this paper, we aim to propose a fine-grained approach in contrast to the previous coarse-grained methods and traditional workflow, as depicted in Figure 1. Thus, we introduce EPOC, a novel approach that combines ZX-calculus [20], circuit partitioning, and circuit synthesis with QOC to enhance pulse generation. Our main contribution is the development of a finer-grained method for generating quantum pulses, which results in increased parallelism and reduced latency. This is accomplished by partitioning circuits into blocks, computing their corresponding unitary matrices, and applying synthesis techniques to identify equivalent circuits composed of unitary gates. By breaking down relatively large circuit blocks into smaller unitary gates, we create additional opportunities for optimization, enabling more efficient and effective pulse generation. However, directly applying QOC to these small unitary gates, which are the products of synthesis, presents a challenge. The issue arises because these blocks are typically too small to fully leverage the advantages of QOC. To address this, we need to aggregate these blocks to achieve a suitable size that allows for effective application of QOC techniques. Therefore, we introduce a regrouping process for the unitary gates, where different unitary gates and two-qubit gates like CNOT are aggregated into a single unitary matrix. This regrouping technique allows us to further trade classical computational power for quantum computational power while limiting the size of the resulting matrices to avoid excessive computational overhead [18].

Finally, we apply QOC to the regrouped unitary matrices to obtain the optimized microwave pulses for the qubits. By combining QOC

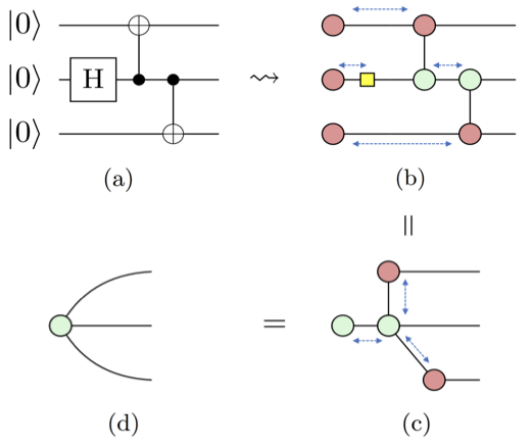


Fig. 2: ZX-Diagram of a GHZ-state preparation circuit. (a) Quantum circuit for preparing a 3-qubit GHZ state $(|000\rangle + |111\rangle)/\sqrt{2}$. (b) Corresponding ZX-diagram. Simplifications are made using rewriting rules such as spider fusion, which is indicated by dashed double-arrow lines for contractable spiders. (c) Further simplification. (d) Final compact representation of the GHZ state [21].

with the synthesized and regrouped unitary matrices, we generate highly optimized and robust quantum circuits tailored to the specific characteristics of the target quantum hardware.

The main contributions can be summarized as follows:

- We propose EPOC, a framework that identifies the optimal circuit size for combining QOC with synthesis techniques to enable efficient pulse generation while managing compilation overhead.
- We introduce a regrouping strategy that combines quantum gates into optimally-sized (not smallest but balanced) unitary matrices to achieve a balance between synthesis optimization potential and QOC benefits.
- We demonstrate through extensive benchmarking that this balanced approach achieves significant improvements in circuit latency and compilation efficiency compared to existing methods.

The remainder of this paper is organized as follows. Section I provides an introduction, followed by the background information in Section II. We introduce our proposed framework, EPOC, in Section III. The evaluation of EPOC is presented in Section IV. Section V discusses related work, and we conclude the paper in Section VI.

II. Background

A. ZX-Calculus

The ZX-calculus, developed by Bob Coecke and Ross Duncan [22], [23], offers a graphical language tailored for representing quantum states and operations on qubits. It utilizes the formalism of string diagrams, with Pauli-X and Pauli-Z matrices as representing basis, thus termed ZX-diagrams. ZX-diagrams represent the topological configuration of quantum circuits, allowing transformations that do not alter qubit connections due to their topological invariance. The calculus includes a set of graphical rewrite rules, collectively known as the ZX-calculus rules. Fundamentally, ZX-diagrams are composed of building blocks called "spiders", which represent various quantum computing elements such as isometries, states, unitary operations, and measurements in specific bases. These diagrams feature nodes colored in green and red, where green nodes correspond to the computational basis states $|0\rangle$ and $|1\rangle$, and red nodes represent the Hadamard-transformed basis states $|+\rangle$ and $|-\rangle$. Additionally, Hadamard nodes are depicted as yellow boxes. Connections in the diagram are made by wires, which can curve, cross, and contract. In Fig. 2, we give an example showing the ZX-diagram of a GHZ state which illustrates the ZX-Calculus and the rewriting rule for circuit simplification.

The theoretical foundation of ZX-Calculus is rooted in the field of dagger compact categories [24], a discussion of which is beyond the scope of this paper but can be explored in further readings [25]. ZX-calculus has found application in various domains of quantum information and computation, including measurement-based quantum computation [26], quantum error correction [27], [28], and quantum circuit optimization [29]. The application of ZX-calculus to quantum circuit optimization is particularly relevant to ours.

B. Synthesis

Quantum circuit synthesis plays a crucial role in quantum computing by generating efficient and optimized quantum circuits from high-level mathematical descriptions of quantum algorithms. Synthesis techniques aim to decompose unitary matrices, representing quantum transformations, into a sequence of elementary quantum gates that can be directly executed on quantum hardware [30]. The quality of synthesized circuits is typically evaluated based on the number of gates, particularly the count of expensive two-qubit gates like CNOT, as well as the overall circuit depth [31]. Shorter circuits with fewer gates are highly desirable, especially for NISQ devices characterized by limited coherence times and noisy operations. Effective synthesis methods can significantly reduce the computational overhead and improve the reliability of quantum algorithms on real hardware [32], [33]. Moreover, quantum circuit synthesis enables hardware design exploration, algorithm discovery, and circuit optimization. Existing synthesis approaches can be broadly categorized into top-down methods, which employ rule-based decomposition techniques, and bottom-up methods, which utilize numerical optimization and search strategies to construct circuits incrementally.

C. Quantum Optimal Control

In the context of a closed quantum system, the dynamics can be described by a Hamiltonian, given as:

$$H(t) = H_0 + \sum_{j=1}^{n} u_j(t) H_j, \tag{1}$$

where H_0 represents the drift Hamiltonian that accounts for the intrinsic evolution of the system, and H_j are the control Hamiltonians modulated by time-dependent control signals $u_j(t)$. The temporal evolution of the quantum state $|\psi\rangle$ in this system is governed by the Schrödinger equation:

$$\frac{d}{dt}|\psi(t)\rangle = -iH(t)|\psi(t)\rangle, \tag{2}$$

assuming the initial state of the system at time $t = 0$ is $|\psi(0)\rangle$.

QOC aims to determine the optimal control signals $u_j(t)$ that transition the system from an initial quantum state to a desired target state. This control strategy is particularly critical in achieving precise state transformations while considering system constraints and minimizing operational errors. To optimize the control signals, QOC employs a cost function, typically defined in terms of the fidelity between the achieved unitary transformation induced by the applied control signals and a desired unitary goal. This fidelity metric quantifies the accuracy of the state transformation. Advanced algorithms such as Gradient Ascent Pulse Engineering (GRAPE) [34] and Chopped

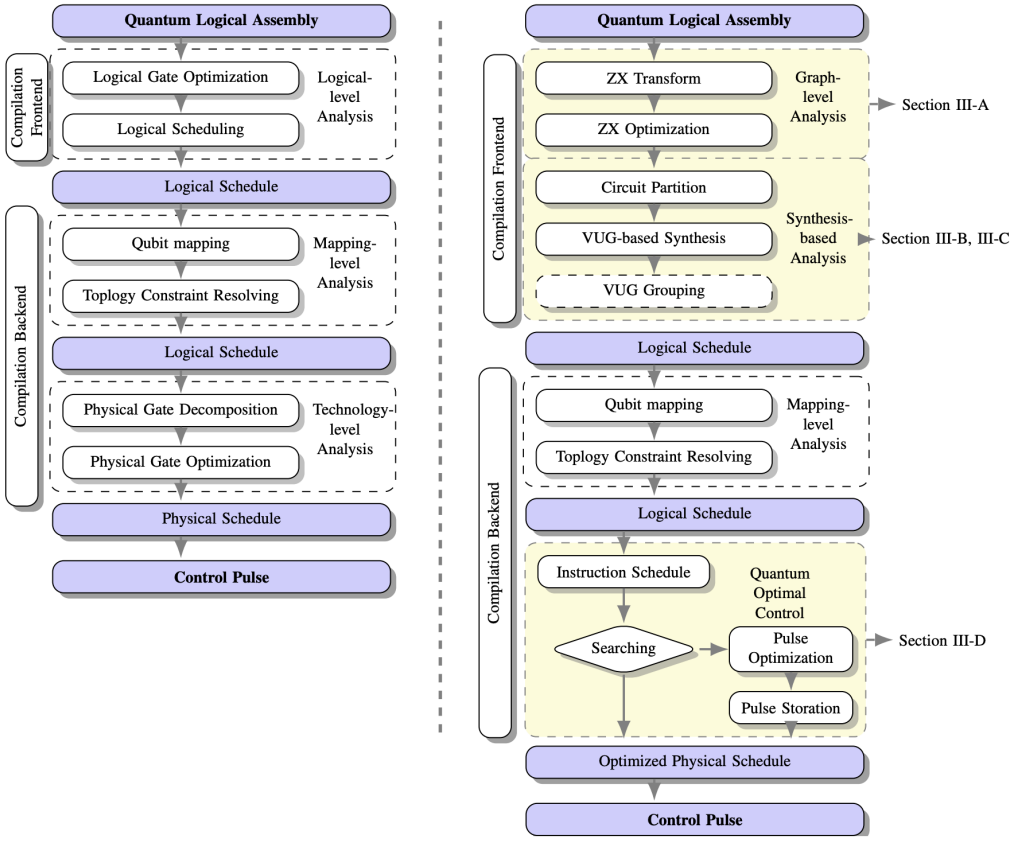


Fig. 3: The comparison between standard gate-based compilation (left) and our approach (right). The colored areas highlight the differences.

Random Basis (CRAB) [35] are utilized to systematically solve the QOC problem. These methods iteratively adjust the control signals to maximize the fidelity, ensuring that the final quantum state closely approximates the target state.

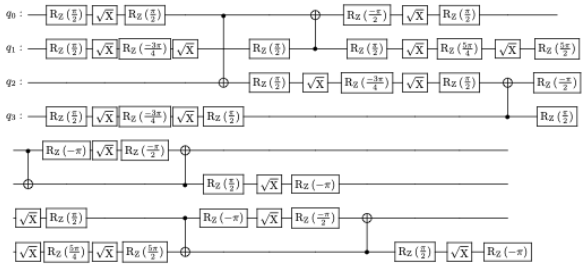
III. Methodology

In this section, we present the methodologies employed in EPOC to accelerate pulse generation. As shown in Figure 3, EPOC consists of several steps, each of which will be discussed in detail throughout this section. We will provide a comprehensive explanation of the techniques and algorithms used in each stage of the EPOC process, highlighting their contributions to improving the efficiency and effectiveness of pulse generation.

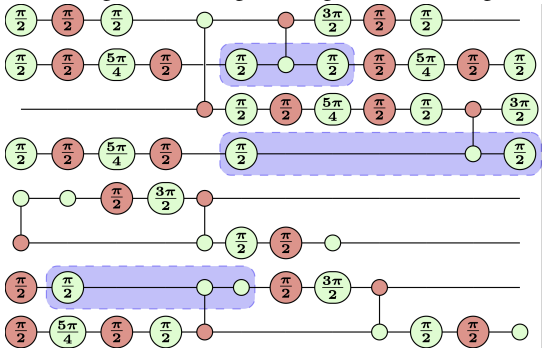
A. Graph-based Depth Optimization

After converting the input circuit into a ZX graph, the gates in the circuit are analyzed to determine if they can be delayed and potentially cancelled against future gates or combined with other gates. This process, known as gate commutation, allows for circuit optimization and depth reduction. For instance, a NOT gate can be placed on either side of a CZ gate for both control and target qubits without altering the gate's functionality. Similarly, a Z gate on the control qubit can be commuted through a CNOT gate if a Z gate also appears on the target qubit, and a NOT gate on the target qubit can be commuted through a CNOT gate if a NOT gate also appears on the control qubit. However, a Hadamard gate cannot be commuted through a CZ gate or a CNOT gate due to its non-commutative properties. Gate commutation enables the aggregation of gates, leading to a reduction in circuit depth. For example, a NOT gate and a Hadamard gate can be combined into a Z gate, while a Z phase gate can be aggregated with a NOT gate, Z phase gate, or T gate by adjusting the rotation phase. Additionally, a pattern consisting of a Z phase gate between two Hadamard gates is transformed into a Hadamard gate between two Z phase gates with negative phases to enhance commutativity and aggregation capabilities. Such a process is simplified with ZX graphs, as in a ZX graph, nodes in the same color are commutative and can be combined into a single node. Apart from gate commutation and gate aggregation, other ZX-calculus rules such as spider fusion are applied to simplify the circuit representations and further reduce the complexity of the quantum circuit. In our work, we utilize PyZX [36], a Python library designed for simplifying ZX-diagrams and optimizing quantum circuits.

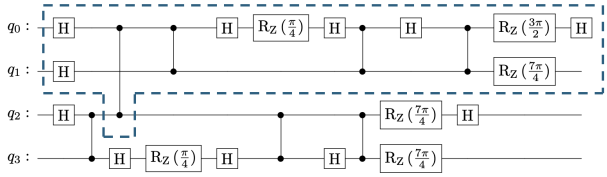
Figure 4 demonstrates a concrete example of the transformation and optimization process. In this example, we consider a bell state preparation circuit with four qubits. The first step involves transforming the original circuit into a ZX graph representation, under the rules provided by ZX-calculus. Once the quantum circuit has been represented as a ZX graph, we can employ optimization methods to simplify the circuit. As shown in Figure 4b, the nodes in the blue boxes of the same color are commutative and can be aggregated into a single node. We iteratively search for such patterns in the graph, ultimately resulting in a simplified and shallower quantum circuit. The optimized ZX graph can then be transformed back into a quantum circuit representation, and the circuit depth is reduced from 23 to 14, resulting in a more efficient and compact version of the original bell state preparation circuit. This example highlights the power of using ZX-calculus and graph-based optimization techniques to simplify quantum circuits. Furthermore, we remark that ZX-calculus for optimization is generally universal for most quantum circuits. For 34 randomly selected circuits, an average depth reduction of 1.48 times is observed as indicated in Figure 5. On extreme occasions, the circuit depth of a variational quantum eigensolver (VQE) is reduced from 7656 to 1110.



(a) Quantum Bell State Generation Circuit with 4 qubits. The circuit is implemented using the rotation gate, SX gate, and CNOT gate.



(b) ZX Graph Representation of the quantum circuit. The nodes within the blue box that share the same color are commutative.



(c) Example of circuit grouping for the optimized circuit after applying ZX-calculus optimization.

Fig. 4: Graph-level optimization steps of the Bell state preparation quantum circuit.

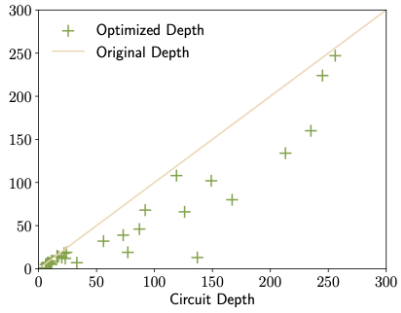


Fig. 5: ZX optimization results for 34 randomly selected quantum circuits. An average depth reduction of 1.48 times is observed across the selected circuits.

B. Greedy Circuit Partition

After processing the circuit using ZX-calculus, we employ a greedy partitioning algorithm to divide it into smaller blocks for synthesis. The algorithm begins with horizontal cutting, grouping qubits together to form initial blocks, followed by vertical cutting to fill each block with as many gates as possible up to a predetermined size limit. By filling the blocks with gates, we ensure each block contains enough operations to benefit from synthesis, with minimal overhead introduced by vertical cutting. The prior ZX-calculus optimization enhances the effectiveness of the greedy partitioning algorithm. ZX-calculus simplifies the circuit by reducing it to a small gate set, resulting in a more uniform structure. This uniformity makes the circuit particularly suitable for the greedy partitioning approach, allowing for a balanced and efficient distribution of gates among the blocks. Figure 4c illustrates the partitioning process, showing a typical circuit block enclosed within a rectangular region after partitioning. These blocks are now ready for the subsequent circuit synthesis process, which will further optimize and transform them into a more compact representation. By combining ZX-calculus optimization with greedy circuit partitioning, we effectively decompose large quantum circuits into smaller, more manageable blocks.

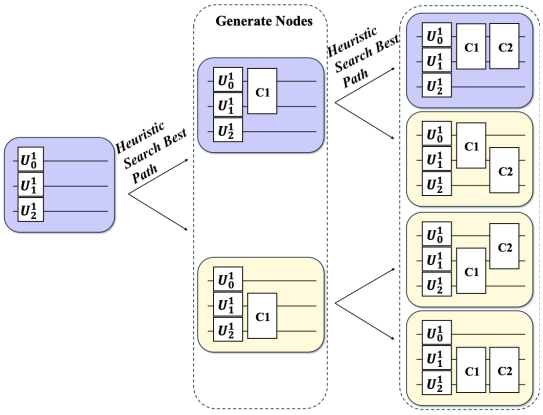
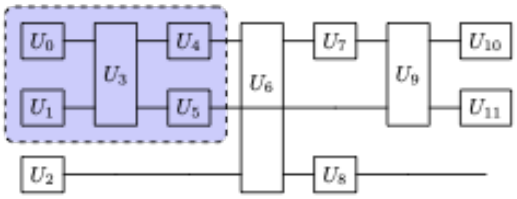


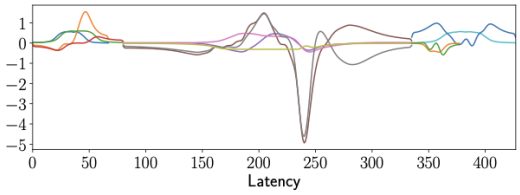
Fig. 6: Illustration of the quantum circuit synthesis process. The synthesis begins with an empty circuit and iteratively generates new nodes by inserting gates at different positions. Each node represents a candidate quantum circuit. The search for the optimal circuit follows the A algorithm, which evaluates nodes based on a cost function that measures the distance between the target unitary matrix and the unitary matrix of the synthesized circuit. The search continues until a circuit that accurately approximates the target unitary matrix with minimal size is found. The resulting synthesized circuit consists of variable unitary gates (VUGs) and CNOT gates.

C. Circuit Size Optimization: Balancing Synthesis and QOC

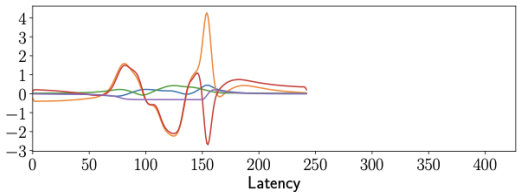
After the greedy circuit partitioning, we obtain a list of grouped gates. We then calculate the unitary matrices of these grouped gates. Since we limit the size of the groups, the computational overhead here is negligible and totally acceptable. In fact, we note that the overhead to compute the unitary matrices remains manageable even with very deep circuit blocks, as long as the number of qubits is moderate. Once we have the unitary matrices of circuit blocks, we leverage insights from QSearch [37] to develop our VUG-based circuit synthesis approach within the BQSKIT [38]. Our synthesis process generates a quantum circuit composed of VUGs and CNOT gates, as illustrated in Figure 6. The synthesized circuit shown in Figure 7a achieves a depth reduction from 11 to 7 compared to the original circuit in Figure 4c. However, a key observation emerges: while traditional synthesis approaches aim to minimize circuit size, this may not be optimal when combining synthesis with QOC. Our analysis reveals that fine-grained VUGs limit the potential benefits of QOC. This insight led us to develop a strategic regrouping approach that creates optimally-sized unitary matrices for QOC processing. Figures 7b and 7c demonstrate the significant impact of this approach: direct pulse generation from individual VUGs results in higher circuit



(a) The synthesis result of the partitioned circuit in the blue box of Fig 4c.



(b) Illustration of the Pulse generation for each gate within the purple box in Fig 7a.



(c) Pulse generation for the entire purple box in Fig 7a, treating all gates as a single grouped unitary.

Fig. 7: The comparison between circuit synthesis + QOC before regrouping strategy and after implement regrouping stratey.

latency, while our regrouping strategy (purple box in Figure 7c) substantially reduces pulse latency. This finding emphasizes the importance of identifying the sweet spot in circuit size that maximizes the combined benefits of synthesis and QOC.

D. Pulse Generation

In our proposed framework EPOC, our methodologies are built upon prior works AccQOC [18] and PAQOC [19]. We employ QOC techniques to generate a pulse library, which essentially serves as a lookup table. The indices of the table are unitary matrices, and each entry stores the corresponding microwave pulses. The techniques in AccQOC and PAQOC involve a binary search to determine a short pulse latency for a given unitary matrix as input. Specifically, the QOC we employ is implemented using the GRAPE algorithm [34], where pulses are divided into segments or time slots with constant values, forming a pulse sequence. The GRAPE method optimizes the time slots according to a cost function, often defined as the fidelity between the pulse unitary and the target unitary, adjusting pulse strengths during each time slot based on gradients. However, pulse simulation and optimization are computationally expensive, which limits the scalability of QOC; thus, we limit the size of the unitary matrices input to the QOC algorithm. Once we have obtained the pulse library, we can accelerate future QOC processes by directly fetching pre-existing pulses for unitary matrices already present in the library. During this process, we also consider the global phase when comparing unitary matrices. In contrast to the previous works on accelerating QOC, EPOC supports the detection of unitary similarity with global phase, allowing us to identify more matched unitary matrices, similar to having a higher cache hit rate.

IV. Evaluation

The experiments were conducted on a Linux-based system with Debian 5.10.179-1 operating system, using 8 nodes. Each node is equipped with 32 cores and 256GB of RAM.

As the first to introduce synthesis into the workflow of QOC, we compare the results from: (1) applying QOC directly on the synthesized circuits and (2) grouping the synthesized results before running QOC. As shown in Figure 8, Figure 9 and Figure 10, we demonstrate although both settings have similar compilation times, our method with an extra grouping step significantly reduces circuit latency and increases circuit fidelity. We evaluate our methods using 17 benchmarks from QASMBench [39], showcasing the effectiveness of incorporating the grouping step in the QOC workflow.

The fidelity is calculated as the accumulated fidelity of each quantum pulse. Specifically, we calculate the fidelity of each pulse and then multiply these fidelities together for all the pulses in the circuit. The fidelity of each individual pulse is determined by the distance between the unitary matrices before and after QOC. This calculation can be described by the following equation:

$$ESP = \prod_{i=1\ldots n} (1 - |U_i - H_i(t)|) \tag{3}$$

where ESP represents the estimated total fidelity of the quantum circuit, n is the number of pulses, U_i is the target unitary matrix for the i-th pulse, and $H_i(t)$ is the unitary matrix achieved by the optimized pulse after applying QOC. The term $|U_i - H_i(t)|$ calculates the distance between the target and optimized unitary matrices for each pulse, and $(1 - |U_i - H_i(t)|)$ represents the fidelity of the i-th pulse. The product of these pulse fidelities gives the overall fidelity of the entire quantum circuit.

Across the 17 benchmarks from QASMBench [39], our method achieves an average reduction of 51.11% in pulse latency with only a 7.11% increase in compilation time by incorporating a regrouping step after synthesis in the QOC workflow. Furthermore, our approach manages to increase the circuit fidelity by an average of 33.77%. These results demonstrate that the regrouping step is necessary and beneficial for reducing circuit latency without introducing significant overhead, highlighting the effectiveness of our proposed method in enhancing the performance of quantum circuits.

We compare our results with both PAQOC [19] and AccQOC [18], two recent works on quantum optimal control. As shown in Table I, our approach achieves significant latency reductions: 31.74% compared to PAQOC and 71.23% compared to AccQOC. Moreover, we consistently report higher fidelity values compared to both approaches - our method achieves an average fidelity of 0.974, surpassing AccQOC's average of 0.890 across all circuits, while also outperforming PAQOC's reported fidelities where data is available. This dual improvement in both latency and fidelity demonstrates the effectiveness of our framework. We attribute these improvements to our novel circuit synthesis process, which effectively optimizes quantum circuits before the QOC step, resulting in reduced latency and increased fidelity. While our synthesis process adds extra computational workload compared to both PAQOC and AccQOC, it mainly involves local entanglement and unitary calculations that can be executed in parallel. Therefore, our approach maintains scalability while significantly reducing pulse latency. We validated our framework by testing it with a large and deep 160-qubit quantum program, obtaining meaningful results. Although this data point isn't in our table due to the lack of comparable data from other methods, it confirms the feasibility and scalability of our approach.

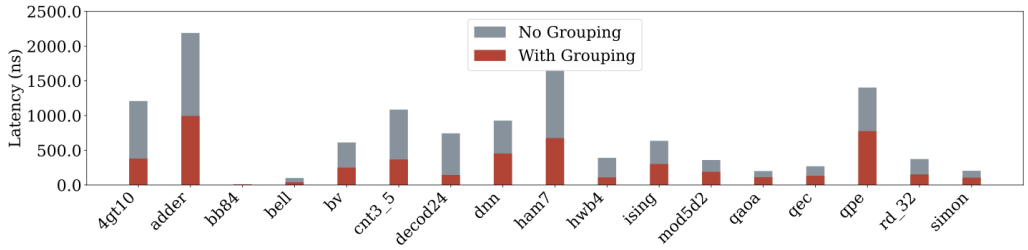


Fig. 8: The latency before and after grouping. We show through the results that the grouping can reduce the circuit latency.

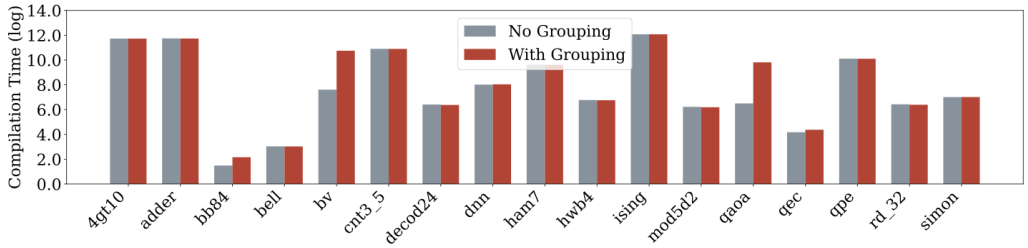


Fig. 9: The compilation time with and without grouping of quantum gates before the final QOC step. This indicates that the latency reduction benefits can be achieved through grouping without incurring a significant increase in compilation time.

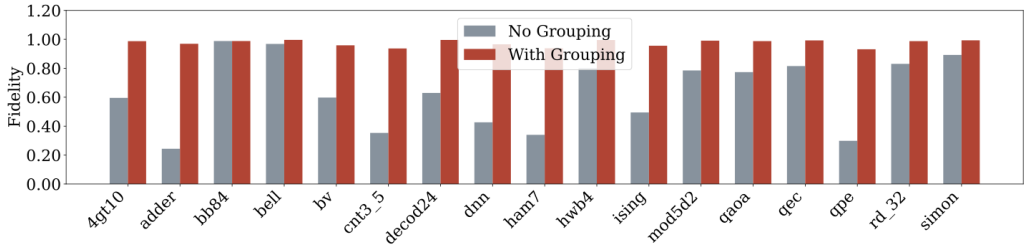


Fig. 10: The fidelity of quantum circuits with and without grouping. The results can be attributed to the fact that the no-grouping approach operates at a very fine granularity, which can lead to the accumulation of errors during the QOC process.

Circuit	Latency (ns)				Fidelity			
	Gate-based	AccQOC [18]	PAQOC [19]	EPOC	AccQOC [18]	PAQOC [19]	EPOC
simon	469	175.22	141.23	**92**	0.91	-	**0.984**
bb84	56.5	24.07	13	**10**	0.659	0.981	**0.988**
bv	901	345.16	321	**268.5**	0.941	0.971	0.968
qaoa	1324.5	370.75	393	**111.5**	0.903	0.952	**0.984**
decod24	1315.5	382.58	315	**144**	0.953	0.982	**0.989**
dnn	3174.5	606.67	385	453.5	0.941	-	**0.965**
ham7	5238.5	1553.09	1186.5	**675.5**	0.920	-	**0.938**

TABLE I: Comparison between AccQOC [18], PAQOC [19], and EPOC (our approach) for 7 widely applied quantum circuits. In most scenarios, EPOC returns a generation result with shortened latency and increased fidelity. On average, our approach achieves a 31.74% reduction in latency compared to PAQOC [19] and a 76.80% reduction compared with the gate-based method to create pulse sequences. The '-' indicates PAQOC returns fidelity as 0 for the corresponding task.

V. Related Work

A recent work AccQOC [18] employs a novel approach by segmenting the quantum circuit into small, uniform subcircuits, each comprising two qubits. It then builds a pulse database for these subcircuits, storing previously generated pulses for efficient reuse. To streamline this process, AccQOC utilizes a similarity graph that maps the distances between different fixed-size subcircuits, including those not currently in the database. The Minimum Spanning Tree (MST) of this graph is then calculated to optimize the order in which control pulses are constructed for the subcircuits. This method significantly enhances the efficiency of pulse compilation, reducing overall time requirements. A more recent work, PAQOC [19], is automatically detecting common gate patterns through subgraph mining for expanded search space exploration. It also systemically constructs customized gate-sets based on their impact on overall program latency and quickly adapts to system recalibrations with its small-scale pattern-based gate generation. However, all the aforementioned compilation strategies generate pulses directly from the partitioned circuit group. Therefore, the group size is constrained due to the unmanageable overhead of pulse generation for large unitaries.

VI. Conclusion

In this paper, we introduced the EPOC framework, integrating ZX-calculus, circuit partitioning, and synthesis techniques with QOC to enhance pulse generation efficiency in quantum circuits. EPOC employs refined granular optimization to significantly reduce circuit latency and computational overhead while increasing parallelism. By strategically regrouping unitary matrices and optimizing their configurations, EPOC improves the scalability of QOC processes and demonstrates considerable performance enhancements in NISQ devices. We achieve notable improvements in pulse generation speed and circuit performance, including an average 31.74% reduction in circuit latency over PAQOC and 71.23% compared to AccQOC. These results indicate significant potential for applying EPOC in practical quantum computing scenarios.

VII. Acknowledgment

ZL's work is supported by Rensselaer Polytechnic Institute. We acknowledge the use of IBM Quantum services for this work.

REFERENCES

[1] J. Zhuang and M. Al Hasan, "Defending graph convolutional networks against dynamic graph perturbations via bayesian self-supervision," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 4, 2022, pp. 4405–4413.

[2] I. N. Levine, D. H. Busch, and H. Shull, *Quantum chemistry*. Pearson Prentice Hall Upper Saddle River, NJ, 2009, vol. 6.

[3] Z. Liang, Z. Wang, J. Yang, L. Yang, Y. Shi, and W. Jiang, "Can noise on qubits be learned in quantum neural network? a case study on quantumflow," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–7.

[4] I. Buluta and F. Nori, "Quantum simulators," *Science*, vol. 326, no. 5949, pp. 108–111, 2009.

[5] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[6] H. Ren and Y. Li, "Modeling quantum devices and the reconstruction of physics in practical systems," *Phys. Rev. Lett.*, vol. 123, p. 140405, Oct 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.123.140405

[7] R. Shaffer, H. Ren, E. Dyrenkova, C. G. Yale, D. S. Lobser, A. D. Burch, M. N. H. Chow, M. C. Revelle, S. M. Clark, and H. Häffner, "Sample-efficient verification of continuously-parameterized quantum gates for small quantum processors," *Quantum*, vol. 7, p. 997, May. [Online]. Available: https://doi.org/10.22331/q-2023-05-04-997

[8] P. Gokhale, O. Angiuli, Y. Ding, K. Gui, T. Tomesh, M. Suchara, M. Martonosi, and F. T. Chong, "Optimization of simultaneous measurement for variational quantum eigensolver applications," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020, pp. 379–390.

[9] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *ASPLOS*, 2019, pp. 1001–1014.

[10] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied physics reviews*, vol. 6, no. 2, p. 021318, 2019.

[11] Z. Liang, J. Cheng, H. Ren, H. Wang, F. Hua, Z. Song, Y. Ding, F. T. Chong, S. Han, X. Qian *et al.*, "Napa: intermediate-level variational native-pulse ansatz for variational quantum algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[12] J. Werschnik and E. Gross, "Quantum optimal control theory," *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 40, no. 18, p. R175, 2007.

[13] C. P. Koch, U. Boscain, T. Calarco, G. Dirr, S. Filipp, S. J. Glaser, R. Kosloff, S. Montangero, T. Schulte-Herbrüggen, D. Sugny, and F. K. Wilhelm, "Quantum optimal control in quantum technologies. strategic report on current status, visions and goals for research in europe," *EPJ Quantum Technology*, vol. 9, no. 1, p. 19, 2022.

[14] M. Bukov, A. G. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, "Reinforcement learning in different phases of quantum control," *Physical Review X*, vol. 8, no. 3, p. 031086, 2018.

[15] N. Leung, M. Abdelhafez, J. Koch, and D. Schuster, "Speedup for quantum optimal control from automatic differentiation based on graphics processing units," *Physical Review A*, vol. 95, no. 4, p. 042318, 2017.

[16] Y. Shi, P. Gokhale, P. Murali, J. M. Baker, C. Duckering, Y. Ding, N. C. Brown, C. Chamberland, A. Javadi-Abhari, A. W. Cross *et al.*, "Resource-efficient quantum computing by breaking abstractions," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1353–1370, 2020.

[17] T. Hao, K. Liu, and S. Tannu, "Enabling high performance debugging for variational quantum algorithms using compressed sensing," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[18] J. Cheng, H. Deng, and X. Qian, "Accqoc: Accelerating quantum optimal control based pulse generation," in *ISCA 2020*. IEEE, 2020.

[19] Y. Chen, Y. Jin, F. Hua, A. Hayes, A. Li, Y. Shi, and E. Z. Zhang, "A pulse generation framework with augmented program-aware basis gates and criticality analysis," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 773–786.

[20] J. van de Wetering, "Zx-calculus for the working quantum computer scientist," *arXiv preprint arXiv:2012.13966*, 2020.

[21] Wikipedia, "Zx-calculus," 2023, [accessed 4-May-2023]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ZX-calculusoldid=1108579483

[22] B. Coecke and R. Duncan, "Interacting quantum observables," in *Automata, Languages and Programming*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 298–310.

[23] ——, "Interacting quantum observables: categorical algebra and diagrammatics," *New Journal of Physics*, vol. 13, no. 4, p. 043016, apr 2011. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/13/4/043016

[24] P. Selinger, "Dagger compact closed categories and completely positive maps," *Electronic Notes in Theoretical computer science*, vol. 170, pp. 139–163, 2007.

[25] J. van de Wetering, "Zx-calculus for the working quantum computer scientist," 2020.

[26] R. Duncan and S. Perdrix, "Rewriting measurement-based quantum computations with generalised flow," in *Automata, Languages and Programming*, S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 285–296.

[27] N. de Beaudrap, R. Duncan, D. Horsman, and S. Perdrix, "Pauli fusion: a computational model to realise quantum transformations from zx terms," *Electronic Proceedings in Theoretical Computer Science*, vol. 318, p. 85–105, May 2020. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.318.6

[28] R. Duncan and M. Lucas, "Verifying the steane code with quantomatic," *Electronic Proceedings in Theoretical Computer Science*, vol. 171, p. 33–49, Dec. 2014. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.171.4

[29] A. Fagan and R. Duncan, "Optimising clifford circuits with quantomatic," *Electronic Proceedings in Theoretical Computer Science*, vol. 287, p. 85–105, Jan. 2019. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.287.5

[30] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum logic circuits," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, 2005, pp. 272–275.

[31] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[32] E. Younis, K. Sen, K. Yelick, and C. Iancu, "Qfast: Conflating search and numerical optimization for scalable quantum circuit synthesis," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 232–243.

[33] E. Smith, M. G. Davis, J. Larson, E. Younis, L. B. Oftelie, W. Lavrijsen, and C. Iancu, "Leap: Scaling numerical optimization based synthesis using an incremental approach," *ACM Transactions on Quantum Computing*, vol. 4, no. 1, pp. 1–23, 2023.

[34] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, "Optimal control of coupled spin dynamics: design of nmr pulse sequences by gradient ascent algorithms," *Journal of magnetic resonance*, vol. 172, no. 2, pp. 296–305, 2005.

[35] T. Caneva, T. Calarco, and S. Montangero, "Chopped random-basis quantum optimization," *Physical Review A*, vol. 84, no. 2, p. 022326, 2011.

[36] A. Kissinger and J. van de Wetering, "Pyzx: Large scale automated diagrammatic reasoning," 2020.

[37] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Towards optimal topology aware quantum circuit synthesis," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2020, pp. 223–234.

[38] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith, and USDOE, "Berkeley quantum synthesis toolkit (bqskit) v1," 4 2021. [Online]. Available: https://www.osti.gov//servlets/purl/1785933

[39] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation," *arXiv preprint arXiv:2005.13018*, 2020.

EDGE: DBMS-Empowered Boolean Decomposition for GIG Synthesis

Ruofei Tang[1], Xuliang Zhu[3], Xinyi Zhang[1], Lei Chen[2], Xing Li[2], Mingxuan Yuan[2], Jianliang Xu[1]
[1] Department of Computer Science, Hong Kong Baptist University, Hong Kong, China
[2] Huawei Noah's Ark Lab, Hong Kong, China
[3] Antai College of Economics and Management, Shanghai Jiao Tong University, Shanghai, China
{csrftang, xinhuang, xujl}@comp.hkbu.edu.hk, zhu.xl@sjtu.edu.cn, {lc.leichen, li.xing3, yuan.mingxuan}@huawei.com

***Abstract*—Boolean decomposition is a powerful technique in logic synthesis that breaks down Boolean functions into simpler components. Decomposition-based logic synthesis yields high-quality results and is particularly effective when combined with small-window optimization methods in Gate-Inverter Graphs (GIG). However, the efficiency limitations of current methods have constrained their applicability in handling large and complex logic. To address this challenge, we propose a novel framework, called *EDGE*, which leverages modern database techniques to accelerate Boolean decomposition, thereby achieving improved synthesis results while maintaining high efficiency. Experimental results demonstrate a runtime speedup of up to 21× and an overall reduction in node count of at least 15% compared to state-of-the-art synthesis methods.**

I. INTRODUCTION

The transformation from a two-level Sum-of-Product (SOP) to a multi-level logic is a key process that affects the area of synthesized circuits. The modern workflow of the transformation process, as shown in Figure 1①, simplifies the input SOP and optimizes it by extracting divisors [1]. From the optimized SOP, a Gate-Inverter Graph (GIG), a data structure used to represent multi-level logic implementations, is generated. A series of small-window optimization methods are subsequently applied to the GIG. This workflow is efficient and generally effective. However, it lacks control over the GIG structure, creating a gap between simpler SOPs and simpler GIGs. Despite some factoring methods intended to enhance logic sharing, these methods are coarse-grained and rely heavily on structural hashing for more localized sharing [2], which is insufficient. Moreover, internal Don't Cares (DCs) are ignored during the transformation and must be recomputed during small-window GIG optimization. To address these issues, Boolean decomposition has been proposed for the transformation [3].

Boolean decomposition is a powerful but time-consuming technique to break down a complex Boolean function into simpler sub-functions. For a given input subset, i.e. the bound set, it extracts a minimal sub-function from the original function, effectively utilizing Don't Cares (DCs) and discovering logic sharing. Figure 1② shows a typical decomposition-based transformation workflow, where the function is repeatedly decomposed until each sub-function is at the gate level, and is converted into a GIG. To improve efficiency, many approaches use single-output disjoint support decomposition (DSD) with binary decision diagrams (BDDs) [3], [4]. BDDs limit the solution space and ignore logic sharing among primary outputs which leads to larger GIGs. Additionally, gate-level decomposition on BDDs is unnecessary, as decomposing small functions is generally less effective compared to SOP-based transformation.

Considering the above limitations, we adopt multi-output Boolean decomposition and propose the transformation flow as shown in Figure 1③. We use multi-output decomposition to break down large functions into small-size sub-functions ($\leq$ 6 inputs) and then synthesize them using the SOP-based workflow. This pipeline allows us to control the overall multi-level structure while achieving effective GIG implementations for each sub-function. To realize this workflow, we face two main challenges:

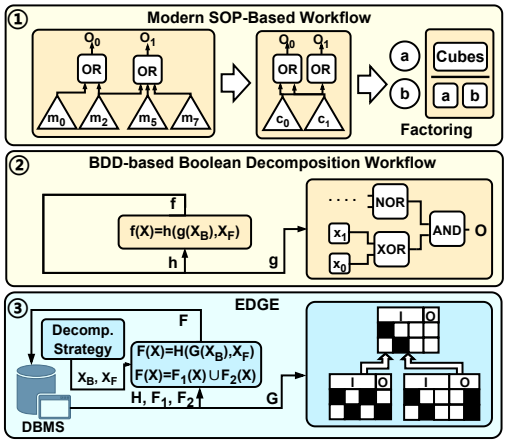


Fig. 1: Comparison of different transformation workflows. ***EDGE*** is our proposed workflow.

1. Multi-output Boolean decomposition faces efficiency challenges due to its NP-hard nature, which limits the size of functions to be processed. Existing solutions use set algebra to form the decomposition process [5], [6], but this approach is difficult to implement and fails to address the runtime problem.
2. The solution space for decomposition grows exponentially with the size of the input set. To the best of our knowledge, there is no framework that can effectively make decisions for a long decomposition chain.

To tackle these challenges, we present the following contributions and results:

- We improve the efficiency of Boolean decomposition using a Database Management System (DBMS). By formulating the decomposition as relational query operations, we optimize its time complexity and leverage various DBMS techniques to accelerate the process.
- We propose a novel Bool**E**an decomposition-based **D**BMS-accelerated **G**IG synth**E**sis framework, ***EDGE***, to manage the continuous decision process of decompositions and schedule the synthesis of each sub-function.
- We propose multiple decomposition strategies for different types of functions. To implement these strategies, we design two heuristic output clustering algorithms for parallel decomposition and introduce an efficient bound set search method for filtering high-quality decompositions.
- We conduct extensive evaluations on ***EDGE***. The results demon-

strate that our framework achieves up to 21× speedup in multi-output decomposition compared to existing methods, while reducing the total GIG size by at least 15% compared to state-of-the-art (SOTA) SOP-based transformation methods.

II. PRELIMINARIES

A. Definitions

Boolean Functions. A Boolean function maps a set of Boolean variables to a single Boolean output, denoted as $y = f(X)$, where y is the *output variable*, f is the *Boolean function*, and $X = \{x_0, x_1, \dots\}$ is the set of *input variables*. When multiple Boolean functions share the same input set X, they can be represented as $Y = F(X)$, where $Y = \{y_0, y_1, \dots\}$ is the set of output variables and $F = \{f_0, f_1, \dots\}$ is the corresponding set of Boolean functions. In this case, F is called a *multi-output Boolean function*. For the remainder of this paper, we use "Boolean functions" to refer to both single-output and multi-output Boolean functions.

Truth Tables. A truth table records the input-output relationships of Boolean functions in a tabular form. A specific combination of input or output values is called a *pattern*. In logic synthesis, due to the presence of *don't cares* (DCs) in designs, a truth table may omit some input patterns, resulting in an *incompletely specified* truth table. **Sum of Products** (SOP) is a compact representation of a truth table with DC values in input and output combinations. Each row in the SOP is called a cube.

Boolean Network. A Boolean network is a directed acyclic graph (DAG) representing Boolean functions [7]. A **Gate-Inverter Graph** (GIG) is a type of Boolean network. A GIG consists of a single type of logic component, and each node has a fixed in-degree (typically 2). Logic NOT gates can be placed on the edges. Common types of GIGs include AND-Inverter Graphs (AIGs), XOR-Inverter Graphs (XIGs), and Majority-Inverter Graphs (MIGs) [8].

Boolean Decomposition. Boolean decomposition is a technique used to break down a Boolean function into simpler sub-functions. Given a Boolean function $Y = F(X)$, there are two types of Boolean decomposition operations [5]:

- *Serial Decomposition.* A serial decomposition extracts a sub-function from a subset of input variables. The decomposition $Y = H(G(X_b), X_f) = F(X)$ is a serial decomposition of F, where $X = X_b \cup X_f$, X_b is the *bound set*, X_f is the *free set*. H and G are sub-functions of Y, and the set $X_s = X_b \cap X_f$ is called the *shared set*. If $X_s = \emptyset$, the decomposition is a *disjoint support decomposition* (DSD). G can be a multi-output Boolean function, and its number of output variables is denoted as $|G|$. For a bound set X_b and a free set X_f in the same serial decomposition, we refer to (X_b, X_f) as a *bound-free pair*.
- *Parallel Decomposition.* A parallel decomposition splits the output set into new sub-functions. If $Y = Y_0 \cup Y_1 \cup \dots$ is a disjoint split of Y, where $Y_i \cap Y_j = \emptyset$ for any $i \neq j$, then $Y_0 = F_0(X), Y_1 = F_1(X), \dots$ is a parallel decomposition of Y, where $F_0, F_1, \dots$ are sub-functions of F.

Relational Algebra Operators. The relational algebra operators used in this paper are listed below [9]:

- σ_θ (select): filtering rows in a relation based on the specified condition θ.
- $\times$ (cartesian-product): producing a relation that is the combination of every tuple in two relations.
- Π_A (project): extracting the specified column set A from a relation, with duplicates removed.
- $\bowtie_\theta$ (join): combining tuples from two relations based on a matching condition θ.

B. Related Works

The inspiration for this work primarily comes from multi-output Boolean decomposition on SOPs [5], [6] and support-reducing decomposition synthesis [3], [4].

In the studies on multi-output decomposition, two basic operations were defined for SOPs: serial decomposition and parallel decomposition [5], [6]. To unify operations on DCs and 1s and 0s, complex set systems were used to build the serial decomposition process, making it difficult to implement and resulting in inefficiency. However, in practice, simplified SOPs do not always have significantly fewer rows than truth tables. Thus, for ease of implementation and optimization of efficiency, we adopt truth tables instead of SOPs. Additionally, no prior work provided a decomposition strategy suitable for modern-scale functions, which motivated us to design ***EDGE***.

In the study on BDD-based DSD for GIG small-window optimization [3], the potential of Boolean decomposition to discover good GIG structures was demonstrated. However, limited by BDD, an inefficient and complex process for logic sharing was required, restricting the window size to 30 nodes and 8 inputs, which reduced the overall effectiveness. In another study [4], the same technique was applied on a larger scale, focusing on reducing the LUT count after mapping rather than optimizing the GIG area. In the process, small-window optimizations were combined to improve the quality of results, but single-output decompositions ignore logic sharing among primary outputs, limiting the effectiveness.

Based on the strengths and addressing the weaknesses of these existing works, we aim to accelerate multi-output Boolean decomposition and extend its applicability to 16 inputs, thereby achieving a better structure on a larger scale and reducing overlap with small-window optimization for improved results.

This work is also influenced by other decomposition-related algorithms, including LUT network optimization [10]–[12] and decomposition method acceleration [13]–[15].

III. *EDGE*

EDGE is a novel framework that utilizes Boolean decomposition for logic synthesis, integrating a DBMS [9] to facilitate large-scale multi-output Boolean decompositions. However, as the scale of processed Boolean functions increases, the solution space grows exponentially, posing substantial challenges to scalability improvements. Therefore, a framework is necessary to evaluate solutions, reduce the search space, and manage the synthesis process. The architecture of ***EDGE***, as depicted in Figure 2, is specifically designed to meet these requirements. Below, we outline the workflow of ***EDGE*** and provide detailed explanations of each component involved.

Workflow. The input truth table is initialized as a Truth Table Unit, which then becomes the root unit node of the Decision Tree, denoted as U_0, and is recorded in the Edge Node Pool. The Decomposition Manager starts working by extracting U_0 from the pool and finding potentially valuable decompositions for the truth table. The selected operations are applied to the unit to generate sub-functions and update the Decision Tree.

The sub-functions are then initialized as new Truth Table Units. If a sub-function's input set size is below a predefined threshold, the unit is marked as terminal and passed to an external GIG synthesis tool [16] for implementation. Non-terminal nodes are added to the Edge Node Pool, and the process repeats until the pool is empty and all edge nodes in the Decision Tree are terminal.

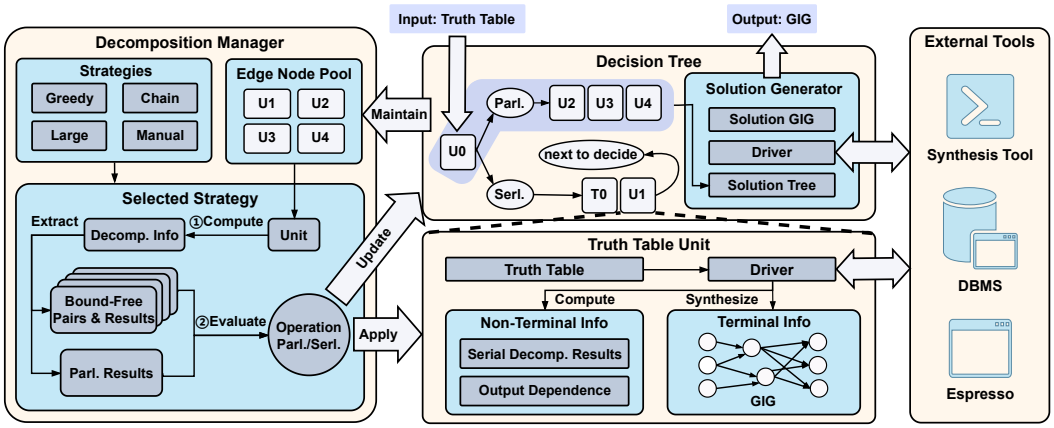


Fig. 2: Architecture of ***EDGE***. Parl. and Serl. are short for parallel and serial decomposition operation nodes. **T** for terminal nodes, **U** for truth table unit nodes.

The Decision Tree can record multiple decomposition-based synthesis solutions, each corresponding to a subtree known as a Solution Tree. By using Solution Trees, the GIGs of sub-functions can be composed, with small-window optimizations performed via the external synthesis tool, resulting in the final GIG output.

Truth Table Unit. The Truth Table Unit serves as the foundational structure in ***EDGE***. It stores the truth table and provides interfaces for decomposition operations and GIG implementation. The truth table is managed within the DBMS, which greatly reduces memory usage and accelerates decompositions. Espresso [17] is used to compute each output's support set for clustering during parallel decomposition.

A unit is marked as terminal when its truth table size is below a specified threshold. This approach is based on the observation that combinations of GIG small-window optimization methods can achieve near-optimal implementation for truth tables with up to 6 inputs in a very short time.

Decision Tree. The Decision Tree serves two main purposes: tracking the synthesis process and generating the final GIG output. Each truth table unit is considered a unit node in the tree.

For a function, there are many possible decomposition paths; the Decomposition Manager strategically selects a small subset of these paths to minimize the solution space. After each decomposition, multiple sub-functions are produced, and the Decision Tree records this entire process.

A sub-tree of the Decision Tree, called the Solution Tree, tracks a complete decomposition-based synthesis process. By traversing the operation nodes of a solution tree in the reverse topological order, the solution GIG is composed. Finally, the GIG implementation of the root node is obtained. Each solution tree yields a different root GIG, and the optimal one is selected as the final output.

Decomposition Manager. The Decomposition Manager determines the decomposition type and details (bound set, free set, or output split) for the nodes from the Edge Node Pool.

Different types of Boolean functions require different strategies to ensure high-quality decomposition. The strategy begins by testing a set of potential decomposition results. These results are then evaluated to determine the final operations. Multiple algorithms are proposed to support the strategies. Details are provided in Section IV.

IV. TECHNIQUES

A. Database-Accelerated Serial Decomposition

Serial decomposition is the core algorithm in ***EDGE*** and is executed many times within a transformation process, thus requiring high efficiency. However, it involves solving an NP-hard problem, and the demand for accuracy makes it hard to approximate the process. Existing works rely on complex set systems, which are challenging to implement and introduce unnecessary computation. To tackle these problems, we use relational algebra query operators to build the algorithm. The algorithm is divided into two stages: Prepare and Commit. For a serial decomposition $Y = F(X) = H(G(X_b), X_f)$, the Prepare phase computes the G sub-function, and the Commit phase computes the H sub-function.

Prepare Phase. The Prepare phase is illustrated in Algorithm 1. Before explaining the process, we first introduce several key concepts and a lemma:

- **Bound / Free / Output Pattern.** The pattern represents a specific set of Boolean variable assignments for the bound set, free set, and output set, respectively.
- **Conflict Pair.** A conflict pair is defined as a tuple of two bound patterns, $\langle X_{bi}, X_{bj} \rangle$, derived from two rows in a truth table, denoted as r_0 and r_1. If r_0 and r_1 have the same free pattern but different output patterns, then the bound patterns of r_0 and r_1 are said to be in *conflict*, forming a conflict pair.

Lemma 1 *Given a serial decomposition $F(X) = H(G(X_b), X_f)$, the decomposition is valid if and only if, for every conflict pair $\langle X_{bi}, X_{bj} \rangle$, it holds that $G(X_{bi}) \neq G(X_{bj})$.*

Proof *Assume $F(X) = H(G(X_b), X_f)$ is a valid decomposition, where there exists at least one conflict pair $\langle X_{bi}, X_{bj} \rangle$ such that $G(X_{bi}) = G(X_{bj})$. By the definition of a conflict pair, X_{bi} and X_{bj} have the same free pattern, i.e., $X_{fi} = X_{fj}$. Since $G(X_{bi}) = G(X_{bj})$ and $X_{fi} = X_{fj}$, it follows that: $H(G(X_{bi}), X_{fi}) = H(G(X_{bj}), X_{fj})$. This contradicts our definition of the conflict pair. Thus, our initial assumption must be false, proving Lemma 1.*

Lemma 1 provides the condition for a valid serial decomposition. Thus, the first task in Algorithm 1 is to identify conflict pairs (line 1). According to the definition of conflict pairs, we can write the following relational algebraic expression to find them:

$$C \leftarrow \Pi_{F_1.Bound, F_2.Bound}\big(\sigma_{F_1.Free=F_2.Free} \big(\sigma_{F_1.Output \neq F_2.Output}(F_1 \times F_2)\big)\big) \tag{1}$$

where F_1, F_2 are aliases of F, and $Bound, Free, Output$ are short for bound, free and output patterns. This is an equivalent method to the SOP-based one in the previous work [6]. Given X as the input set, this query has a time complexity of $O((2^{|X|})^2)$. Although existing

Algorithm 1 Serial Decomposition Phase Prepare

Input: Truth Table F, Bound Set X_b, Free Set X_f
Output: Truth Table G
1: Conflict Pairs $C \leftarrow \mathsf{LookupConflictPairs}(F, X_b, X_F)$;
2: Graph $\mathcal{G}_C \leftarrow (\Pi_{X_b}(F), C)$; ▷ $\mathcal{G}_C$: (Nodes, Edges)
3: Map $M \leftarrow \mathcal{G}_C.\mathsf{Coloring}()$; ▷ M : Bound Pattern → Color ID
4: Map $P \leftarrow \mathsf{AssignPattern}(M)$; ▷ P : Color ID → New Pattern
5: $G \leftarrow M \bowtie P$; ▷ M joins P on Color ID
6: **return** G;

Algorithm 2 Bound-Free Set Pair Search

Input: Truth Table F, Flag $allow_share$, Search Range $[l, u]$
Output: Map M ▷ M: bound set size → truth table array
1: Let X, Y be the input and output variable set of F;
2: Set $C \leftarrow \bigcap_{y \in Y} \mathsf{Supports}(y)$;
3: Integer $\langle lower,\ upper \rangle \leftarrow \langle l, \mathsf{Min}(u, |C| - 1) \rangle$;
4: **for** integer $k \in [lower,\ upper]$ **do**
5: **for** set $X_b \in \mathsf{Combinations}(C, k)$ **do**
6: Truth Table $G \leftarrow \mathsf{SerialPrepare}(F, X_b, X \setminus X_b)$;
7: $M[k] \leftarrow M[k] \cup \{\langle G, X_b, X \setminus X_b \rangle\}$;
8: $M[k].\mathsf{sort}()$;
9: **if** $allow_share$ **then**
10: $\mathsf{ExtendSharedSet}(M)$;
11: **return** M;

methods are applied to cubes, they still rely on expanding DCs to identify conflict pairs, resulting in the same time complexity.

To improve efficiency, we propose the following optimizations:

1. **Join Rewriting.** By adjusting the selection order, we can rewrite the cartesian-product into a conditional join, which greatly reduces the operation cost.
2. **Symmetry Elimination.** Due to the symmetry in conflict relations and output pattern comparison, we can add constraints to the inner join and replace $\neq$ with $<$ in the selection, which halves the rows to scan.

After these revisions, we refine Equation (1) as follows:

$$C \leftarrow \Pi_{F_1.Bound, F_2.Bound}\big(\big(\sigma_{F_1.Output < F_2.Output} (F_1 \bowtie_{F_1.Free = F_2.Free \wedge F_1.Bound < F_2.Bound} F_2)\big)\big) \tag{2}$$

With these optimizations, the time complexity is reduced to $O(2^{\frac{2|X_b|+|X_f|}{2}})$. Additionally, executing the query in a DBMS provides several benefits [9]:

- **Sort-merge join.** For $F_1 \bowtie F_2$, the DBMS would use a sort-merge join rather than a double full table scan.
- **Multi-thread acceleration.** The DBMS splits the query operation into multiple independent subtasks and accelerates them through multithreading.
- **Storage optimizations.** These techniques are optimized for handling table data, reducing memory usage, improving storage efficiency, and speeding up queries.

After finding conflict pairs in Algorithm 1, we construct a valid minimal sub-function using graph coloring (lines 2-3). Each node represents a unique bound pattern, and each edge represents a conflict. After coloring, each color corresponds to a unique input pattern in the new function G. We assign each color a distinct pattern and construct G based on the coloring results (lines 4-5). Since graph coloring has near-linear time solutions under edge constraints [18], it is not a significant factor in efficiency.

Commit Phase. In the Commit Phase, we compute the sub-function H according to the sub-function G. Since G is a mapping from the X_b as a subset from X, then to obtain H, all we need to do is a natural join of G and F on X_b, denoted as $H = G \bowtie F$, which is a very fast process.

B. Search of Bound-Free Pair

In Section IV-A, we improve the efficiency of serial decomposition. However, for a truth table with $|X|$ inputs, there can be $O(2^{|X|})$ possible bound-free pairs, even for DSDs. To avoid the overhead of exhaustive search, we present Algorithm 2.

The search is conducted using the bound set length as the unit. In line 1, we find the intersection of output supports, as including output-irrelevant inputs is meaningless. In lines 4-7, we iteratively test the results of serial decomposition on bound-free pairs within the search range and store the result tuples in M. In line 8, we sort the results based on two metrics of G within the same bound set size: (1) the number of outputs, and (2) the number of unique output patterns. We retain only the top-K tuples in M, where K is a hyperparameter. In lines 9-10, an optional step allows for extending the free set to form shared sets.

Shared sets can enhance connections between sub-functions, benefiting GIG optimization. However, searching for suitable shared sets is costly, so it is provided as an optional action, executed only when time permits. The heuristic algorithm for extending the free set of a DSD is as follows:

S1. Denote the result tuple to extend as $\langle G, X_b, X_f \rangle$. Initialize a set $S = \{\{x\} \mid x \in X_b\}$.
S2. Pop X_S from S, and get $G_S \leftarrow \mathsf{SerialPrepare}(F, X_b, X_f \cup X_S)$. If $|G_S| = 1$ or $|G_S| - |X_S| \leq |G|$, update $T \leftarrow T \cup \{X_S \cup \{x\} \mid x \in X_b \setminus X_S\}$. Keep a record of the largest $X_S \in S$ as X_L.
S3. Repeat Step 2 until $S = \emptyset$. Return $X_L \cup X_f$.

Note that when the bound set size approaches the input size, the extension of free sets is less effective.

C. Heuristic Clustering of Parallel Decomposition

Parallel decomposition separates outputs with weak logical connections when serial decomposition struggles to find meaningful bound-free pairs, enabling the synthesis process to continue. We design two methods for parallel decomposition. The first method clusters output variables based on their support sets, and if this fails, we apply a second method that clusters outputs by unique pattern numbers.

By Support Set. When the support sets of a group of output variables differ significantly, serial decomposition will not yield meaningful results. To address this, we cluster the outputs based on their support sets. The Jaccard distance is used to measure the distance between two outputs:

$$D(y_i, y_j) = 1 - \frac{|\mathsf{Supports}(y_i) \cap \mathsf{Supports}(y_j)|}{|\mathsf{Supports}(y_i) \cup \mathsf{Supports}(y_j)|} \tag{3}$$

where $y_i,\ y_j \in Y$, $D(y_i, y_j)$ is the distance between y_i and y_j. Hierarchical clustering with average linkage is then used to adaptively determine the number of clusters.

By Unique Pattern. The process for this method is shown in Algorithm 3. The function $\mathsf{COUNT}(query)$ returns the number of rows in the query result. The goal is to minimize unique output patterns in the two sub-functions.

D. Decomposition Strategies

Decomposition strategies determine the appropriate decomposition for each input truth table after search and evaluation. To handle functions with various characteristics, we propose three automatic

Algorithm 3 Unique Pattern Based Parallel Decomposition

Input: Truth Table F
Output: Cluster C_0, C_1
1: Let Y be the output variable set of F;
2: **for** $y \in Y$ **do**
3: **if** $\frac{\mathsf{COUNT}(\Pi_{C_0 \cup \{y\}}(F))}{\mathsf{COUNT}(\Pi_{C_0}(F))+1} \leq \frac{\mathsf{COUNT}(\Pi_{C_1 \cup \{y\}}(F))}{\mathsf{COUNT}(\Pi_{C_1}(F))+1}$ **then**
4: $C_0 \leftarrow C_0 \cup \{y\}$;
5: **else**
6: $C_1 \leftarrow C_1 \cup \{y\}$;
7: **return** C_0, C_1;

Algorithm 4 Greedy Strategy

Input: Truth Table F, Termination Size t, Search Range r
Output: Operations Op
1: $F_R \leftarrow \mathsf{RemoveTrivialOutputs}(F)$;
2: Let X, Y be the input and output variable set of F_R;
3: **if** $|X| \leq t$ **then** ▷ Termination condition
4: $Op \leftarrow \mathsf{Terminate}$;
5: **return** Op;
6: $Results \leftarrow \mathsf{SerialPairSearch}(F, True, [|X|-r, |X|-1], k)$;
7: $Results \leftarrow Results \cup \mathsf{SerialPairSearch}(F, True, [2, 2+r], k)$;
8: **if** $|Y| = 1$ **then** ▷ Single-output situation
9: $Op \leftarrow \mathsf{EvaluateSingle}(Results)$;
10: **if** $Op = \emptyset$ **then**
11: $Op \leftarrow \mathsf{EarlyTerminate}$;
12: **else if** $|Y| > 1$ **then** ▷ Multi-output situation
13: $Op \leftarrow \mathsf{EvaluateMultiple}(Results)$;
14: **if** $Op = \emptyset$ **then**
15: $Op \leftarrow \mathsf{HeuristicParallel}(F)$;
16: **return** Op;

strategies: Greedy, Large, and Chain. Additionally, we provide a **Manual** mode for debugging and developing new strategies.

Greedy. The greedy strategy is designed to handle general cases. It prioritizes support-reducing serial decomposition, with parallel decomposition applied only if no good bound-free pair is found. The process is shown in Algorithm 4.

In line 1, RemoveTrivialOutputs preprocesses F by removing output variables whose support set contains 2 or fewer inputs, as these are easy to implement and could interfere with finding meaningful bound-free pairs. In lines 2-4, we check if the termination condition is met. If so, the Terminate operation is returned, and the truth table is passed to the GIG synthesis tool. In lines 6-7, bound-free pairs are searched within the specified range from both sides. Lines 8-15 evaluate single-output and multi-output Boolean functions using different criteria to decide the final operation. Elements of $Results$ are denoted as $\langle G = Y_G(X_b), X_b, X_f \rangle$ for subsequent use.

For single-output Boolean functions, the results of serial decompositions are first filtered based on the following conditions:

- **Not Support-reducing:** $|G| = |X_b|$.
- **No Logic Sharing:** $\mathsf{COUNT}(\Pi_{Y_G}(G)) = 2^{|X_b|}$.

The second condition comes from Shannon Decomposition:

$$f(X, x_i) = x_i \cdot f(X, 1) + \overline{x_i} \cdot f(X, 0) \tag{4}$$

which represents the worst-case result of a serial decomposition without any logic sharing, refined here to the pattern level.

If no valid tuple remains after filtering, EarlyTerminate is returned for F even if $|X| > t$. For the remaining decompositions, the following criteria are used to determine the best option, in order of priority:

1. **Is Single-Output:** $|G| = 1$.
2. **Estimated Area Sum:** $|G| \cdot 2^{|X_b|} + |F_R| \cdot 2^{|G|+|X_f|}$.

Algorithm 5 Chain Strategy

Input: Chained Truth Table F_C
Output: Operation Op
1: Let X, Y be the input and output variable set of F_C;
2: $y_0 \leftarrow \arg\min_{y \in Y} |\mathsf{Supports}(y)|$;
3: **if** $|\mathsf{Supports}(y_0)| \leq 2$ **then**
4: $Op \leftarrow \mathsf{ParallelByGroup}(Y \setminus \{y_0\},\ \{y_0\})$;
5: **else**
6: $Op \leftarrow \mathsf{SerialPrepare}(F, \mathsf{Supports}(y_0), X \setminus \mathsf{Supports}(y_0))$;
7: **return** Op;

3. **Bound Set Scale:** $\mathsf{COUNT}(\Pi_{Y_G}(G))$.

For multi-output decompositions, similar steps are followed, but with differences in filtering conditions and evaluation criteria:

- **No Logic Sharing:** $|G| = \sum_i |G_i| \wedge \mathsf{COUNT}(\Pi_{Y_G}(G)) = 2^{|G|}$, where $G_i = \mathsf{SerialPrepare}(\Pi_{X, \{y_i\}}(F), X_b, X_f)$, $y_i \in Y$.

If no valid tuple remains after filtering, a parallel decomposition is selected as the operation on F. The evaluation criteria are similar to the single-output case, but **Is Single-Output** and **Estimated Area Sum** are modified as follows:

- **Is Single-Output:** $|G| \leq |F|$.
- **Estimated Area Sum:** $\frac{|G|}{\sum_i |G_i| - |G|} \cdot 2^{|X_b|} + |F_R| \cdot 2^{|G|+|X_f|}$.

Here, an extra bias is introduced to prioritize logic sharing.

Large. The Large strategy is a variant of the greedy strategy, designed to reduce the runtime for large truth tables by avoiding searches of large bound sets. Adjustments include:

1. Bound-free pairs are searched only from the lower side instead of both sides (line 7 in Algorithm 4).
2. Pairs are searched step by step rather than over a range. Once a feasible serial decomposition is found, it is returned.
3. If the bound set size becomes large (≥ 12), EarlyTerminate or HeuristicParallel is selected as the operation.

These modifications may impact result quality, and users can choose the strategy according to the specific scenario.

Chain. The Chain strategy is designed for a special type of functions where $\mathsf{Supports}(y_0) \subseteq \mathsf{Supports}(y_1) \subseteq \mathsf{Supports}(y_2) \subseteq \cdots$ with all outputs logically related. The optimal structure of such functions forms a chain. In practice, these functions are identified based on their output support sets. The process of the chain strategy is shown in Algorithm 5. There is no need to search for pairs, as the optimal solution is obvious. Once the bottom-level output becomes trivial, it is separated from the function.

V. EXPERIMENTS

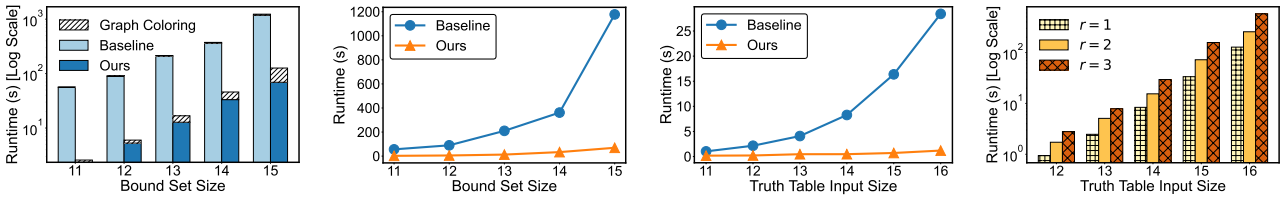
EDGE is implemented in Python, using *DuckDB* [19] as the DBMS and *NetworkX* [20] for graph algorithms support. All experiments are conducted on a server with an AMD EPYC 7742 CPU and on IWLS 2023 contest benchmarks.[1]

A. Efficiency of Serial Decompositions

This subsection evaluates the efficiency of our serial decomposition implementation under various settings. The original method based on Equation (1) could not complete within 3,600 seconds in our experiments. Consequently, we compare our approach exclusively with the complexity-reduced method based on Equation (2) without DBMS acceleration, which we will refer to as the baseline.

EXP-1: Comparison on 16-Input Truth Tables. In this experiment, we compare the total runtime of serial decomposition for 16-input

[1] https://www.iwls.org/iwls2023/2023_IWLS_Contest_Benchmarks.zip



(a) 16-Input Total Time (b) 16-Input Conflict Pairs Search (c) Varying-Input Total Time (d) Bound-Free Pair Search

Fig. 3: Efficiency of Serial Decomposition

truth tables with different bound set sizes. In Figure 3(a), "Baseline" and "Ours" represent the runtime of the conflict pairs search component in the baseline method and our approach, respectively, while "Graph Coloring" denotes the runtime associated with graph coloring. As can be seen, the pairs search component dominates the overall runtime. Our proposed approach achieves a reduction in the runtime of this component by 94-95% compared to the baseline method across different bound set sizes, as shown in Figure 3(b). Consequently, the total runtime using our approach is 10-21× better than that of the baseline method.

EXP-2: Comparison for Fixed Bound Set Size. In this experiment, we compare the runtime for different input sizes with a fixed bound set size of **10**. Figure 3(c) shows the runtime trend as the input size increases. While the initial improvement in runtime is 10× at an input size of 11, this enhancement grows exponentially as the input size increases, proving the scalability of our optimizations.

EXP-3: Evaluation of Bound-Free Pair Search. This experiment evaluates the efficiency of bound-free pair search with varying truth table sizes and different bound-free pair search ranges. We use 32 parallel threads to accelerate the search, without applying the Large strategy. Figure 3(d) shows that for a truth table with 16 inputs, the total runtime with $r = 3$ is the longest, reaching 576 seconds, while with $r = 1$, it takes only 127 seconds. This difference arises from two factors: (1) the number of bound-free pairs within the range, and (2) the runtime of decompositions for different bound set sizes. For a 12-input truth table, reducing 4 inputs results in a runtime of only 2.7 seconds for $r = 3$. This highlights the rationale behind our Large strategy: to quickly reduce truth table size while focusing on decomposition quality for smaller functions.

B. *Effectiveness of* ***EDGE***

EXP-4: Comparison of Synthesis Results. In this experiment, we evaluate and compare the GIG synthesis results of ***EDGE*** against SOTA SOP-based transformation processes. All input truth tables are considered random, with no specific optimizations for any instance. The termination size is set to 6 for truth tables with more than 8 inputs and 4 otherwise. The benchmarks are grouped based on the official information from the IWLS 2023 contest, and all synthesis results are verified by comparing the simulated truth tables. The quality of the synthesis is measured by the number of AIG nodes. We compare three methods implemented in ABC [16]. The sequence used for all methods includes reading truth tables, simplifying SOPs, and converting to AIGs: *read_truth -xf; collapse; sop;* [factorization command]*; strash;*. Original does not use a factorization command, while `fx` and `fxch` use *fx* and *fxch* commands, respectively.

Table I shows the node counts of AIGs after transformation. For each dataset, ***EDGE*** produces fewer initial AIG nodes than other methods, achieving an 18% reduction in the total node count compared to the best SOTA (`fx`) results. Although having fewer initial AIG nodes does not necessarily mean fewer nodes after optimization, a smaller initial size reduces the runtime of powerful but time-consuming optimization methods [3], [21]. The average runtime for ***EDGE*** per truth table is **230** seconds. Notably, for truth tables with 14 or more inputs and multiple outputs, the primary source of runtime is the Espresso computation for output supports.

TABLE I: Number of Nodes After Transformation

Benchmarks	Original	`fx`	`fxch`	***EDGE***
ex00-ex01	1,018	736	795	**654**
ex02-ex07	4,626	3,229	3,494	**3,074**
ex08-ex09	4,384	2,660	2,817	**2,452**
ex10-ex15	5,978	3,833	3,902	**3,523**
ex16-ex27	20,171	13,722	14,266	**11,469**
ex28-ex49	19,569	13,199	14,424	**12,351**
ex50-ex67	37,561	23,322	25,211	**16,526**
ex68-ex99	18,310	12,521	13,510	**9,986**
Total	111,617	73,222	78,419	**60,035**

TABLE II: Number of Nodes After GIG Optimization

Benchmarks	Original	`fx`	`fxch`	***EDGE***
ex00-ex01	704	669	704	**602**
ex02-ex07	3,408	2,959	3,147	**2,878**
ex08-ex09	2,958	2,474	2,574	**2,364**
ex10-ex15	4,122	3,443	3,485	**3,162**
ex16-ex27	13,984	12,571	13,007	**10,787**
ex28-ex49	13,341	11,893	12,658	**11,662**
ex50-ex67	25,704	21,313	22,791	**15,739**
ex68-ex99	12,489	10,936	11,649	**9,040**
Total	76,710	66,258	70,015	**56,234**

Table II shows the node counts of AIGs after transformation and optimization using a sequence of instructions: *fraig; resyn; resyn2; resyn2a; resyn3; drw; dc2; refactor;*, repeated 50 times to approach the optimization limit. Our method outperforms all SOTA methods across all datasets, achieving a 15% reduction in total node count compared to the best SOTA (`fx`) results. In 36 specific test cases, our method achieves more than a 10% reduction compared to the best SOTA results.

VI. CONCLUSION

In this work, we propose a two-level logic transformation process based on multi-output Boolean decomposition to enhance the quality of GIG synthesis. To improve efficiency, we have formulated the process as relational query operations and leverage DBMS techniques to accelerate computations and reduce redundancy. For better synthesis quality, we have proposed a decomposition framework, ***EDGE***, with tailored strategies to handle truth tables of varying characteristics. Experimental results demonstrate that ***EDGE*** achieves significant improvements in decomposition efficiency and higher-quality synthesis results compared to SOTA methods.

Acknowledgment. This work is partially supported by the Hong Kong Research Grants Council and Guangdong Basic and Applied Basic Research Foundation (Project No. 2023B1515130002). Lei Chen is the corresponding author.

References

[1] B. d. O. Schmitt, A. Mishchenko, V. N. Kravets, R. K. Brayton, and A. I. Reis, "Fast-extract with cube hashing," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 145–150.

[2] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "Fraigs: A unifying representation for logic synthesis and verification," ERL Technical Report, Tech. Rep., 2005.

[3] L. Machado and J. Cortadella, "Boolean decomposition for aig optimization," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 143–148.

[4] ——, "Support-reducing decomposition for fpga mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 213–224, 2018.

[5] T. Łuba and H. Selvaraj, "A general approach to boolean function decomposition and its application in fpgabased synthesis," *VLSI Design*, vol. 3, no. 3-4, pp. 289–300, 1995.

[6] J. A. Brzozowski and T. Luba, "Decomposition of boolean functions specified by cubes," *Journal of Multiple Valued Logic and Soft Computing*, vol. 9, no. 4, pp. 377–418, 2003.

[7] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 263–268.

[8] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2015.

[9] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. McGraw-Hill Education, 2019.

[10] A. Mishchenko, R. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks," in *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2008, pp. 38–44.

[11] A. T. Calvino, G. De Micheli, A. Mishchenko, and R. Brayton, "Enhancing delay-driven lut mapping with boolean decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[12] A. Mishchenko, R. Brayton, A. Tempia Calvino, and G. De Micheli, "Boolean decomposition revisited," in *International Logic Synthesis Workshop (IWLS)*, 2023.

[13] A. Mishchenko, B. Steinbach, and M. Perkowski, "An algorithm for bi-decomposition of logic functions," in *Proceedings of the 38th annual design automation conference*, 2001, pp. 103–108.

[14] M. Choudhury and K. Mohanram, "Bi-decomposition of large boolean functions using blocking edge graphs," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2010, pp. 586–591.

[15] V. Callegaro, F. S. Marranghello, M. G. Martins, R. P. Ribas, and A. I. Reis, "Bottom-up disjoint-support decomposition based on cofactor and boolean difference analysis," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*. IEEE, 2015, pp. 680–687.

[16] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.

[17] R. Brayton, *Logic Minimization Algorithms for VLSI Synthesis*. Springer, 1984.

[18] T. Mostafaie, F. M. Khiyabani, and N. J. Navimipour, "A systematic study on meta-heuristic approaches for solving the graph coloring problem," *Computers & Operations Research*, vol. 120, p. 104850, 2020.

[19] M. Raasveldt and H. Mühleisen, "Duckdb: an embeddable analytical database," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1981–1984.

[20] A. Hagberg, P. J. Swart, and D. A. Schult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[21] F.-X. Reichl, F. Slivovsky, and S. Szeider, "Circuit minimization with qbf-based exact synthesis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 4087–4094.

Enabling Data-Deduplication-Assisted Data Relocation for Interlaced Magnetic Recording

Chen-Jui Tu
National Yang Ming Chiao Tung University
Taiwan
Email: raytu.tpe@gmail.com

Shuo-Han Chen*
National Yang Ming Chiao Tung University
Taiwan
Email: shch@nycu.edu.tw

***Abstract*—Interlaced Magnetic Recording (IMR) drives have been regarded as promising hard disk drives (HDDs) to meet the ever-growing storage demand in the post-AI era. Among various technologies, IMR drives achieve increased storage capacity by overlapping tracks in an interlaced fashion; however, updating data on overlapped tracks necessitates rewriting up to two tracks, which can degrade performance. Previous strategies have attempted to reduce track rewrites by either delaying the timing of overlapping based on storage space usage or relocating frequently updated data to non-overlapped tracks through update-frequency-based hot/cold data separation. In contrast, this paper proposes a novel approach that leverages data deduplication as a natural solution to the rewrite challenge in IMR drives. Unlike conventional hot/cold separation, this paper utilizes deduplication metadata, specifically reference counts, as a direct indicator for data relocation. The rationale behind the proposed approach is that deduplicated data, which remains static unless reference counts drop to one, can be stored or migrated to overlapped tracks without inducing future track rewrites. The proposed approach diverges from traditional frequency tracking methods by leveraging data deduplication information to guide data placement and reduce rewriting. Evaluation results show that the proposed scheme effectively reduces the accumulated read/write latency by 37.57% on average compared with state-of-the-art data management on IMR drives with data deduplication enabled.**

***Index Terms*—IMR, deduplication, relocation, HDD**

I. Introduction

Owing to the explosive growth of digital data in the post-AI era, both hardware and software designs have been explored to either enhance the storage density or reduce the data volume without significantly changing existing file and storage systems. Among various hardware solutions, hard disk drives (HDDs) are favored for their low cost-per-gigabyte, high reliability, and long lifespan. While the superparamagnetic effect (SPE) [1] limits the storage density of existing conventional magnetic recording (CMR) HDDs, various new technologies, such as the shingled magnetic recording (SMR) [2–4] and interlaced magnetic recording (IMR) [5–7], have been widely studied. Although SMR has been commercialized [8], the shingled track layout leads to an excessive number of data rewrites under random workload and greatly limits the random write performance. Thus, SMR becomes less preferable and issues related to unpredictability of I/O performance have also been reported [3, 4, 9]. Recently, the advances of interlaced magnetic recording (IMR) technology have outperformed SMR and enables even-higher storage density with alleviated track rewrites.

Figure 1 compares CMR, SMR and IMR technologies. Unlike SMR that has multiple tracks overlapped on top of previous tracks, IMR organizes *top tracks* and *bottom tracks* in an interlaced fashion and each top track overlaps at most two adjacent bottom tracks. IMR achieves higher linear density by applying higher laser current to the wide bottom tracks and has narrow top tracks placed closer for higher areal density [10]. Such an interlaced layout enables top tracks to be updated freely and induces at most two track rewrites when updating bottom tracks if adjacent top tracks contain valid data. As a result, IMR is considered a more attractive alternative over SMR with the lowered number of track rewrites. To further alleviate the overhead when updating bottom tracks on IMR drives, space allocation strategies, including the rewrite-minimized data placement [11], the three-phase data management [12], and the Z-Alloc [13], are proposed to postpone the allocation of top track until all bottom tracks are in use. After top tracks are allocated, another perspective is to reduce updates on bottom tracks by either buffering updates on free top tracks, such as the top buffer [13] and selective track buffering [14], or relocating frequently-updated bottom tracks to non-overlapped top tracks. Examples of data relocation include the block swap [13], the track flipping [14], and the data hotness aware movement [15]. Nevertheless, *most existing mechanisms employ similar approaches to swap or relocate data blocks based on update frequency for hot/cold data separation.*

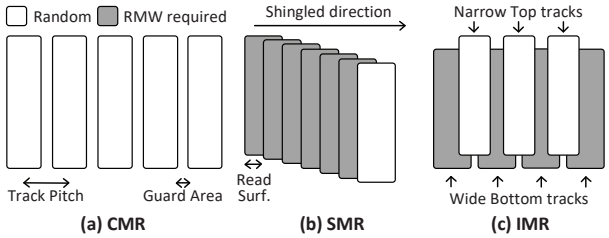


Fig. 1. Comparison of CMR, SMR, and IMR, in which the track pitch and the guard area indicate the distance between data tracks and the distance to avoid interference. For RMW-required tracks, read surfaces are reserved and exposed for the disk read head to retrieve stored data.

Examples of software solutions for enlarging usable storage space include the data compression and data deduplication techniques [16]. While data compression encodes the digital data using fewer bits, data deduplication eliminates duplicate copies of repeated data. Although both techniques can effectively lower data size, data compression has higher performance overhead since encoding and decoding are required during data writes and reads, respectively. In addition, owing to the decoding overhead, the *readahead*[1] [17] mechanism in existing file systems may induce additional decoding overhead if the access pattern prediction is incorrect. In other words, data deduplication is preferable with read performance consideration.

This work was supported in part by the National Science and Technology Council, Taiwan, under grant nos. 113-2221-E-A49-189-MY3, 111-2221-E-A49-198-MY3, and 113-2221-E-007-134.

*Corresponding author

[1]By predicting the access pattern, the readahead mechanism prefetches file data into main memory to reduce the latency of future access.

Data deduplication identifies and eliminates redundant copies of data, retaining a single instance at block- or chunk-level by (1) calculating the *cryptographically secure hash-based fingerprints* [16], (2) comparing fingerprints to identify duplication, (3) storing only one copy of the duplicate data, (4) maintaining mapping information between the stored data and those deduped chunks or files. Even though data deduplication techniques have been studied for over 20 years [16] and machine learning deduplication has also been proposed [18, 19], *the potential of leveraging data deduplication information to facilitate the IMR management received much less attention.*

Notably, instead of utilizing data deduplication information for tracking update frequency and separating hot/cold data, this paper is constructed based on the fact that *deduplicated data chunks*[2] *remain unchanged unless the reference count drops to one.* Specifically, deduplicated data chunks do not receive updates directly. Instead, updates are applied to new or unique chunks. If the reference count of a duplicated chunk drops to one, suggesting it is no longer shared, it may then become eligible for modifications or updates. This paper introduces a *deduplication-assisted data relocation* (DADR) scheme to reduce track rewrites by utilizing data deduplication information, specifically the reference count of each block or chunk, enabling effective data relocation and space allocation. Integrating data deduplication onto IMR drives requires sharing deduplication information between the host system (e.g., file systems) and the disk firmware (e.g., Interlaced Translation Layer (ITL) [14]). This is because, in the proposed scheme, fingerprint calculation is assumed to be conducted at the host system, as IMR drives are typically resource-constrained. The deduplication information is then maintained within ITL, based on details passed down from the host system as a *hint* via the ATA command set [20, 21].

The DADR scheme is threefold. First, after valid top tracks start overlapping bottom tracks, non-deduped bottom tracks are swapped with deduped top tracks during bottom track updates to reduce the number of track rewrites. Meanwhile, during track rewrites, deduped top tracks are actively relocated to invalid or non-deduped bottom tracks. Second, after fingerprint calculation is performed on the host system, the proposed DADR scheme actively relocates non-deduped and deduped data writes to top and bottom tracks. Notably, since the size of deduped blocks or chunks (i.e., 4 KB) is typically smaller than a IMR tracks (i.e., 2 MB), non-deduped and deduped data chunks are packed into non-deduped and deduped data tracks, respectively. Third, the DADR scheme delays the allocation of top tracks that are on top of non-deduped bottom tracks. Above measures are effective since deduped data chunks are static as long as the reference count is larger than 1, thus allowing the proposed scheme to effectively lower track rewrites on IMR drives. Another benefit of integrating data deduplication onto IMR drives is lowering the total amount of written data, thus further reducing the overall latency during data writes. The main contribution of this study lies in *leveraging the deduplication information to facilitate the IMR management, reducing track rewrites and enhancing performance.* Evaluation results show that the number of track rewrites and the accumulated latency are reduced by 41.34% and 37.75% on average compared with the state-of-the-art ITL [13] with data deduplication, Z-alloc, block swap, and top buffer enabled.

[2]Deduplicated data chunks refer to data chunks that have undergone the data deduplication process.

II. Background and Motivation

A. Interlaced Magnetic Recording (IMR)

Interlaced Magnetic Recording (IMR) technology achieves higher storage density by organizing data tracks in an interlaced layout, as shown in Figure 1(c). By placing narrower top tracks on at most two wider bottom tracks, IMR is capable of reading data from bottom tracks through the residual track surface. Nevertheless, due to the track overlap, writing data on bottom tracks destroys valid data on adjacent top tracks. To restore those destroyed data, the *read-modify-write (RMW)* operation is introduced. As illustrated in Figure 2(a), before $Track$ 1 is updated, data on $Tracks$ 8 and 9 are first read out. Then, after updates to $Track$ 1 are already merged, $Tracks$ 1, 8 and 9 are then written back. Inevitably, additional write traffic is induced due to track rewrites during the RMW operations and becomes performance overhead. To lower the amount of write traffic and the number of track rewrites, existing strategies can be categorized into 3 groups, including space allocation, track relocation, and track buffering [12–14]. Notably, these strategies are typically implemented in the interlaced translation layer (ITL) to manage IMR drives with a *logical-to-physical (L2P) mapping table*. For instance, based on the ratio of space utilization, the Z-alloc [13] has three phases of allocation, and the allocation direction is reversed when phases switch.

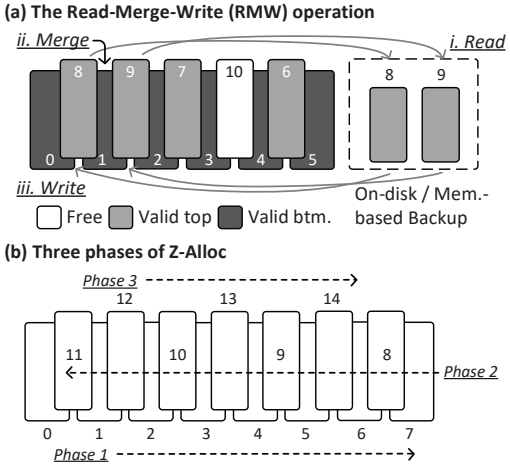


Fig. 2. Read-merge-write operation and Z-alloc of IMR drives, in which track indexes are labeled according to their allocation sequence.

As shown in Figure 2(b), the $Phase$ 1 of Z-alloc only allocates bottom tracks. Then, the $Phase$ 2 reverses the direction and only allocates every other top tracks. Finally, the $Phase$ 3 reverses the direction again and allocates those residual top tracks. Although Z-alloc can effectively delay the timing of allocating top tracks, *the possibility of storing frequently-updated data onto bottom tracks remains.* To relocate those frequently-updated data, data relocation, such as the block swap [13] and the data hotness aware movement [15], exchanges frequently-updated data on bottom tracks with infrequently-updated data on top tracks. In other words, update frequency tracking is required for hot/cold data separation. Nevertheless, *update frequency could change overtime according to locality and induce additional relocation overhead when the locality*

shifts. Lastly, an example of track buffering is the top buffering [13], which reserves a number of top tracks for buffering bottom track updates to delay or avoid the track swap. Even though previous strategies can effectively reduce the number of track rewrites on IMR drives, *the possibility of further reducing the amount of written data or facilitating IMR data management through data deduplication has received much less attention.*

B. Data Deduplication

Since 2000s, data deduplication techniques have been commercialized and widely adopted by cloud storage providers. For instance, Microsoft and EMC report that roughly 50% and 85% of data in their storage systems are duplicated [16]. The process of data deduplication can be summarized as Figure 3. First, data duplication are identified at either *block-* or *chunk-level*. For chunk-level deduplication, incoming data steams are divided into smaller data chunks via either fixed-size chunking (FSC) or content-defined chunking (CDC), where CDC can achieve higher deduplication ratio than FSC. Second, each data chunk is labeled with a fingerprint generated by hash functions, such as MD5 and SHA-256, and are compared with other fingerprints to check whether the data chunk is duplicate or not. The process of comparison is known as *fingerprint indexing*, which can be accelerated by Bloom filters [16].

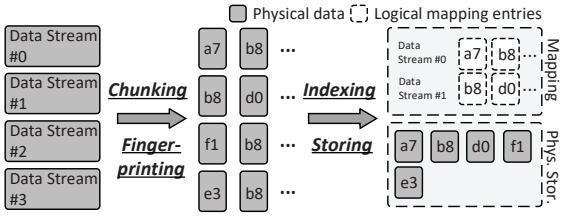


Fig. 3. The process of chunk-level deduplication, in which data streams go through the chunking, fingerprinting, indexing, and storing steps for data deduplication.

Finally, mapping entries are maintained between the stored physical copy of the data chunk and the logically deduplicated data chunks or files. In this study, data deduplication mapping entries are assumed to be maintained within the ITL after the host system conducts deduplication and passes the results through the reserved field in the ATA command set. This paper reveals that the reference count can be repurposed to indicate whether a data chunk will be updated. Notably, rather than using the reference count for access frequency prediction, data with a reference count greater than 1 remains static and receives no updates unless the reference count drops to 1. According to the investigation of this study, *such an indication from data deduplication techniques has not been explored to facilitate the data management of IMR drives.*

C. Motivation

This section provides an illustrative example to explain the existing inefficiency of the state-of-the-art space allocation and data relocation strategies, namely Z-alloc and block swap, on IMR drives. As shown in Figure 2(b), existing Z-alloc divides the track allocation into 3 phases based on the ratio of space utilization. However, as shown in Figure 4(a), in $Phase$ 1, all data, including frequently-updated data, can only be written to bottom tracks. If subsequent updates to the frequently-updated data happen during either $Phase$ 2 or $Phase$ 3, track rewrites or block swaps become inevitable and induce performance overhead, as shown in Figure 4(b). Such an unwanted overhead can be avoided if data that will not be updated, namely dedupled data chunks with reference count larger than 1, can be identified and proactively stored on bottom tracks. Consequently, the adjacent top tracks of bottom tracks containing deduplicated data chunks can be allocated for data that could be updated directly in $Phase$ 1.

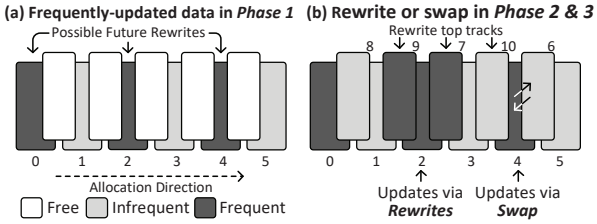


Fig. 4. Motivational example, in which frequently-updated data are written on bottom tracks in the Phase 1 of Z-alloc and induce track rewrites or swaps in Phase 2 and 3.

To relocate frequently- (*i.e.,* hot) and infrequently-updated (*i.e.,* cold) data, the existing block swap strategy needs to attain the hot and cold data information on the block basis. Typically, such hot/cold information is obtained from keeping the access count of each data block. Other strategies, such as Least Recently Used (LRU), can be also utilized; however, both the access count and LRU mechanism could be affected by temporal locality, leading to frequent block swaps under thrashing scenarios. The reference count in data deduplication techniques, on the other hand, provides a clear indication of whether a stored data chunk may be updated—specifically, stored data chunks with a reference count greater than 1 are static and receive no updates unless the reference count drops to 1. It would be beneficial if deduplication information could substitute for access count or LRU mechanisms, as this approach would reduce the influence of temporal locality while identifying blocks that remain static.

III. Deduplication-Assisted Data Relocation (DADR) Scheme

A. Overview

To facilitate the data management of IMR drives, this paper presents a deduplication-assisted data relocation (DADR) scheme with the goal of lowering the number of track rewrites by leveraging the data deduplication information. Figure 5 illustrates the system architecture, showing how deduplication metadata facilitates the DADR scheme in managing IMR drives. In the considered system architecture, the generation of deduplication fingerprint is conducted at the host system for the better processing capability. Then, deduplication index, the logical address, and the original data are passed through conventional write requests of ATA command. Under this implementation, a two-level mapping table is presented to keep track of both the L2P mapping entries of the original ITL and the data deduplication information. Logical addresses of stored data chunks are mapped to deduped entries based on the entry indices, and each deduped entry records the fingerprint, the reference count, and the physical address. As deduplication information is shared between the host system and the proposed scheme, deduped entries are cached onto the host system and periodically flush are conducted to ensure consistency. Alternatively, if a specialized

controllers or special intellectual property (IP) cores can be integrated into the IMR drives for fingerprint generation, the fingerprint could be generated within the ITL, eliminating the need to pass it down from the host system.

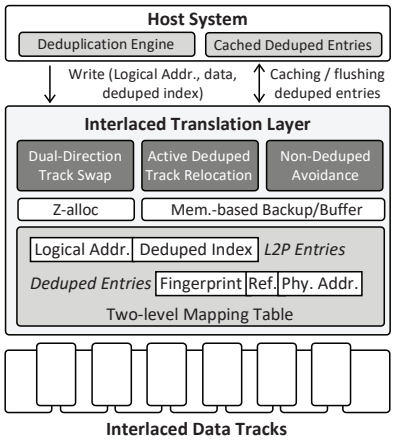


Fig. 5. The system architecture of DADR scheme, in which Z-alloc is utilized with the aids of proposed components.

Based on the two-level mapping table, the main components of the DADR scheme are illustrated below. First, the *dual-direction track swap* is proposed to swap to-be-updated bottom tracks with top tracks that have the largest deduped reference count. The main rationale behind the dedupe-based track swap is that the only physical copy of deduped data chunks remains static if it is referenced by more than one logical data chunk. In other words, this physical copy remains unchanged even when one of its logical data chunk is being updated, which either remaps to another deduped physical data chunk or triggers copy-on-write (COW) to create another new physical data chunk. Second, based on the state-of-the-art allocation method (*i.e.,* Z-alloc), the *active deduped track relocation* is included to actively relocate non-deduped and deduped data to the top and bottom tracks, respectively, during the $Phase$ 1 of IMR track allocation. Lastly, the *non-deduped avoidance* is included during $Phases$ 2 and 3 to skip allocation of those top tracks with their adjacent bottom tracks storing non-deduped data chunks (*i.e.,* reference count equals to 1), and introduces $Phase$ 4 to utilize those residual top tracks after $Phase$ 3. In summary, this paper presents a pioneering design that first integrates data deduplication on IMR drives and leverage deduplication information to facilitate data management.

B. Dual-Direction Track Swap

To avoid frequent track rewrites, previous studies [13, 15] have suggested that swapping hot bottom tracks with cold top tracks is an effective action after valid top tracks start overlapping valid bottom tracks. While most of the previous designs rely on the update frequency of each data track for identify hot/cold data, the proposed DADR scheme includes the dual-direction track swap to utilize the reference count of data deduplication mechanism for track swaps. Accordingly, the to-be-updated bottom tracks with the reference count of 1 are going to be swapped with the top track that has the highest reference count. In an extreme case where all top tracks have a reference count of 1, the RMW operation will be triggered to update the bottom tracks and preserve the overlapped valid top tracks. Nevertheless, according to our experimental results, such an extreme case has not been observed with real-world workloads. An example is given in Figure 6 to better illustrate the above procedure. In this example, an update request is made to $Track$ 2, and the overlapping $Track$ 9 and $Track$ 7 are buffered to prevent data loss. Data on $Track$ 2 are swapped with the top track with the highest reference count, which is the $Track$ 9. Subsequently, the original data on $Track$ 9 are relocated to $Track$ 9 and the original data on $Track$ 2 are merged with updates and stored to $Track$ 9.

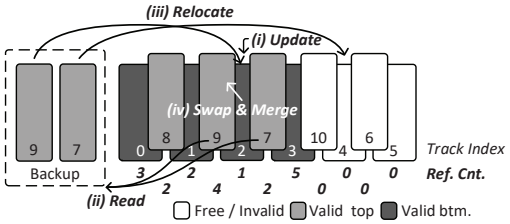


Fig. 6. Dual-direction track swap, which lowers the possibility of future rewrites by swapping and relocating tracks based on the deduped reference count.

Notably, in addition to proactively swapping non-deduped bottom and deduped top tracks, the proposed dual-direction track swap further leverages the deduped reference count to relocate adjacent deduped top tracks of the to-be-updated bottom tracks when the to-be-updated bottom track is swapped with a deduped top tracks. This is because, as shown in Figure 6, overlapping top tracks of to-be-updated bottom tracks are buffered. If those top tracks contains deduped data chunks, it would be beneficial to relocate the buffered top track to an invalid or a non-deduped bottom track since deduped data chunks (i.e., the reference count is larger than 1) remains static. Such an approach allows top tracks to be reserved for data that could possibility be updated. In Figure 6, during the swap of $Track$ 2 and $Track$ 9, the $Track$ 7 with a reference count of 2 is also read out. To fully utilize the track rewrite opportunity and avoid future rewrite overhead, the $Track$ 7 with the reference count of 2 is moved to a free bottom track, namely $Track$ 4, during the updates of $Track$ 2. Then, $Track$ 7 becomes free again for future allocation. Such an approach makes the rewrite overhead worthwhile and fully utilizes the track rewrite opportunity.

C. Active Deduped Track Relocation

Following the Z-alloc method on IMR drives, $Phase$ 1 of Z-alloc could force frequently-updated data to be stored on bottom tracks and leads to track rewrites or block swaps during $Phase$ 2 and $Phase$ 3. In other words, it would be beneficial if infrequently- and frequently-updated data can be redirected to bottom and top tracks during $Phase$ 1 of Z-alloc. In the proposed DADR scheme, the active deduped track relocation, as summarized in Figure 7, is included to relocate data according to their deduped reference count, which can be observed while data are buffered on memory before being stored permanently. Since buffering is widely used on HDDs to improve response time, the active deduped track relocation does not alter the existing data flow. Data with a deduped reference count greater than 1 are stored on bottom tracks, while non-deduped data with a reference count of 1 are proactively stored on the top tracks above them. Top tracks that are on top of bottom tracks containing deduped data chunks are referred to as allocatable top tracks. If allocatable top tracks do not exist, those non-deduped

data are still stored on bottom tracks since writing to top tracks on top of free bottom tracks leads to RMW operations while those free bottom tracks are allocated in the future. Notably, since deduped data chunks are typically smaller than a track of IMR drives, multiple data chunks are stored on the same tracks, which are non-deduped or deduped tracks. That is, data chunks are stored onto data tracks based on whether their reference count is larger than one or not. This is because the proposed scheme is designed based on whether a data block will be updated or not, and the ranking of the reference counts makes no difference.

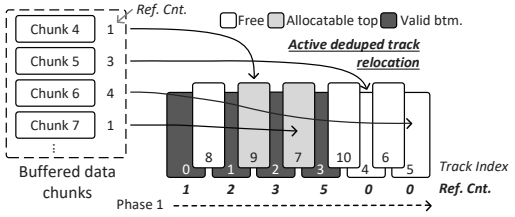


Fig. 7. Active deduped track relocation, which aims to avoid storing non-deduped data chunks onto bottom tracks in $Phase$ 1 of Z-alloc.

D. Non-Deduped Avoidance

To further leverage deduplication information to lower track rewrite overhead on IMR drives, the proposed DADR scheme introduces the non-deduped avoidance into the $Phase$ 2 and $Phase$ 3 of Z-alloc to delay the allocation of top tracks that are on top of bottom tracks containing non-deduped data chunks. Meanwhile, the allocation of those top tracks is delayed until a newly-included $Phase$ 4 after the $Phase$ 3 finishes. Such approach further reduces the possibility of track rewrites since those bottom tracks with non-deduped data chunks can still be freely updated without inducing track rewrites during $Phase$ 2 and $Phase$ 3. Notably, in this study, instead of completely replacing the state-of-the-art space allocation method on IMR drives, the proposed DADR scheme explores the merits of dedupication information to assist the data management of IMR drives and reduce the possibility of track rewrites. In other words, the components of the proposed DADR scheme can be integrated to other allocation methods on IMR drives, while integrating data deduplication to reduce the size of written data.

IV. Performance Evaluation

A. Experiment Setup

In this session, experiments are conducted to evaluate the effectiveness of the proposed DADR scheme in terms of reducing the accumulated read and write latencies, as well as the number of rewritten data tracks. Since the current implementation involves altering the ITL of IMR drives, both the existing Tracklace [13] approach and the proposed DADR scheme are implemented and simulated on an IMR simulator [6]. Notably, the Tracklace includes methodologies such as Z-alloc, block swap, and top buffer approaches. The evaluation uses three types of workloads: two real-world traces from the FSL-Dedup Traces dataset and one synthetic trace with a lower data deduplication ratio. The characteristics of each trace used in the experiment are detailed in Table I.

The two real-world traces, labeled *user000* and *user002*, provide valuable insights into the performance of storage systems under deduplication conditions. In the selection process for these two traces, the main consideration is the peak data chunk sizes and the aim to understand how data chunk size impacts the effectiveness of the proposed scheme. The peak chunk sizes for the two traces are near 0 bytes and approximately 250,000 bytes. Given that FSL-Dedup traces capture a snapshot of a deduplication-enabled storage system, additional data update requests were manually and randomly generated during this evaluation to simulate real-world usage. The synthetic trace, on the other hand, was generated with a normal distribution of data write sizes for each entry, using a mean of 128 KB and a standard deviation of 30 KB. This synthetic trace helps to model a scenario with different deduplication ratios for helping to understand the effectiveness and performance trend of the proposed scheme, thus providing a contrast to real-world traces.

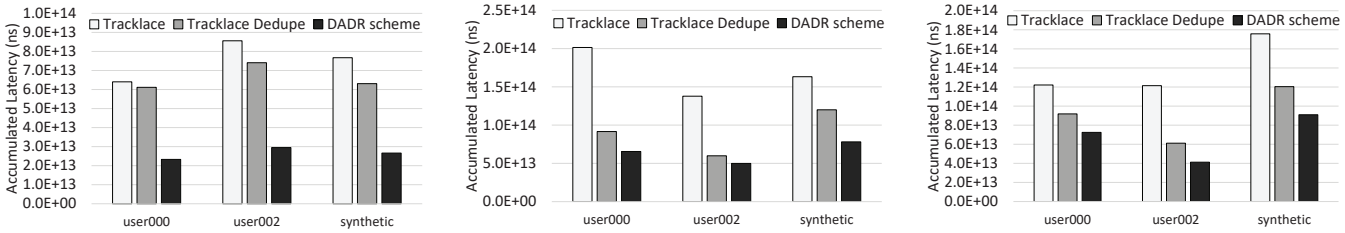
TABLE I
Workload characteristics.

Traces	Write Size (GB)	Dedupe (%)	Update size (GB)
user000	70.8	16.9	44.1
user002	210.9	27.2	128
synthetic	130.8	14.4	98.69

B. Experimental Results

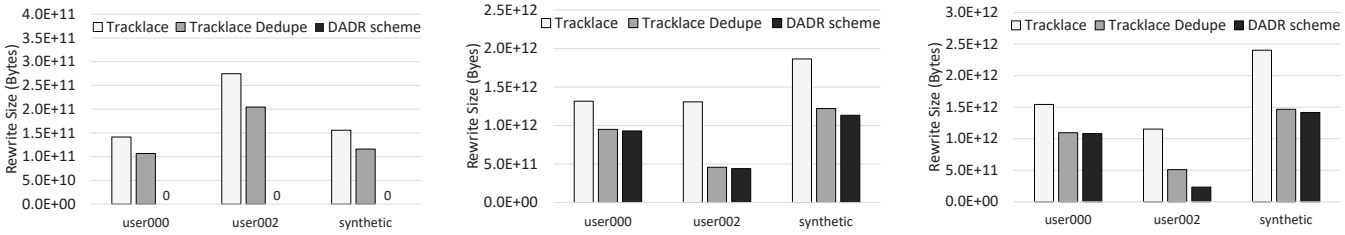
To eliminate the effect of data deduplication and focus on investigating the effectiveness of data management of the DADR scheme, the improvement percentage shown in this paper is derived by comparing the proposed DADR scheme and the Tracklace with data deduplication enabled (*i.e.,* Tracklace Dedupe). The experimental results of Tracklace without enabling data deduplication are also included for completeness. In addition, to understand the capability of the proposed DADR scheme under different space utilization scenarios, three ratios of space utilization are chosen, which are 50%, 80%, and 90%. Accordingly, the accumulated latency comparisons are first shown in Figure 8. When the ratio of space utilization is 50%, the accumulated latency can be reduced by 59.96% on average when compared with Tracklace Dedupe. This is because the DADR scheme can actively relocate data to the bottom and top tracks based on the reference count of the deduplication mechanism to differentiate static data or data that could be possibly updated. Meanwhile, the reason why Tracklace performs worse than Tracklace Dedupe is that Tracklace already enters the Phase 2 of Z-alloc, the top buffer of Tracklace is enforced for every update to bottom tracks, thus inducing additional overhead.

After the ratio of space utilization grows up to 80% and 90%, the proposed scheme achieves latency reductions of 26.68% and 26.06% on average, respectively. The reason for the reduction drop is that the dual-direction block swap of the proposed scheme is triggered to relocate data due to updates to bottom tracks. Nonetheless, the reduction is still positive and up to 32% for user002 at 90% space utilization. On the other hand, as one of the design goals of the DADR scheme is to reduce track rewrites, the accumulated rewrite data size is reported in Figure 9. For 50% space utilization, the DADR scheme induces no track rewrites since the dual-direction block swap is not triggered. On the contrary, Tracklace requires RMW operations when flushing the top buffer to the bottom tracks. After the ratio of space utilization grows to 80% and 90%, the average reductions are 4.93% and 19.63%, respectively. The reason is



(a) 50% space utilization (b) 80% space utilization (c) 90% space utilization

Fig. 8. Comparison of accumulated latency under different space utilization, including the latency of data reads/writes for serving incoming requests and the latency of data rereads/rewrites during block swap for all three methods and top buffer flush, which is enabled for Tracklace and Tracklace Dedupe.



(a) 50% space utilization (b) 80% space utilization (c) 90% space utilization

Fig. 9. Comparison of accumulated rewrite size of block swaps under different space utilization. This comparison is included to show the number of track rewrites between each compared methods.

that, at 80% space utilization, the dual-direction block swap is constantly triggered. Then, at 90%, there are much fewer block swaps as most of the active updated data are relocated to top tracks. In other word, even when the space utilization is at 90%, the included active deduped track relocation is still effective in terms of reducing the amount of data rewrite.

V. Conclusion

To avoid the locality shifting issue of conventional data management on IMR drives, this study introduces a pioneering approach called the deduplication-assisted data relocation (DADR) scheme. The DADR scheme leverages data deduplication information to optimize data management on IMR drives and strategically relocates deduped data to the bottom tracks while moving non-deduped data with a higher likelihood of updates to the top tracks. Notably, instead of tracking updating frequency or trying to reflect updating frequency through the deduplication information, the proposed DADR scheme is designed based on the fact that deduped data remain static unless the reference count drop to 1. Accordingly, the DADR scheme actively redirects data to the bottom and top tracks and delays the allocation of top tracks on non-deduplicated bottom tracks. With the above mechanism, the DADR scheme demonstrates a significant reduction in both track rewrites and accumulated read/write latency. On average, it achieves a 41.34% reduction in track rewrites and a 37.75% reduction in accumulated read/write latency compared to the existing Tracklace approach with data deduplication enabled. By minimizing track rewrites, the DADR scheme presents a promising approach for managing IMR drives data deduplication.

References

[1] A. Amer, J. Holliday, D. D. E. Long, E. L. Miller, J.-F. Paris, and T. Schwarz, "Data management and layout for shingled magnetic recording," *IEEE Transactions on Magnetics*, vol. 47, no. 10, pp. 3691–3697, 2011.

[2] S. Greaves, Y. Kanai, and H. Muraoka, "Shingled recording for 2–3 tbit/in^2," *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3823–3829, 2009.

[3] W. He and D. H. Du, "SMaRT: An approach to shingled magnetic recording translation," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, Santa Clara, CA, Feb. 2017, pp. 121–134.

[4] A. Aghayev and P. Desnoyers, "Skylight—A window on shingled disk operation," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, Santa Clara, CA, Feb. 2015, pp. 135–149.

[5] E. Hwang, J. Park, R. Rauschmayer, and B. Wilson, "Interlaced magnetic recording," *IEEE Transactions on Magnetics*, vol. 53, no. 4, pp. 1–7, 2017.

[6] S.-H. Chen and K.-H. Huang, "Leveraging journaling file system for prompt secure deletion on interlaced recording drives," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 3, pp. 619–634, 2023.

[7] S. Yoon and E. Hwang, "Joint multi-level readback for array-reader-based interlaced magnetic recording," *IEEE Transactions on Magnetics*, vol. 57, no. 3, pp. 1–5, 2021.

[8] W. Digital, "Product brief: Wd red nas hdd - western digital," accessed on 10.22.2023. [Online]. Available: https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/internal-drives/wd-red-hdd/product-brief-western-digital-wd-red-hdd.pdf

[9] Y.-J. Chuang, S.-H. Chen, Y.-H. Chang, Y.-P. Liang, H.-W. Wei, and W.-K. Shih, "Dstl: A demand-based shingled translation layer for enabling adaptive address mapping on smr drives," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 4, jul 2020.

[10] S. Granz, W. Zhu, E. C. S. Seng, U. H. Kan, C. Rea, G. Ju, J.-U. Thiele, T. Rausch, and E. C. Gage, "Heat-assisted interlaced magnetic recording," *IEEE Transactions on Magnetics*, vol. 54, no. 2, pp. 1–4, 2018.

[11] Y. Liang, M.-C. Yang, and S.-H. Chen, "Magic: Making imr-based hdd perform like cmr-based hdd," *IEEE Transactions on Computers*, vol. 71, no. 3, pp. 643–657, 2022.

[12] K. Gao, W. Zhu, and E. Gage, "Write management for interlaced magnetic recording devices," Patent US9 508 362B2, Nov, 2016.

[13] F. Wu, B. Li, B. Zhang, Z. Cao, J. Diehl, H. Wen, and D. H. Du, "Tracklace: Data management for interlaced magnetic recording," *IEEE Transactions on Computers*, vol. 70, no. 3, pp. 347–358, 2021.

[14] M. H. Hajkazemi, A. N. Kulkarni, P. Desnoyers, and T. R. Feldman, "Track-based translation layers for interlaced magnetic recording," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, Renton, WA, Jul. 2019, pp. 821–832.

[15] Y. Liang and M.-C. Yang, "Move-on-modify: An efficient yet crash-consistent update strategy for interlaced magnetic recording," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 97–102.

[16] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[17] E. Shriver, C. Small, and K. A. Smith, "Why does file system prefetching work?" in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '99, USA, 1999, p. 6.

[18] J. Park, J. Kim, Y. Kim, S. Lee, and O. Mutlu, "DeepSketch: A new machine Learning-Based reference search technique for Post-Deduplication delta compression," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, Santa Clara, CA, Feb. 2022, pp. 247–264.

[19] S. Tarun, R. S. Batth, and S. Kaur, "A scheme for data deduplication using advance machine learning architecture in distributed systems," in *2021 International Conference on Computing Sciences (ICCS)*, 2021, pp. 53–60.

[20] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed Solid-State drive," in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, Philadelphia, PA, Jun. 2014.

[21] S.-H. Chen, C.-F. Wu, M.-C. Yang, and Y.-H. Chang, "A file-oriented fast secure deletion strategy for shingled magnetic recording drives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2463–2476, 2022.

Scalable Community Detection Using Quantum Hamiltonian Descent and QUBO Formulation

Jinglei Cheng[§*], Ruilin Zhou[†*], Yuhang Gan[†], Chen Qian[†], Junyu Liu[§]
[§]University of Pittsburgh [†]University of California, Santa Cruz
[*]Authors contributed equally to this research. Corresponding to: jic373@pitt.edu, junyuliu@pitt.edu

***Abstract*—We present a quantum-inspired algorithm that utilizes Quantum Hamiltonian Descent (QHD) for efficient community detection. Our approach reformulates the community detection task as a Quadratic Unconstrained Binary Optimization (QUBO) problem, and QHD is deployed to identify optimal community structures. We implement a multi-level algorithm that iteratively refines community assignments by alternating between QUBO problem setup and QHD-based optimization. Benchmarking shows our method achieves up to 5.49% better modularity scores while requiring less computational time compared to classical optimization approaches. This work demonstrates the potential of hybrid quantum-inspired solutions for advancing community detection in large-scale graph data.**

***Index Terms*—Quantum Hamiltonian Descent, QUBO, Community Detection**

I. Introduction

Quantum computing with the principles of quantum mechanics can process information in different ways from classical computers. This is because qubits can exist in superposition, and therefore can represent multiple states simultaneously [1]. This capability, combined with entanglement and interference, allows quantum algorithms to solve certain classes of problems, such as integer factorization and combinatorial optimization, more efficiently than classical methods. Therefore, quantum computing has lead to advancements in cryptography [2], optimization [3], machine learning [4], and materials science [5]. In parallel, quantum-inspired algorithms bring similar quantum concepts to classical computing, with principles such as tunneling, adiabatic transitions, and Hamiltonian dynamics to address complex problems without requiring quantum hardware. These approaches have demonstrated enhanced performance in areas like traffic routing and combinatorial optimization, where classical systems emulate quantum behaviors to achieve good efficiency [6].

Community detection is an important problem in network analysis, and it's essential for interpreting the structural and functional organization of complex systems [7]. An example is given in Figure 1. The overall goal is to identify communities—groups of nodes with denser connections among themselves than with the rest of the network [8]. In large-scale graphs, which are common in various real-world applications such as social networks, biological systems, and communication networks, effective community detection provides valuable insights into network behavior and properties, and can support applications like anomaly detection, network optimization, and information dissemination [9]. The impact of community detection is deep across many domains. In social networks, it enables the identification of groups with shared interests or behaviors, which can enhance recommendation systems and targeted marketing strategies [10], [11]. In biological networks, community detection reveals functional modules or protein complexes, and contributes to a better understanding of cellular processes and disease mechanisms [12], [13]. Additionally, fields like communication networks, financial systems, and transportation networks takes community detection to optimize efficiency, bolster security, and improve resilience against failures or attacks [14], [15].

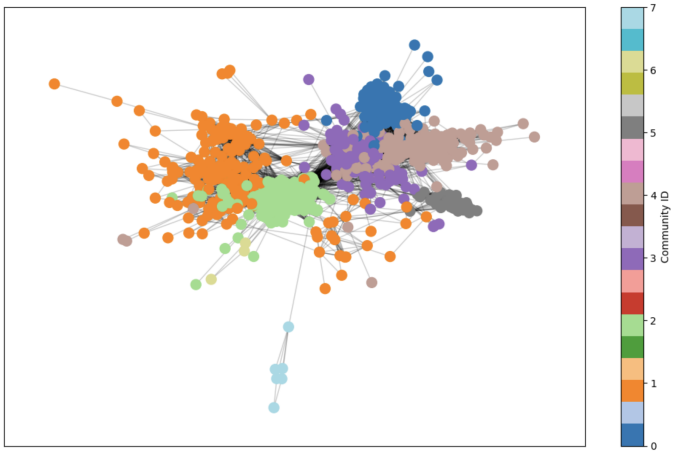


Fig. 1. Visualization of community structure in a complex network. The network consists of nodes grouped into communities, each represented by a different color.

Detecting communities in large-scale graphs presents substantial computational challenges. Traditional algorithms often struggle with scalability, resulting in increased processing times and high resource consumption as network size grows [16]. Furthermore, maintaining the accuracy and quality of detected communities becomes more complex with larger datasets due to the intricate network structures and the presence of noise [10]. These challenges call for the development of more scalable algorithms capable of handling large-scale graphs without compromising performance or accuracy [17].

Quantum-inspired algorithms inherit conceptions from quantum computing to enhance classical computational methods [18]–[22]. Quantum Hamiltonian Descent (QHD) [23], [24] is an algorithm designed to efficiently solve optimization problems by leveraging quantum tunneling effects to escape local minima, and therefore enhances performance in non-convex optimization scenarios. By simulating the behavior of quantum systems, Quantum Hamiltonian Descent can navigate complex energy landscapes to find optimal solutions more effectively than some traditional optimization techniques [25]. This makes it a promising approach for tackling the computational demands of large-scale community detection, where the formulation is not always convex.

Our approach involves formulating the community detection problem as a QUBO model as shown in Figure 2. The QUBO formulation is a versatile framework for representing combinatorial optimization problems where the objective is to minimize a quadratic function of binary variables [26]. By expressing community detection in this form, we can apply optimization algorithms like Quantum Hamilto-

nian Descent directly. In this way, the challenge of running graph algorithms on GPUs can be addressed by transforming them into a QUBO formulation, which allows for the deployment of GPU-accelerated algorithms. Quantum Hamiltonian Descent is particularly effective for solving QUBO models due to its ability to efficiently explore huge and complex solution spaces. By mimicking the dynamics of quantum systems, it can avoid becoming trapped in local minima and converge rapidly to high-quality solutions. This efficiency is essential for large-scale problems where traditional optimization methods may be too slow or require excessive computational resources.

To further improve efficiency and solution quality, we have developed a multi-level algorithm that iteratively refines community assignments. This algorithm alternates between QUBO formulation and Quantum Hamiltonian Descent solving at different levels of graph granularity. Starting from a coarse representation of the graph, it progressively refines the community structure by incorporating more detailed information in each iteration. This hierarchical approach helps in escaping local optima and enhances the overall accuracy of the community detection process.

Our method has been benchmarked against GUROBI [27], which is a leading commercial solver renowned for its optimization capabilities. On smaller problem instances, our algorithm matches GUROBI's performance, demonstrating its effectiveness and reliability. Notably, for larger problems exceeding 1,000 variables, our approach surpasses GUROBI. This performance advantage is crucial for practical applications involving large-scale graphs where computational resources and time are significant constraints. Recognizing the computational intensity of large-scale community detection, we have implemented our algorithm on multi-GPU architectures. GPUs offer massive parallel processing power, which accelerates the Quantum Hamiltonian Descent process and handles the extensive computations required for large graphs. Our implementation demonstrates excellent scalability and makes it feasible to analyze large networks in reasonable time budget. This efficient use of high-performance computing resources indicates the practicality of our approach for real-world applications.

Our contributions are:

- Developed a novel quantum-inspired community detection algorithm utilizing QHD optimization
- Formulated community detection as a QUBO problem with an iterative multi-level refinement approach
- Demonstrated competitive performance with GUROBI solver on small instances and superior scalability for problems exceeding 1,000 variables

The remainder of this paper is organized as follows. Section II provides the necessary background on community detection, QUBO problems, and quantum-inspired optimization techniques. In Section III, we present the formal mathematical framework of our approach, detailing the QUBO formulation. Section IV describes our implementation, including the multi-level algorithm design and QHD algorithm. We present comprehensive experimental results and performance analysis in Section V. Finally, Section VI concludes the paper with a summary of our contributions and directions for future research.

II. Background

A. Quantum Hamiltonian Descent

QHD introduces a new quantum counterpart to classical gradient descent methods, derived through path integral quantization of the continuous-time limit of gradient descent algorithms. Unlike conventional approaches that only quantize specific components of classical algorithms, QHD quantizes the entire dynamical system, resulting in a quantum evolution governed by a Hamiltonian $\hat{H}(t) = e^{\phi_t}(-\frac{1}{2}\Delta) + e^{\chi_t} f(x)$, where e^{ϕ_t} and e^{χ_t} are damping parameters controlling system energy flow. This formulation allows QHD to take advantages of quantum tunneling effects to escape local minima by considering all possible paths at the same time, including those prohibited in classical mechanics. The algorithm exhibits three distinct phases - kinetic, global search, and descent - and has demonstrated superior performance over both classical gradient-based methods and quantum adiabatic algorithms in solving non-convex optimization problems.

B. Community Detection

Community Detection (CD), also known as graph clustering or graph partition, is one fundamental task in graph theory and network science with the main goal of identifying groups of nodes within a graph that are more densely connected to each other than the rest. Applications of community detection span various fields, such as social network analysis [28], biology [29], computer networks, and recommendation systems [30]. Based on the definition of the nodes, applications, and the definition of a good partition. Various methods have been proposed, such as the modularity-based method [31], which aims to maximize modularity, which is a measure of the strength of the division of a network into communities, spectral clustering, which involves the use the eigenvalues and eigenvectors of matrices like the Laplacian derived from the graph to identify community structures based on the network's spectral properties and hierarchical clustering which use methods such as divisive clustering to build a hierarchy of communities either by progressively merging nodes (bottom-up) or recursively splitting larger communities (top-down) based on similarity measures.

III. Formalization

In this section, we introduce how we transform the CD problem into QUBO form and the main considerations of our formulations.

A. Modularity

Formally, the community detection problem with the goal of maximizing modularity can be formulated as follows: Given an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, the task of community detection is to partition the vertices into k non-empty communities such that nodes within the same community are more densely connected while nodes in different communities are sparsely connected. One commonly used metric to measure how well the graph is partitioned is 'modularity' which is defined by:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \delta(c_i, c_j) \tag{1}$$

where A_{ij} is the element of the adjacency matrix, d_i and d_j are the degrees of nodes i and j. And $\delta(c_i c_j)$ is the Kronecker delta function that equals 1 if nodes i and j are in the same community and 0 otherwise.

Various heuristics have been proposed to address the community detection problem. Cut-based methods aim to minimize inter-community edges while maintaining balanced partitions [32]. Modularity-based methods focus on maximizing the density of intra-community edges relative to a null model [33]. Another approach involves using quantum annealing; however, this method introduces additional overhead due to the need to embed the problem instance onto the quantum annealer [34].

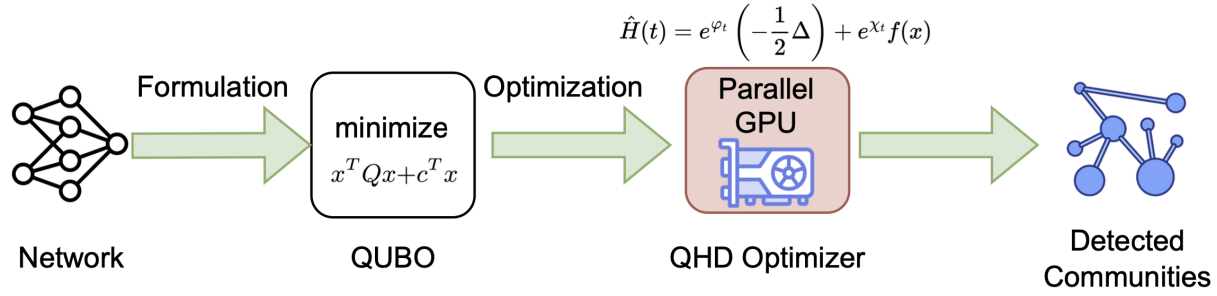


Fig. 2. Overview of our quantum-inspired community detection approach. The input network is reformulated as a QUBO optimization problem, which is then solved using QHD accelerated by parallel GPU computation. The QHD optimizer uses quantum-inspired dynamics to efficiently identify lowest cost function values in the complex landscape, and delivers superior scalability for instances with thousands of nodes.

B. QUBO Formulation for Community Detection

Inspired by previous classical and quantum methods, we propose a quantum-inspired method that takes Quantum Hamiltonian Descent to replace the original compute-intensive components of classical algorithms. This method first transforms the CD problem into a QUBO formulation, which is the first step for the use of both classical and quantum-inspired solvers that can be executed on GPUs without the need for direct embedding onto quantum hardware. The specific formalization is as follows.

Algorithm 1 QUBO Construction for Community Detection

Input:
n: number of nodes
k: number of communities
B: modularity matrix
$G(V, E)$: input graph
w_1, w_2, w_3: penalty weights
Output: QUBO matrix Q and linear terms b
Initialize $Q \in \mathbb{R}^{nk \times nk}, b \in \mathbb{R}^{nk}$ to zero
function IDX(i, c) **return** $i \cdot k + c$
end function
// Modularity term
$Q_{\text{idx}(i,c),\text{idx}(j,c)} \leftarrow -w_1 B_{ij}$
// Assignment constraints
$Q_{\text{idx}(i,c_1),\text{idx}(i,c_2)} \leftarrow 2w_2$
$b_{\text{idx}(i,c)} \leftarrow -w_2$
// Cut penalty
$i_c \leftarrow \text{idx}(u, c), j_c \leftarrow \text{idx}(v, c)$
$Q_{i_c,j_c} \leftarrow -2w_3$

1) Direct QUBO Formulation for Small Networks: To adapt the modularity optimization problem for QUBO solvers, we construct a QUBO objective function that encapsulates both the modularity measure and the necessary constraints for valid community assignments. Let binary variables $x_{i,c} \in \{0, 1\}$, where:

$$x_{i,c} = \begin{cases} 1 & \text{if vertex } i \text{ is assigned to community } c, \\ 0 & \text{otherwise.} \end{cases}$$

Assume a maximum of k communities, with $c \in \{1, 2, \ldots, k\}$. The modularity contribution can be expressed using the binary variables as:

$$Q_M = \frac{1}{2m} \sum_{i,j \in V} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \sum_{c=1}^{k} x_{i,c} x_{j,c}. \tag{2}$$

. Different from the previous formalization, which only focused on maximizing modularity, we also added other terms. To ensure that each vertex is assigned to exactly one community, we impose the following constraint:

$$Q_A = \lambda_A \sum_{i \in V} \left(1 - \sum_{c=1}^{k} x_{i,c} \right)^2, \tag{3}$$

where λ_A is a penalty coefficient that enforces the constraints, and expanding the squared term ensures that derivation from the constraint incur a quadratic penalty. To prevent trivial solution where all vertices are assigned to a single community or most vertices are assigned to several communities and aim to provide a balance size of communities, we also introduce the constraints on the size of communities:

$$Q_S = \lambda_S \sum_{c=1}^{k} \left(\sum_{i \in V} x_{i,c} - \frac{n}{k} \right)^2, \tag{4}$$

where λ_S is the penalty coefficient for balancing community sizes and $\frac{n}{k}$ represents the desired average community size. Based on these, the complete QUBO objective function to be maximized can be formed as

$$Q = -Q_M + Q_A + Q_S, \tag{5}$$

where $-Q_M$ corresponds to the maximization of the modularity, Q_A enforces the constraint that each vertex will be in exactly one community, and Q_S enforces a balanced constraint, which is also a consideration in most CD formalization. In general, our formalization gives comprehensive objective functions that take multiple aspects into consideration instead of solely maximizing the modularity metric. More details are shown in Algorithm III-B.

2) Scale to Larger Networks: While previous direct QUBO formalization is effective for small to medium-sized networks($|V| \leq 1000$), larger networks require more scalable approaches. To address this and inspired by classical methods [35], we employ a hierarchical methodology that maintains solution quality while reducing computational complexity. More specifically, our method includes the following steps: Coarsening, initial partition, uncoarsening, and refinement. More details are shown in Algorithm III-B2.

Coarsening Phase: We iteratively reduce the graph size by aggregating vertices into super-nodes, thereby preserving the community structure at multiple scales. At each coarsening step, a heavy-edge matching strategy is utilized to select vertex pairs with strong connectivity:

$$w(e) = \alpha \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} + \beta \frac{A_{uv}}{\max_{e \in E} A_{uv}}, \tag{6}$$

Algorithm 2 Multilevel Community Detection

Input:
$G_0(V_0, E_0)$: input graph
k: number of communities
θ: coarsening threshold
Output: Community assignments $P : V_0 \to 1, \ldots, k$
function COARSEN(G_i)
    Match nodes in G_i to form super-nodes
    Combine matched nodes to create G_{i+1} **return** G_{i+1}
end function
function PROJECT(P_{i+1}, G_i)
    Map community assignments from level $i+1$ to i **return** P_i
end function
// Coarsening Phase
$i \leftarrow 0$
while $|V_i| > \theta$ **do**
    $G_{i+1} \leftarrow$ COARSEN(G_i)
    $i \leftarrow i + 1$
end while
$m \leftarrow i$ // Final level
// Initial Solution
$P_m \leftarrow$ SOLVEBASE(G_m, k)
// Uncoarsening Phase
for $i = m - 1$ **downto** 0 **do**
    $P_i \leftarrow$ PROJECT(P_{i+1}, G_i)
    $P_i \leftarrow$ REFINE(P_i, G_i, k)
end for
return P_0

where $N(u)$ denotes the neighborhood of vertex $u.\alpha$, and β are weighting coefficients that balance the contribution of neighborhood overlap and edge weight. This coarsening phase reduces the number of vertices while preserving the essential community structure.

Initial Partition: Once the graph is sufficiently coarsened to a manageable size, we apply the direct QUBO formulation (5) to partition the coarsest graph. This step provides an initial community assignment that captures the high-level structure of the original graph.

Uncoarsening and Refinement: After initial partitioning is found in the coarsest graph, the partitioning is then projected back to the original graph through the following steps:

1) **Projection**: Starting from the coarsest level, iteratively map the community from each coarsened graph G_i to the next finer graph G_{i-1}. This is based on the super-node correspondence established during the coarsening phase which can effectively transfer the high-level community structure to more detailed graph levels.
2) **Refinement**: At each level, we optimize the community assignments by iteratively reassigning nodes to communities that result in the highest gain in modularity. This involves evaluating potential moves for each node and updating its community assignment if the move improves the overall modularity. The refinement process continues until certain iterations of the threshold are met or no further improvement is achieved.

IV. IMPLEMENTATION

A. QHD on QUBO

QHDOPT [25] implements Quantum Hamiltonian Descent for solving nonlinear optimization problems. Its key innovation is implementing this optimization via quantum evolution governed by the time-dependent Schrödinger equation:

$$i\frac{\partial}{\partial t}\Psi(t,x) = \left(e^{\phi_t}\left(-\frac{1}{2}\Delta\right) + e^{\chi_t} f(x)\right)\Psi(t,x)$$

where Δ is the Laplacian operator and e^{ϕ_t}, e^{χ_t} control the energy distribution of the quantum system.

A significant computational advantage of this approach is that it requires only matrix multiplication operations, avoiding the need to solve linear systems (Ax=b). This makes the method particularly amenable to modern hardware acceleration. For practical implementation, QHDOPT employs a discretization and embedding strategy where the continuous Hamiltonian is discretized into an N^n-dimensional operator:

$$\hat{H}(t) = e^{\phi_t}\left(-\frac{1}{2}L_d\right) + e^{\chi_t} F_d$$

where L_d represents the discretized Laplacian and F_d encodes the objective function.

The method's matrix-multiplication-based formulation opens up possibilities for various compression techniques that can transform large sparse problems into smaller, denser ones. This includes techniques like tensor network compression [36], hierarchical matrix approximations [37], and randomized sketching methods [38]. Any additional constraints created during problem transformation can be efficiently handled through penalty-based methods, avoiding the need for complex constraint satisfaction algorithms.

When dealing with QUBO problems, QHDOPT can be efficiently accelerated using GPUs through careful implementation of the quantum Hamiltonian dynamics. The key is mapping the QUBO problem $\min_{x\in\{0,1\}^n} x^T Qx + b^T x$ to QHD's Hamiltonian framework in a way that exploits massive parallelism. Sparse operations can be particularly accelerated using specialized GPU packages like cuSPARSE [39] or MAGMA [40], which provide optimized implementations for sparse matrix operations. The time evolution can be implemented using GPU-optimized linear algebra operations, with the Laplacian operator Δ computed using parallel finite difference schemes and the potential term $e^{\chi_t} f(x)$ evaluated using batched matrix operations.

By leveraging frameworks like PyTorch [41] or JAX [42] that provide automatic differentiation and GPU acceleration, the quantum dynamics can be simulated efficiently - the gradient computations for the Hamiltonian evolution can be parallelized across the quantum state dimensions, and the time evolution steps can be batched for exploring multiple initial conditions simultaneously. The classical refinement step in QHDOPT can utilize GPU-accelerated optimizers to efficiently project solutions back to binary values. This hybrid quantum-classical approach combines quantum effects with massive parallel computing capabilities, making it particularly effective for large-scale optimization problems where traditional methods might struggle.

B. Multi-level Community Detection

This multilevel community detection algorithm is designed to efficiently handle large-scale graphs by employing a hierarchical approach. The algorithm operates in three main phases: First, in the coarsening phase, it repeatedly combines nodes to create a hierarchy of progressively smaller graphs while preserving the community structure. Once the graph is sufficiently small, it solves the community detection problem directly on this coarsened graph. Finally, in the uncoarsening phase, it progressively maps the solution

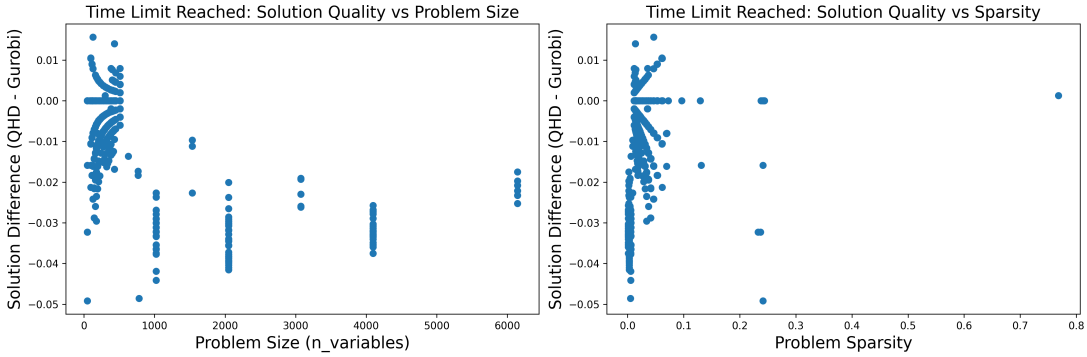


Fig. 3. Solution Quality Comparison When GUROBI Hits Time Limit: For 739 instances where GUROBI exceeded time limits (predominantly in larger problems), QHD found better solutions in 71.4% of cases. For larger-scale problems, QHD demonstrated superior performance by finding better solutions than GUROBI within the same time constraints. This advantage stems from QHD's inherent parallel processing capabilities, which can be further enhanced through additional GPU resources and optimized sparse matrix operations, suggesting even greater potential for scaling to larger problem instances.

back to finer levels while refining the communities at each step. This multilevel approach allows the algorithm to maintain both computational efficiency and solution quality by working with smaller problems during the initial solution while preserving the ability to make fine-grained adjustments during refinement.

V. Evaluation

A. Evaluation Setting

The experimental evaluation was conducted on a system running Debian Linux with kernel version 5.10.0-22-amd64. The computing platform features a dual 16-core CPU (32 cores total). The server is equipped with 251 GB of RAM. Implementation of QHD on QUBO is running with four NVIDIA A5000 GPUs and the comparison with GUROBI is conducted with CPUs as mentioned.

B. QUBO Solver with GUROBI and QHD

Comparing the computational performance between GUROBI and QHD presents unique challenges due to their different control parameters. While GUROBI can be configured with node exploration limits and termination times, QHD allows adjustment of sample sizes and iteration counts. To establish comparisons, we adopted a time-based benchmarking approach: first measuring QHD's execution time, then allocating GUROBI the same as its time limit. This methodology is justified because superior performance can be demonstrated either through faster execution at equal solution quality or through better solutions within equal time constraints. Our experiments focused on the latter approach, comparing solution quality between the two solvers under equivalent time constraints. The results demonstrate that QHD achieves higher solution quality while GUROBI cannot finish with status as OPTIMAL within the allocated time. It's worth noting that QHD's performance can be further enhanced with additional computational resources due to its inherent massive parallelism capabilities. The results in Figure **??** and Figure 3 demonstrate robust performance patterns across a substantial dataset of 938 instances. In the 199 cases where GUROBI found proven optimal solutions (typically smaller problems with mean size of 54 variables), QHD matched these optimal solutions in 75.4% of cases. More importantly, in the larger set of 739 instances where GUROBI hit its time limit (predominantly larger problems with mean size of 614 variables), QHD demonstrated superior performance by finding better solutions in 71.4% of cases and matching GUROBI's solutions in another 17.2%. While the time-limited instances show lower sparsity (mean 0.028 vs 0.157), this is primarily a characteristic of larger problem instances rather than a direct driver of performance difference. The results, based on this comprehensive dataset, strongly indicate QHD's effectiveness for larger-scale optimization problems where traditional solvers face computational limitations, regardless of sparsity.

C. Evaluation on Small Size Networks

Our experimental evaluation compared the Quantum Hamiltonian Descent approach with the GUROBI-based modularity optimization across a diverse set of network instances. As shown in Table I, the test bed has 10 networks ranging from a graph with (52 nodes and 146 edges) to an instance of (1,034 nodes, 26,749 edges), with edge densities from 3.4% to 15.2%. As shown in Fig. 5, the QHD-based method demonstrated robust performance, achieving higher modularity scores in 8 out of 10 instances compared to the GUROBI implementation. The average modularity difference of 0.0029 in favor of QHD suggests that quantum-inspired optimization can effectively match or slightly exceed traditional exact optimization approaches. This robust performance, where QHD achieves higher modularity scores in 8 out of 10 test instances while using only 20% of GUROBI's computational time with four GPUs, demonstrates its practical advantages over classical modularity optimization methods, with the potential for even greater efficiency gains through increased GPU parallelization.

TABLE I
Instance Properties and Modularity Scores

Instance	Nodes	Edges	Density %	GUROBI	QHD
0	333	2,519	4.56	0.4523	**0.4610**
107	1,034	26,749	5.01	**0.5290**	0.5241
348	224	3,192	12.78	0.3055	**0.3063**
414	150	1,693	15.15	0.5438	**0.5438**
686	168	1,656	11.80	0.3347	**0.3347**
698	61	270	14.75	0.5369	**0.5369**
1684	786	14,024	4.55	0.5528	**0.5640**
1912	747	30,025	10.78	0.5167	**0.5239**
3437	534	4,813	3.38	0.6724	**0.6784**
3980	52	146	11.01	**0.4619**	0.4619

D. Evaluation on Large Size Networks

Table II and Figure 6 of GUROBI and QHD performance across different networks reveal a significant density-dependent pattern. In the Facebook network (density 0.0108), QHD achieves a notable 5.49% higher modularity score (0.7512 ± 0.0258) compared to

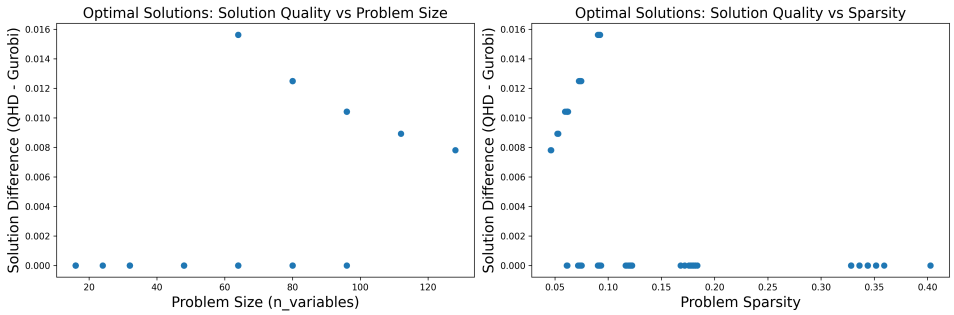


Fig. 4. Solution Quality Comparison When GUROBI Reaches Optimality: Among 199 instances where GUROBI found optimal solutions (predominantly in smaller problems), QHD achieved identical solutions in 75.4% of cases. In cases where QHD did not match GUROBI's optimal solutions (24.6% of optimally solved instances), the objective value differences remained within a 1.6% relative gap, primarily occurring in smaller instances where the absolute differences in objective values were minimal.

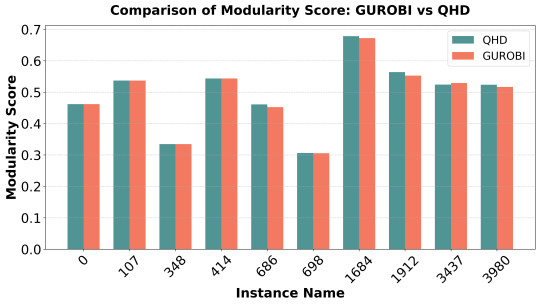


Fig. 5. Performance comparison between QHD and GUROBI on network instances ranging from 52 to 1,034 nodes. QHD achieves superior modularity scores in 80% of test cases with an average improvement of 0.0029, while requiring only 20% of GUROBI's computational time using four GPUs. Results demonstrate QHD's effectiveness across different network densities (3.4%-15.2%).

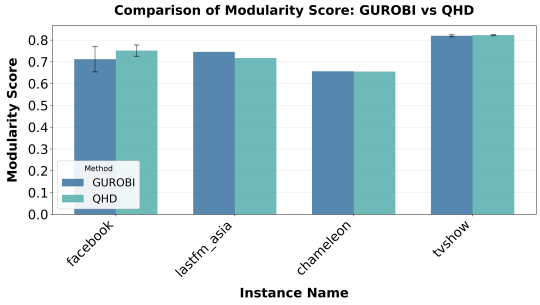


Fig. 6. The performance advantage varies with network density, from QHD's 5.49% improvement on Facebook (density 0.0108) to GUROBI's 3.79% advantage on sparse LastFM (density 0.0010). Both methods show comparable performance on medium-density networks.

GUROBI (0.7121 ± 0.0579), while also demonstrating better stability with lower variance. The densest network, Chameleon (density 0.0121), shows nearly identical performance between the methods, with only a 0.19% difference favoring GUROBI. The TVShow network, despite its lower density (0.0023), exhibits near performance from both methods, with QHD showing a marginal advantage of 0.33% (QHD: 0.8223 ± 0.0025, GUROBI: 0.8196 ± 0.0044) and both methods maintaining very low variance. Notably, in the sparsest network, LastFM (density 0.0010), GUROBI outperforms QHD by 3.79%. These results indicate that QHD's advantages are most pronounced in moderately dense networks. And GUROBI maintains good performance across network types, particularly in sparser networks, which is attributed to its branch-and-cut algorithm.

TABLE II
COMPARISON OF GRAPH PROPERTIES AND MODULARITY SCORES

Instance	Nodes	Edges	Density %	GUROBI	QHD
facebook	4,039	88,234	1.08	0.7121	**0.7512**
lastfm_asia	7,626	27,807	0.10	**0.7455**	0.7172
musae_chameleon	2,279	31,372	1.21	**0.6567**	0.6554
tvshow	3,894	17,240	0.23	0.8196	**0.8223**

VI. CONCLUSION

This work demonstrates the effectiveness of quantum-inspired optimization for community detection through our QHD-based approach. By reformulating community detection as a QUBO problem and leveraging GPU-accelerated quantum-inspired optimization, we achieve comparable or superior performance to traditional methods while reducing computational time. Our experimental results show strong performance on moderately dense networks, with up to 5.49% improvement in modularity scores and enhanced stability compared to GUROBI. The implementation of multi-GPU acceleration suggests further scalability potential for larger networks. These findings highlight the promising intersection of quantum-inspired algorithms and high-performance computing for complex network analysis tasks. Future work could explore extending this approach to other graph optimization problems and investigating additional acceleration techniques for handling ultra-large-scale networks. Better designed algorithms to formulate graphs into denser matrices can reduce the number of variables in QUBO, and combination with high-performance sparsity computation will also be helpful. Our results demonstrate that quantum-inspired methods, combined with modern computing architectures, offer a practical pathway for advancing the capabilities of community detection in real-world network analysis applications.

VII. ACKOWLEDGEMENT

JL is supported in part by the University of Pittsburgh, School of Computing and Information, Department of Computer Science, Pitt Cyber, PQI Community Collaboration Awards. And by NASA under award number 80NSSC25M7057. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. RL is supported by NSF Grants 2322919, 2420632, 2426031, 2426940, 2114113, and DOE Grant DE-SC0022069. We thank Yuxiang Peng from University of Maryland and Jiaqi Leng from UC Berkely for their helpful and insightul discussions.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. Cambridge, UK: Cambridge University Press, 2010.

[2] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, pp. 124–134.

[3] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

[4] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, pp. 195–202, 2017.

[5] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, pp. 242–246, 2017.

[6] A. Laio and M. Parrinello, "Escaping free-energy minima," *Proceedings of the national academy of sciences*, vol. 99, no. 20, pp. 12 562–12 566, 2002.

[7] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3–5, pp. 75–174, 2010.

[8] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[9] M. E. J. Newman, *Networks: An Introduction*. Oxford, UK: Oxford University Press, 2010.

[10] S. Fortunato and D. Hric, "Community detection in networks: A user guide," *Physics Reports*, vol. 659, pp. 1–44, 2016.

[11] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics Reports*, vol. 533, no. 4, pp. 95–142, 2013.

[12] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabasi, "Hierarchical organization of modularity in metabolic networks," *Science*, vol. 297, no. 5586, pp. 1551–1555, 2002.

[13] A.-L. Barabasi, N. Gulbahce, and J. Loscalzo, "Network medicine: A network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 1, pp. 56–68, 2011.

[14] M. Newman, *Networks*. Oxford, UK: Oxford University Press, 2018.

[15] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics," *Physics Reports*, vol. 424, no. 4–5, pp. 175–308, 2006.

[16] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Computing Surveys*, vol. 51, no. 2, pp. 1–37, 2018.

[17] E. Abbe, "Community detection and stochastic block models: Recent developments," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6446–6531, 2017.

[18] J. M. Arrazola, A. Delgado, B. R. Bardhan, and S. Lloyd, "Quantum-inspired algorithms in practice," *Quantum*, vol. 4, p. 307, 2020.

[19] N. Chepurko, K. Clarkson, L. Horesh, H. Lin, and D. Woodruff, "Quantum-inspired algorithms from randomized numerical linear algebra," *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, pp. 3879–3900, 2022.

[20] C. Shao and A. Montanaro, "Faster quantum-inspired algorithms for solving linear systems," *arXiv preprint arXiv:2103.10309*, 2021.

[21] V. Yelleti and R. Krishna, "Quantum-inspired evolutionary algorithms for feature subset selection: A comprehensive survey," *arXiv preprint arXiv:2407.17946*, 2023.

[22] H. Okawa, Q.-G. Zeng, X.-Z. Tao, and M.-H. Yung, "Quantum-annealing-inspired algorithms for track reconstruction at high-energy colliders," *arXiv preprint arXiv:2402.14718*, 2024.

[23] J. Leng, E. Hickman, J. Li, and X. Wu, "Quantum hamiltonian descent," *arXiv preprint arXiv:2303.01471*, 2023.

[24] J. Leng, J. Li, Y. Peng, and X. Wu, "Expanding hardware-efficiently manipulable hilbert space via hamiltonian embedding," *arXiv preprint arXiv:2401.08550*, 2024.

[25] S. Kushnir, J. Leng, Y. Peng, L. Fan, and X. Wu, "Qhdopt: A software for nonlinear optimization with quantum hamiltonian descent," *arXiv preprint arXiv:2409.03121*, 2024.

[26] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, and H. Wang, "The unconstrained binary quadratic programming problem: A survey," *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[27] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[28] M. Azaouzi, D. Rhouma, and L. Ben Romdhane, "Community detection in large-scale social networks: state-of-the-art and future directions," *Social Network Analysis and Mining*, vol. 9, no. 1, p. 23, 2019.

[29] F. R. Khawaja, Z. Zhang, Y. Memon, and A. Ullah, "Exploring community detection methods and their diverse applications in complex networks: a comprehensive review," *Social Network Analysis and Mining*, vol. 14, no. 1, p. 115, 2024.

[30] F. Gasparetti, G. Sansonetti, and A. Micarelli, "Community detection in social recommender systems: a survey," *Applied Intelligence*, vol. 51, pp. 3975–3995, 2021.

[31] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, p. 026113, 2004.

[32] H. Shin, J. Park, and D. Kang, "A graph-cut-based approach to community detection in networks," *Applied Sciences*, vol. 12, no. 12, p. 6218, 2022.

[33] M. Rosvall, J.-C. Delvenne, M. T. Schaub, and R. Lambiotte, "Different approaches to community detection," *Advances in network clustering and blockmodeling*, pp. 105–119, 2019.

[34] M. Fernández-Campoamor, C. O'Meara, G. Cortiana, V. Peric, and J. Bernabé-Moreno, "Community detection in electrical grids using quantum annealing," *arXiv preprint arXiv:2112.08300*, 2021.

[35] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, University of Minnesota, Minneapolis, MN, 1998, version 5.1.0. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[36] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Foundations and Trends® in Machine Learning*, vol. 9, no. 4-5, pp. 249–429, 2016.

[37] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*. Springer, 2015.

[38] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.

[39] NVIDIA Corporation, *cuSPARSE Library*, 2024, version 12.6. [Online]. Available: https://docs.nvidia.com/cuda/cusparse/index.html

[40] U. o. T. Innovative Computing Laboratory, *MAGMA Library*, 2024, version 2.8.0. [Online]. Available: https://icl.utk.edu/magma/

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[42] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "Jax: composable transformations of python+numpy programs," 2018. [Online]. Available: http://github.com/google/jax

Parallel Dynamic Partitioning for Datapath Combinational Equivalence Checking

Shuai Zhou*§, Weikang Zhang*, Xindi Zhang†, Zite Jiang*§, Haihang You*¶, Shaowei Cai†¶
*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†Key Laboratory of System Software (Chinese Academy of Sciences) and
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
§School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China
Emails: {zhoushuai22s, zhangweikang, jiangzite19s}@ict.ac.cn; zhangxd@ios.ac.cn; caisw@ios.ac.cn; youhaihang@ict.ac.cn

***Abstract*—Combinational Equivalence Checking (CEC) is a crucial technique in electronic design automation for verifying the functional equivalence of combinational circuits. Recently, combinational circuit design increasingly incorporates more complex arithmetic structures, commonly known as datapath circuits. However, existing state-of-the-art tools often exhibit subpar performance in solving datapath CEC problems. To further advance the exploration on datapath CEC process, this study introduces PDP-CEC (Parallel Dynamic Partitioning Combinational Equivalence Checking), a novel parallel CEC approach integrating circuit partitioning and dynamic task scheduling into the CEC process, enhancing the efficiency of CEC for datapath circuits. PDP-CEC introduces an innovative method for selecting critical nodes to split the search space of the CEC problem, facilitating the efficient generation of numerous independent subproblems. Meanwhile, a dynamic task scheduling strategy is implemented in PDP-CEC to ensure load balancing and prevent hard-to-solve subproblems from stalling the entire process. Compared to the most advanced tools such as ABC and Hybrid-CEC, PDP-CEC significantly accelerates CEC process, achieving speedups ranging from 5.11x to 125.27x, while effectively solving approximately three times more datapath CEC problems. With excellent scalability, PDP-CEC shows substantial improvements in combinational equivalence checking for datapath circuits, offering an efficient parallel approach to meet the demands of large-scale datapath CEC tasks.**

***Index Terms*—Combinational Equivalence Checking, parallel computing, circuit partitioning, dynamic task scheduling.**

I. Introduction

Combinational equivalence checking (CEC) involves verifying whether two combinational circuits are functionally equivalent. In contemporary circuit design processes [1], circuits are optimized through structural rewriting [2] or logic synthesis [3] after the RTL design stage. CEC is then employed to ensure that the circuit's functionality remains consistent before and after those optimizations. Its further applications cover various scenarios in electronic design automation (EDA) and digital integrated circuit design, including functional equivalence removal [4], computation of structural choice [5] and sequential equivalence checking [6].

Today, there has been a clear trend toward designing combinational circuits with increasingly complex arithmetic components [7]. Modern circuits are typically composed of fundamental arithmetic units, such as multipliers and multiplexers, collectively referred to as datapath circuits [8]. However, state-of-the-art CEC tools that are based on the SAT sweeping algorithm [9] [10] perform poorly when applied to datapath circuits.

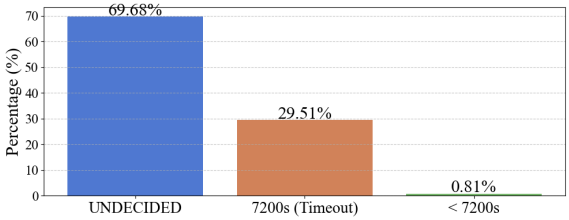


Fig. 1. Result distribution of ABC "&cec" on 742 industrial datapath circuits. "UNDECIDED" indicates the tool's inability to generate the result.

For example, 742 industrial datapath circuit cases were verified with the ABC "&cec" command [11], a widely used tool for logic synthesis and formal verification in both industry and academia, whose core is the SAT sweeping algorithm. As shown in Fig. 1, nearly 70% of the cases returned "UNDECIDED", while 29% produced no results after two hours. Only 1% completed verification within the time limit.

To improve the weak performance of these tools on datapath CEC problems, prior studies have predominantly concentrated on optimizing the CEC solving process, including more intelligent simulation methods [9] and unified representations [12]. As research on serial algorithms reaches saturation, it becomes crucial to explore universal parallel workflow for datapath circuits. There are two key challenges in parallelizing the datapath CEC process:

- **Ineffective partition** Circuits with high coupling and low locality, if divided naively, risk disrupting functional dependencies, which will create interdependent subcircuits. To generate independent subcircuits, the existing parallel approach [13] partitions circuits with Transitive Fan-In (TFI) cones of Primary Outputs (POs). Similarly, RepCut [14] employs hypergraph partition based on TFI levels. However, these partitioning methods that rely on TFI do not be well-suited for datapath circuits, which typically have a limited number of POs.

¶ Corresponding author

- **Load imbalance** Due to the inherent unpredictability of each subcircuit's workload, driven by the SAT solver at the core of CEC, efficiently balancing tasks in parallel execution becomes challenging. A parallel strategy [15] has been proposed in Satisfiability Modulo Theories (SMT) domain, with a binary tree to store subproblems' results and guide thread actions. However, this approach falls short for the parallel CEC workflow. Starting the execution from the root node's task can lead to significant thread stalling, particularly for the root problem and its nearby subproblems.

To overcome these challenges, we propose PDP-CEC (Parallel Dynamic Partitioning Combinational Equivalence Checking), a novel parallel CEC approach for datapath circuits that integrates circuit partitioning and dynamic task scheduling into the CEC process.

The circuit partitioning algorithm in PDP-CEC utilizes a distinctive node selection method to split the search space, effectively bypassing structural constraints and generating numerous independent subproblems. It incorporates a heuristic to estimate the complexity reduction achieved through partitioning, ensuring that removing assignment nodes propagated and inferred from the selected nodes substantially diminishes the circuit's overall complexity. To enhance this process, we integrate the Metis graph partitioning algorithm [16] and develop an approach for selecting suitable nodes in alignment with the circuit's structure.

Furthermore, PDP-CEC implements a recursive workflow based on a binary splitting tree to guide task scheduling during execution, which shows in Fig. 2. PDP-CEC employs the "master-worker" structure, comprising a master for expanding the binary splitting tree, a monitor for managing task scheduling, and workers for solving subproblems in parallel. With master pre-partitioning the circuit to initialize the tree, the task scheduling strategy helps PDP-CEC attain load balancing among workers, effectively reducing resource waste from idle workers and preventing hard-to-solve subproblems from stalling the entire process.

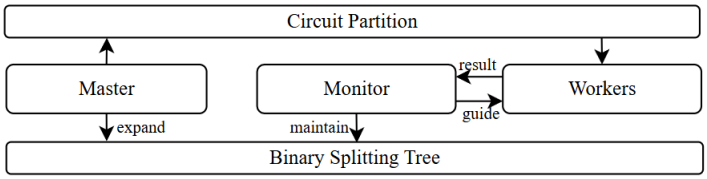


Fig. 2. The thumbnail of PDP-CEC's workflow.

In summary, we present PDP-CEC to fully exploit the potential of circuit structures, and our work makes the following contributions:

- We propose PDP-CEC, which integrates circuit partitioning and dynamic task scheduling to parallelize datapath CEC. With excellent scalability, PDP-CEC considerably outperforms state-of-the-art tools like ABC and Hybrid-CEC achieving speedups ranging from 5.11x to 125.27x with 128 threads in solving time and effectively resolving nearly three times more datapath CEC problems.
- We design a circuit partitioning algorithm for PDP-CEC that employs targeted node selection to split the search space, bypassing structural constraints and generating independent subproblems while ensuring that removing assignment nodes propagated and inferred from the selected nodes substantially decreases the circuit's overall complexity.
- We introduce a dynamic task scheduling strategy to attain load balancing during execution, avoiding resource waste from idle workers and preventing hard-to-solve subproblems from stalling the entire process.

Paper Organization Section II introduces the SAT sweeping algorithm. Section III presents the circuit partitioning algorithm and heuristic for selecting crucial nodes. Section IV details the workflow of PDP-CEC and gives an example of our dynamic scheduling strategy. Section V shows the experimental results, comparing PDP-CEC with state-of-the-art CEC tools. Finally, Section VI concludes the study and discusses future work.

II. Preliminaries

In this section, we introduce the typical CEC process which relies on the *SAT sweeping* algorithm and give an overview in Fig. 3.

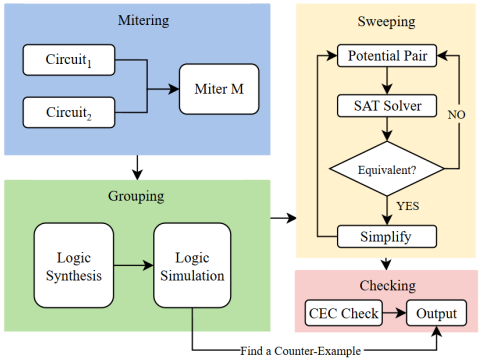


Fig. 3. The typical CEC process with SAT sweeping algorithm.

A complete CEC process typically involves four stages: constructing a miter circuit through mitering, grouping functionally potential-equivalent nodes by logic synthesis [17] and simulation, employing a SAT solver to identify potential-equivalent nodes and simplify the original miter circuit in a topological order through sweeping, and finally, checking the equivalence of the simplified circuit as the output.

Primarily suited for datapath CEC problems, Hybrid-CEC process [18] differs from traditional CEC in two key aspects: (1) it employs the ISD method [18] to reduce the number of pairs requiring verification by comparing TFI's structure with previously proven equivalent pairs, and (2) it partly replaces the SAT solver with the Exact Probability-based Simulation (EPS) method [19], which is better suited for solving low-

width pairs in datapath circuits. However, this method cannot serve as a comprehensive solution for all datapath circuits.

III. Circuit Partitioning Algorithm

Conventional graph partitioning methods often fail to yield parallelizable subcircuits because of high coupling within datapath circuits. To confront the structural intricacies of these circuits, we employ search space splitting to partition them into subcircuits for the following two main reasons:

- Structural information is lost when circuits are sent to the SAT solver. Search space partitioning on circuit level allows for better use of this information.
- Search space splitting avoids the intricate connectivity between gate nodes, focusing on dividing the search space instead of the structure.

In an And-Inverter Graph (AIG), each node represents an AND gate (or an XOR gate after preprocessing in PDP-CEC) with a maximum fan-in of 2 and a minimum fan-out of 1, except for output gates. Given the inherent characteristics of the circuit, each gate can only take values of 0 or 1, corresponding to low or high voltage levels. When assigning 0 or 1 to a gate, the search space of the original circuit is effectively divided into two independent regions. Based on the assignment, forward inference along the input direction and backward propagation along the output direction will be performed to assign new nodes through the property of gates. Removing those gates with assignments modifies the circuit structure, resulting in two smaller, independent subcircuits that each cover half the search space of the original circuit. We refer to this process of partitioning through assignment propagation as circuit incremental partitioning.

Algorithm 1 IncrementalPartitioning

1: **Input:** $TreeLevel\ L, Miter\ M(AIG)$
2: **Output:** $BinaryTree\ BStree$
3: Insert M into $BStree$
4: $Queue\ BSque$
5: $BSque.push(M)$
6: **while** $BStree.level \leq L$ **do**
7: $u \leftarrow BSque.pop()$
8: $x \leftarrow$ NodeSelecting(u)
9: $C_0 \leftarrow$ AssignValue$(u, x, 0)$
10: $C_1 \leftarrow$ AssignValue$(u, x, 1)$
11: Insert C_0 and C_1 into $BStree$
12: Push C_0 and C_1 into $BSque$
13: **end while**
14: **return** $BStree$

The Algorithm 1 describes the process of generating a binary splitting tree $BStree$ with a specified level L from the input miter circuit M. As the tree unfolds layer by layer, each parent node branches into two child nodes, representing an equal partitioning of the problem space. This partitioning process continues until it reaches the setting level L, which is determined by the circuit complexity and the number of threads available for parallel execution.

A. Subcircuits Creation

In lines 9 and 10 of Algorithm 1, the *AssignValue* function is invoked to partition circuit $C(V, E)$ into two subcircuits, $C_0(V_0, E_0)$ and $C_1(V_1, E_1)$ by assigning the gate x to 0 and 1, respectively. In the circuit topology, signals assigned to one gate can propagate in two directions. The term *inference* describes backward propagation along the input direction; for instance, if an AND gate's output is 1, both inputs must also be 1. Conversely, forward propagation occurs along the output direction, referred to as *propagation*. In this case, an AND gate's output of 0 will cause the output of the subsequent gate to be 0 as well, unless an inverter is present between them. Removing the assigned gates divides the circuit into two subcircuits with a combined search space equivalent to that of the original. In an AIG, this removal may create single-input gates, which can be simplified to wires, effectively reducing the overall circuit complexity.

Uncertain gates are stored as constraints. For instance, if an AND gate is assigned a 0, the constraint is that both inputs cannot simultaneously be 1, represented as (x_1,x_2,TT). These constraints serve as unique properties of the subcircuits and are subsequently converted into CNF expressions for SAT solver.

B. Nodes Selection

In line 8 of Algorithm 1, the *NodeSelecting* function is called to select appropriate nodes for first assignment. The selection of nodes is critical for the efficiency of the parallel execution process, as we aim to always choose nodes that significantly reduce circuit complexity during partitioning. Before introducing the algorithm for node selection, we first present the heuristic formula used to estimate the complexity of datapath circuit for SAT solving, which is evaluated by counting the number of XOR and AND gates, as expressed in the following formula:

$$\text{Complexity} = 3\log_2(\textit{num}_x) + \log_2(\textit{num}_a), \tag{1}$$

where $\textit{num}_x$ and $\textit{num}_a$ represent the number of XOR and AND gates, respectively.

In datapath circuits, components like multipliers and multiplexers are primarily built from adders, where XOR gates calculate sums and AND gates generate carries. Thus, the number of XOR gates, as a key indicator of the number of adders, carries more weight in (1) while AND gate count reflects circuit size. After partitioning, the number of removed nodes helps estimate subcircuit complexity using (1). However, calculating the exact impact of each node is time-consuming, so we further propose the following heuristic formulas to estimate the extent to which the selected nodes reduce overall circuit complexity:

$$\text{NC} = \frac{\textit{edge}_b - \textit{edge}_b \cdot \left(p_t \cdot \alpha \cdot (1 - \textit{rate}_x)\right)^{\textit{level}}}{1.0 - p_t \cdot \alpha \cdot (1 - \textit{rate}_x)}, \tag{2}$$

$$\text{Eval} = 3\log_2(\text{NC} \cdot \textit{rate}_x) + \log_2(\text{NC} \cdot (1 - \textit{rate}_x)). \tag{3}$$

The heuristic formula (2) for estimating the reduction in node count (NC) after partitioning is based on the geometric

series sum formula. The first term, $edge_b$, represents the number of different types of output edges, as edges with opposite signal assignments can aid propagation. The common ratio is $p_t \cdot \alpha \cdot (1 - rate_x)$, where p_t is the probability of a positive output edge, α is the average number of outgoing edges from a node, and $rate_x$ is the probability of XOR gates in the circuit. The variable *level*, representing the node's distance to the circuit's PO, helps gauge the propagation range. The 0 signal will continue to propagate through positive output edges after one round of propagation, so we use the probability of a positive output edge in the common ratio. By splitting NC into AND and XOR gates with $rate_x$, and applying (1), we can calculate the complexity reduction (Eval) for each node in (3). The node with the highest evaluation score (Eval) is chosen as the initial assignment node for partitioning, whose details are described in Algorithm 2.

Algorithm 2 NodeSelecting

```
Input: Circuit C
Output: Node N
Array Nodes ← GetPotentialNodes(C)
for node ∈ Nodes do
    h_0 ← EvalCompute(node, 0)
    h_1 ← EvalCompute(node, 1)
    node.sum_h ← h_0 + h_1
end for
best_node ← max_sum_h(Nodes)
return best_node
```

Algorithm 2 first identifies the range of potential optimal nodes in the circuit. For circuits solved with EPS, the range of selection is limited to the primary inputs (PI), as the number of PIs is a key factor influencing the speed of EPS. Since XOR chains pose a challenge for SAT-based methods [18], the Metis algorithm is applied to identify a cut set containing most of the XOR gates, aiming to reduce the number of XOR gates through incremental partitioning for circuits solved with SAT. The node that yields the greatest complexity reduction is then selected as the initial assignment node.

IV. Dynamic Task Scheduling Workflow

In PDP-CEC, potential equivalent pairs are directed to either SAT or EPS, depending on the width of miter circuit formed by the pairs' TFI cones. Once the solving method is determined, the original CEC problem will be divided into subproblems, which are then solved in parallel by threads within a thread pool. However, uneven distribution of subproblems across threads can lead to load imbalances. To prevent this issue, PDP-CEC incorporates a dynamic task scheduling workflow, including a binary splitting tree to guide task scheduling.

The workflow of PDP-CEC is illustrated in Fig. 4. Benefiting from the independence of the subproblems generated by partitioning, the framework adopts a "master-worker" structure. Avoiding complexity in control logic, we designed three types of threads: the master thread, the monitor thread, and worker threads in the thread pool.

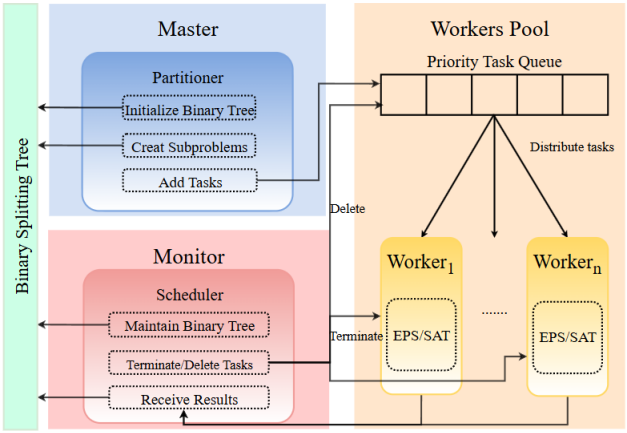


Fig. 4. The Workflow of PDP-CEC.

Master thread initializes the binary splitting tree with the help of the partitioner. In situations where the task queue has gaps, it identifies unresolved leaf nodes, generates new subproblems via the partitioner, adds these to the task queue, and facilitates the growth of the binary splitting tree.

Monitor thread serves as a central control mechanism for managing task scheduling within the workflow. It works alongside the master thread to oversee the node information stored in the binary splitting tree, controls the states of tasks, propagates information about nodes throughout the tree, and removes UNSAT node tasks from the task queue or instructs the threads currently handling those tasks to terminate.

Worker threads solve subproblems in the task queue concurrently, utilizing the solver designated for the original problem. A priority queue within the thread pool ensures that subproblems of lower complexity are solved first, with results being relayed to monitor thread for timely updates to the node information in the binary splitting tree.

A. Binary Splitting Tree

The binary splitting tree is primarily updated when Monitor thread receives new result information, which involves two main operations:

- **Inference towards the root node:** If a node is determined to be UNSAT, and its sibling node has also been proven UNSAT, then their UNSAT results can infer that their parent node is UNSAT. This process proceeds up towards the root node.
- **Propagation towards the leaf nodes:** If a node is UNSAT, then all its descendant nodes in that branch are also UNSAT.

B. Dynamic Task Scheduling Strategy

In the parallel process, each node in the tree has four states: unsubmitted, waiting, running, and completed. Initially, all nodes are in the "unsubmitted" state. Master thread sorts the leaf nodes according to (1), then adds them to the task queue where node's state is "waiting". Tasks in the queue are assigned to Worker threads, which update the nodes' states to

"running" before solving. After obtaining the results, Monitor thread retrieves the task IDs and updates the corresponding node's state to "completed".

We present an execution example in Fig. 5 that demonstrates our scheduling method. Five Worker threads are used, and the left side of the figure shows the binary splitting tree at an intermediate stage. After node 6 is completed and its UNSAT result is passed to Monitor thread, it infers node 3 as UNSAT since both nodes 6 and 7 are UNSAT. Node 2, the sibling of node 3, is subsequently pushed into the task queue and picked up by an idle thread. The propagation from node 6 triggers the termination of the corresponding worker for node 10 and marks node 11 as UNSAT. Since there are still idle threads available, Master thread identifies the next unresolved leaf node, node 8, splits it, and appends nodes 12 and 13 to the tree. Both nodes 12 and 13 are then queued for execution, with the idle Worker thread taking on node 12, while node 13 remains in the task queue.

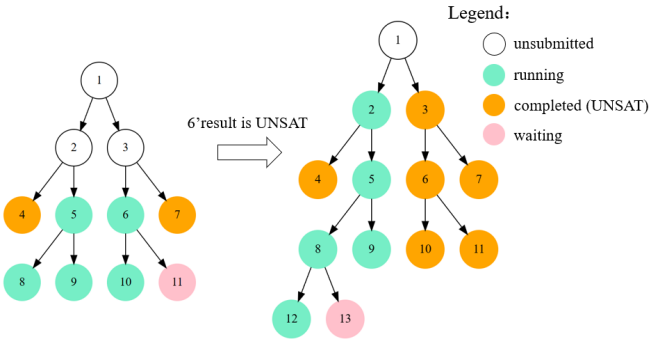


Fig. 5. A running example of our dynamic task scheduling strategy.

V. Experimental Results

In this section, we present the experimental setup and compare PDP-CEC with current representative CEC tools, such as ABC and Hybrid-CEC. Then we identify the factors contributing to the superior performance of PDP-CEC compared to Hybrid-CEC. Finally, we analyze the scalability of PDP-CEC.

A. Experimental Setup

All experiments were performed on a server with 128 Intel Xeon Platinum 8153 cores (2.00 GHz) and 2048 GB of memory, running CentOS 7. PDP-CEC is implemented in C++ and compiled with GNU g++ (version 9.3.1), with all competing methods compiled and executed under the same conditions. Our implementation utilizes the Metis [20] library for graph partitioning and the standard library for parallelism.

The benchmarks used in our experiments are derived from the Integrated Circuit EDA Elite Challenge 2022 [21] with modifications. We select 24 datapath circuits, featuring various sizes and high widths (higher than 32), including multipliers and adders. Each of the 24 instances consists of two functionally equivalent circuits in AIG format and has only a single PO, making parallel method [13] that depends on TFI unsuitable for these benchmarks.

For each experiment, the running time limit for each instance is set to 7200 seconds, and 'TO' means timeout. The unit of running time is seconds, and experiments were conducted with 128 threads by default.

B. Assessment of Datapath Benchmarks

We compare the performance of PDP-CEC with three other CEC tools in Table I:

- OPCEC: A simplified PDP-CEC version without pre-partitioning or the monitor's task submission strategy.
- ABC-SP: The split-based parallel CEC command in ABC, using "&splitprove -P 100 -L 7" for comparison.
- H-CEC: The Hybrid-CEC, a serial solution for datapath CEC.

TABLE I
Performance Comparison of Different CEC Methods on Datapath Benchmarks

Instance	Gates	PDP-CEC	OPCEC	ABC-SP	H-CEC
dpath_1	735	**9.63**	10.54	120.35	200.56
dpath_2	1034	**417.44**	523.08	1415.43	TO
dpath_3	1037	1386.46	**1172.67**	1377.82	TO
dpath_4	6905	**7.57**	8.97	TO	265.01
dpath_5	8003	**2771.45**	TO	TO	TO
dpath_6	6218	**7.21**	8.65	TO	298.92
dpath_7	7073	**12.39**	18.33	TO	1552.04
dpath_8	7994	**2917.33**	TO	TO	TO
dpath_9	6227	**7.25**	8.30	TO	156.46
dpath_10	7076	**14.32**	17.94	TO	490.71
dpath_11	7997	**4162.91**	TO	TO	TO
dpath_12	7079	**15.04**	17.86	TO	585.54
dpath_13	8000	**3180.89**	TO	TO	TO
dpath_14	8063	**4138.11**	TO	TO	TO
dpath_15	8087	**4158.53**	TO	TO	TO
dpath_16	8078	**4940.52**	TO	TO	TO
dpath_17	8066	**3926.71**	TO	TO	TO
dpath_18	8075	**4774.12**	TO	TO	TO
dpath_19	8072	**4134.73**	TO	TO	TO
dpath_20	8081	**4863.37**	TO	TO	TO
dpath_21	8084	**5108.63**	TO	TO	TO
dpath_22	8069	**3891.60**	TO	TO	TO
dpath_23	8090	**3788.22**	TO	TO	TO
dpath_24	9134	TO	TO	TO	TO
Best		22	1	0	0
Solved		23	9	3	7

The comparison results in Table I show that PDP-CEC is consistently the fastest among all methods. Compared to H-CEC, PDP-CEC solves 23 out of 24 instances within 7200 seconds, about 3x more than H-CEC. For instances completing within the time limit, PDP-CEC achieves speedups ranging from 5.11x to 125.27x over H-CEC, with the greatest benefits observed in larger circuits.

PDP-CEC also outperforms ABC-SP, whose search-space splitting strategy resembles PDP-CEC's. However, with the *NodeSelecting* algorithm, PDP-CEC generates more efficient subproblems, while its dynamic task scheduling strategy enables faster subproblem resolution, which results in more solved instances and reduced running time compared to ABC-SP. Finally, PDP-CEC outperforms OPCEC on all other in-

stances except dpath_3, where OPCEC performs better. This is because the key subproblems for dpath_3 are located in the lower levels of the binary splitting tree, a common occurrence in smaller circuits. Overall, the results highlight PDP-CEC's advantage due to its effective task scheduling and partitioning approach.

C. Comparative Analysis of PDP-CEC and Hybrid-CEC

TABLE II
PERFORMANCE COMPARISON OF SAT AND EPS BETWEEN H-CEC AND PDP-CEC WITH 128 THREADS.

Instance	Gates	T_{SAT}		T_{EPS}	
		PDP-CEC	H-CEC	PDP-CEC	H-CEC
dpath_14	8063	4120.48	TO	9.33	331.04
dpath_15	8087	4138.71	TO	9.82	339.92
dpath_16	8078	4939.53	TO	10.99	836.79
dpath_17	8066	3906.93	TO	9.78	423.97
dpath_18	8075	4754.60	TO	9.52	452.88
dpath_19	8072	4113.94	TO	10.79	1685.97
dpath_20	8081	4843.47	TO	9.90	1570.13
dpath_21	8084	5088.58	TO	10.05	1369.43
dpath_22	8069	3871.51	TO	10.09	355.38
dpath_23	8090	3768.02	TO	10.20	454.33

PDP-CEC also makes acceleration on Exact Probability-based Simulation (EPS), which is the key method in H-CEC. Table II shows the performance comparison between PDP-CEC and H-CEC on SAT and EPS. In this experiment, we extract 10 instances from the benchmark set, which H-CEC cannot solve. These results highlight that H-CEC struggles to handle these instances because of its insufficient SAT component. In contrast, PDP-CEC successfully resolves pairs identified as intractable SAT problems, which is critical for solving more complex datapath circuits. Additionally, PDP-CEC achieves speedups ranging from 35.48x to 158.60x over H-CEC for EPS. While the speedup is expected to be linear with the number of threads, slight deviations occur as a result of the overhead associated with the thread pool and graph partitioning. Remarkably, superlinear speedups are possible with sufficient system support.

D. Scalability Analysis of PDP-CEC

TABLE III
EXECUTION TIME WITH VARIOUS THREADS FOR PDP-CEC.

Instance	PDP-CEC				
	8t	16t	32t	64t	128t
dpath_1	33.62	26.75	16.38	13.74	9.63
dpath_4	16.49	8.52	7.25	8.30	7.57
dpath_6	14.49	8.67	9.92	8.69	7.21
dpath_7	48.59	31.23	20.84	19.37	12.39
dpath_9	19.03	11.12	9.45	8.29	7.25
dpath_10	55.20	31.64	19.47	18.92	14.32
dpath_12	56.26	30.20	22.30	19.94	15.04
average	34.81	21.16	15.09	13.89	10.49

We assess the scalability of PDP-CEC by varying the thread count from 8 to 128. The instances selected for this experiment, listed in Table III, were solved by PDP-CEC across

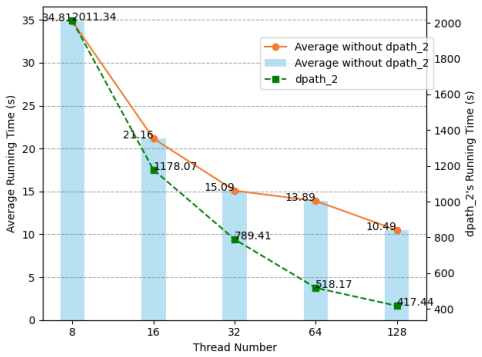


Fig. 6. The line-bar chart of execution time with various threads for PDP-CEC. Average data is come from Table III. The instance dpath_2 is separately illustrated.

different thread configurations. To minimize estimation errors caused by the extended running time of instances dpath_2, we report the average running time of other instances in Fig. 6, with dpath_2 shown separately as a distinct line. At lower thread counts, both average line and dpath_2 line show a near-linear speedup. However, the running time converges more gradually as threads increase, due to additional overhead from thread management and graph partitioning. In summary, the results showcase PDP-CEC as a scalable solution with strong potential for adaptation to distributed systems, supporting application to even larger datapath circuits.

VI. Conclusion

This study introduces a novel Parallel Dynamic Partitioning CEC approach, PDP-CEC, breaking the limitations of existing CEC methods, particularly in solving datapath CEC problems. By exploiting the inherent properties of circuits, PDP-CEC employs search space splitting in circuit partitioning for datapath CEC problem. An algorithm is designed in PDP-CEC for selecting assigned nodes and generate massive subproblems efficiently without dependencies. PDP-CEC applies a dynamic task scheduling strategy to manage the parallel execution of subproblems, preventing stalled workers and ensuring efficient resource utilization. Compared to three other CEC tools, PDP-CEC demonstrates superior performance, solving more instances and reducing overall running time. In conclusion, PDP-CEC demonstrates strong capability in solving datapath CEC problems, providing a scalable and efficient solution that surpasses the performance limitations of existing methods.

Future research will aim to extend PDP-CEC to distributed systems, further enhancing scalability and enabling applications on larger datapath circuit designs.

Acknowledgment

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300 and the Postdoctoral Fellowship Program (Grade B) of China Postdoctoral Science Foundation, Grant No. GZB20240760.

REFERENCES

[1] M. Fujita, "Toward unification of synthesis and verification in topologically constrained logic design," *Proc. IEEE*, vol. 103, no. 11, pp. 2052–2060, Nov. 2015.

[2] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. De Micheli, "Structural rewriting in XOR-majority graphs," in *Proc. 24th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2019, pp. 663–668.

[3] Brayton, Robert K., Gary D. Hachtel, and Alberto L. Sangiovanni-Vincentelli. "Multilevel logic synthesis." Proceedings of the IEEE 78.2 (1990): 264-300.

[4] L. Amarú, F. Marranghello, E. Testa, C. Casares, V. Possani, J. Luo, P. Vuillod, A. Mishchenko, and G. De Micheli, "SAT-sweeping enhanced for logic synthesis," in *Proc. 57th ACM/IEEE Design Automation Conf. (DAC)*, 2020, pp. 1–6.

[5] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2894–2903, Dec. 2006.

[6] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2008, pp. 234–241.

[7] V. Sklyarov, I. Skliarova, and A. Kabulov, "Hardware accelerators for solving computationally intensive problems over binary vectors and matrices," in *AIP Conf. Proc.*, vol. 2781, no. 1, 2023.

[8] S. Harris and D. Harris, *Digital Design and Computer Architecture*, 2015. Morgan Kaufmann.

[9] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2006, pp. 836–843.

[10] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *IEEE/ACM Int. Conf. on Computer Aided Design, 2004. ICCAD-2004.*, pp. 50–57, 2004.

[11] A. Mishchenko, "ABC: A system for sequential synthesis and verification," *URL http://www.eecs.berkeley.edu/alanmi/abc*, vol. 17, 2007.

[12] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," ERL Tech. Rep., 2005.

[13] V. N. Possani, A. Mishchenko, R. P. Ribas, and A. I. Reis, "Parallel combinational equivalence checking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 3081–3092, Oct. 2019.

[14] H. Wang and S. Beamer, "RepCut: Superlinear parallel RTL simulation with replication-aided partitioning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 572–585.

[15] M. Zhao, S. Cai, and Y. Qian, "Distributed SMT Solving Based on Dynamic Variable-Level Partitioning," in *International Conference on Computer Aided Verification*, 2024, pp. 68–88.

[16] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, 1998.

[17] P. Bjesse and A. Boralv, "DAG-aware circuit compression for formal verification," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pp. 42–49, 2004: IEEE.

[18] Z. Chen, X. Zhang, Y. Qian, Q. Xu, and S. Cai, "Integrating exact simulation into sweeping for datapath combinational equivalence checking," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–9.

[19] S.-C. Wu, C.-Y. Wang, and J.-A. Hsieh, "The potential and limitation of probability-based combinational equivalence checking," in *2006 15th Asian Test Symposium*, pp. 103–108, 2006: IEEE.

[20] G. Karypis, "Metis: A Graph Partitioning Library," 2024. [Online]. Available: https://github.com/KarypisLab/METIS. [Accessed: Oct. 9, 2024].

[21] ICISC, "ICISC EDA Challenge," 2022. [Online]. Available: https://eda.icisc.cn/. [Accessed: Oct. 9, 2024].

LeakyDSP: Exploiting Digital Signal Processing Blocks to Sense Voltage Fluctuations in FPGAs

Xin Zhang[1 2 4†], Jiajun Zou[1 2†], Yi Yang[1 2], Qingni Shen[1 2*],
Zhi Zhang[3*], Yansong Gao[3], Zhonghai Wu[1], Trevor E. Carlson[4]
[1]School of Software and Microelectronics, Peking University
[2]PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University
[3]The University of Western Australia
[4]National University of Singapore
Email: {zhangxin00, jjzou2002, tkike}@stu.pku.edu.cn, qingnishen@ss.pku.edu.cn,
{zhi.zhang, garrison.gao}@uwa.edu.au, wuzh@pku.edu.cn, tcarlson@comp.nus.edu.sg

***Abstract*—In recent years, cloud providers are dedicated to enabling FPGA multi-tenancy to improve resource utilization, but this new sharing model introduces power side-channel threats, where attackers detect voltage fluctuations from co-located circuits. This paper proposes LeakyDSP, a novel on-chip sensor that maliciously configures DSP blocks to sense fine-grained voltage fluctuations but is overlooked by existing studies. Our experimental results show that LeakyDSP achieves high sensitivity to voltage fluctuations and strong robustness to different placements. Besides, we apply LeakyDSP to extract full AES keys with 25 k-78 k traces and build covert channels with a high transmission rate of 247.94 bit/s.**

I. Introduction

In recent years, leading commercial cloud service providers such as Amazon AWS [3], Alibaba [2], and Microsoft Azure [24] have introduced remote access to high-performance Field Programmable Gate Arrays (FPGAs) [6]. To maximize FPGA utilization efficiency, both academia and industry have focused their efforts on enabling the co-residence of FPGA circuits by independent tenants, referred to as multi-tenant FPGAs. For example, researchers have proposed different end-to-end frameworks to virtualize FPGA resources in the cloud environment [15], [18], [20], [21], [33], [38]. Besides, Xilinx has integrated Stacked Silicon Interconnect [36] and Dynamic Function eXchange [35] into advanced FPGAs to facilitate multi-tenant FPGAs.

While this new sharing model has attracted considerable attention from both academia and industry, it introduces new security risks, notably power side-channel leakages [13], [39], [42]. Specifically, different users' circuits on a multi-tenant FPGA share the same power delivery network (PDN) [6]. Consequently, transient voltage fluctuations in the shared PDN, induced by computing activities of a victim circuit, can be sensed by a co-located attack circuit. Leveraging this capability to sense voltage fluctuations, various remote power side channel attacks have been proposed to extract cryptographic keys [14], [32], [42], build covert channels [42], fingerprint computation workloads [13], and steal DNN model architectures [41] or inputs [25].

To sense voltage fluctuations, prior studies have proposed various malicious circuit designs that exploit combinations of traditional FPGA logic resources, i.e., Look-up Tables (LUTs) [14], [31], [32], [42], carry chains (CARRY) [10], routing resources [28], and flip-flops (FFs). For instance, time-to-digital converters (TDCs) [10] chain CARRY resources as a long delay line and connect each CARRY output precisely to an FF within the same slice. Besides, the ring oscillator (RO) circuits configure LUTs to implement *combinational loop* that generates oscillating signals, which are then captured by FFs.

In response, several countermeasures have been proposed to prevent the deployment of such circuits on multi-tenant FPGAs [4], [10], [30]. First, as TDC circuits require a long delay chain placed in continuous vertical areas, they can be mitigated by restricting access to such areas [10]. Second, as RO circuits inherently contain combinational loops and some TDC circuits [12] include latch structures, bitstream checks targeting these key structures have been introduced [4], [30]. Although there are new sensor designs [14], [28], [31], [32] that bypass these placement constraints and bitstream checks, all these designs are crafted by configuring traditional FPGA logic resources that are in the primary focus of bitstream checks [17]. Outside the scope of traditional FPGA resources, the DSP blocks dedicated to complex mathematical operations remain unexplored. To this end, we are interested in the following research questions:

Can voltage fluctuations be sensed through maliciously configuring DSP blocks? If so, what attacks can be launched, and what information might be leaked?

In this paper, we propose LeakyDSP, a new on-chip sensor that is built by configuring DSP blocks to sense fine-grained voltage fluctuations without requiring any traditional FPGA logic resources. In a multi-tenant FPGA environment, DSP blocks are partitioned into separate virtual areas, allowing each tenant exclusive access to their DSP blocks [21]. Our key observation is that each dedicated DSP block has three

† Co-first authors. * Corresponding authors.

main sub-components, i.e., a pre-adder, a multiplier, and an arithmetic logic unit (ALU). The signal delay incurred by either sub-components is sensitive to voltage, resulting in different signal propagation delay between output and input of a DSP block when different voltage fluctuations occur.

To this end, we craft LeakyDSP as follows. *First*, we amplify the effect of voltage variations on the output of DSP blocks by maximizing the signal propagation delay. This is achieved by creating a chain within a single DSP block that links the aforementioned sub-components without intermediate registers. Then, multiple configured DSP blocks are connected in series to accumulate their signal delays. To capture and store the resulting flipped bits, the output register of the final DSP block is instantiated. *Last*, we employ the `IDELAY` primitive to dynamically adjust the initial delay of DSP blocks after this sensor is deployed onto an FPGA board. This post-deployment calibration enhances LeakyDSP's robustness to different FPGA placements.

To demonstrate the viability of LeakyDSP, we first characterize the sensitivity of LeakyDSP in two experiments. In each experiment, the TDC [10] (i.e., the most studied circuits to sense voltage fluctuations) is re-implemented as a baseline and compared against LeakyDSP regarding their sensitivity. Specifically, we put LeakyDSP (resp., TDC) under a given placement and measure its sensitivity to voltage fluctuations caused by different victim activities. *Further*, considering that PDN influences its circuits unfairly, we evaluate the impact of spatial proximity to a given victim circuit on LeakyDSP (resp., TDC) by placing it onto different FPGA locations. Both experimental results show that LeakyDSP achieves fine-grained sensitivity to voltage fluctuations and strong robustness to different placements.

We then demonstrate the security implications of LeakyDSP by mounting two end-to-end case studies. First, we perform a correlation power analysis (CPA) against an AES-128 module running on FPGA, with 25 k-78 k traces for extracting the full AES key under 8 different placements of LeakyDSP and 4 operating frequencies of the AES circuit. Second, we use LeakyDSP to establish a covert channel with a high transmission rate of 247.94 b/s and low error rate of 0.24%.

Summary of contributions: The main contributions of this paper are as follows:

- We propose LeakyDSP, a new on-chip sensor design on multi-tenant FPGAs. To the best of our knowledge, we are the first to identify the information leakage through DSP blocks.
- We perform a comprehensive evaluation of LeakyDSP's capabilities. Experimental results show that LeakyDSP achieves high sensitivity to voltage fluctuations and strong robustness to different placements.
- We demonstrate the security implications of LeakyDSP, including extracting full AES keys and building covert channels.

The source code to reproduce the experiments is released at https://github.com/jjzou2002/LeakyDSP.

II. Background and Related Work

A. Power Side Channels on Multi-tenant FPGAs

FPGAs offer significant advantages for customized computation due to their inherent re-programmability and high-performance capabilities [26]. In recent years, the emergence of multi-tenant cloud FPGA environments, attracting prominent interest from both industry and academia [6], promotes flexible and cost-effective access to advanced FPGA functionalities. Specifically, a single FPGA instance is shared among numerous cloud tenants simultaneously, maximizing resource usage efficiency and potentially escalating performance. However, the sharing of an FPGA by multi-tenants, unfortunately, opens the door for various types of side channel leakages, particularly power side channels [13], [14], [28], [31], [32], [42]. Various security implications have been demonstrated via power side channels, including extracting cryptographic keys [14], [32], [42], building covert channels [8], [9], [42], fingerprinting victim computations [13], and stealing DNN model architectures [41] or inputs [25].

One key challenge in these attacks is to craft an on-chip sensor whose output is affected by its own supply voltage and to co-locate it with the victim circuit. In a multi-tenant FPGA scenario, this sensor can sense the voltage fluctuations caused by a victim circuit sharing the same PDN. All existing on-chip sensors exploit the malicious connection of traditional FPGA resources (e.g., LUTs, carry chains, wires, and FFs). As the most studied one, TDC-based circuits [10], [13] use the FPGA carry chain as a delay unit to construct a delay line, with each output connected to an FF that samples the signal traversing through the delay line. Besides, RO [41], [42], VITI [32], PPWM [31], and 1LUTSensor [14] initialize LUT and FF in their own ways. Last, RDS [28] freely places several FFs to abuse the delay incurred by routing resources, e.g., wires.

B. Digital Signal Processing Blocks

As FPGAs have transitioned from prototyping systems to deployable systems (e.g., BrainWave [22] and Catapult [23]), manufacturers have integrated digital signal processing (DSP) blocks into FPGA boards. Unlike traditional FPGA logic resources, which consume substantial area and power for complex mathematical calculations, DSP blocks offer a specialized hardware solution that significantly enhances both performance and efficiency, though at the cost of a degree of re-programmability. Accessed through dedicated hardware

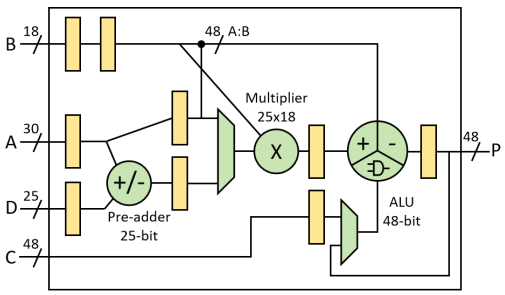


Fig. 1: Structure of the Xilinx `DSP48E1` primitive [27].

primitives, these DSP blocks can be configured for a wide range of arithmetic and logic operations [27].

For instance, starting with the Virtex-6 and 7-series architectures, the `DSP48E1` primitives are dedicated to the efficient processing of high-bitwidth data, enabling complex functions such as multipliers and filters to be implemented without relying on traditional FPGA resources. Figure 1 shows a simplified representation of the `DSP48E1` primitive. It has four inputs with different lengths and is composed of three main sub-components, i.e., a pre-adder, a multiplier, and an arithmetic logic unit (ALU). The pre-adder is a 25-bit addition/subtraction unit that operates on input D and the lower 25 bits of input A. The multiplier multiplies the lower 25 bits of input A or the result of pre-adder with the 18-bit input B. The ALU performs addition, subtraction, and logical operations using the output from the multiplier, input C, concatenated inputs A and B, or the previous output of the DSP block. These sub-components can be flexibly configured to allow the DSP block to perform a range of functions.

III. LeakyDSP

A. Threat Model

We focus on a multi-tenant FPGA scenario where different tenants' circuits are co-located on the same FPGA. Aligned with previous FPGA power side channels [10], [13], [14], [28], [31], [32], our only assumption is that the FPGA tenant applications effectively utilize the shared PDN to optimize resource allocation and bolster overall operational efficiency. To ensure robust logical isolation, each tenant is allocated physically separate FPGA regions to deploy their circuits. The cloud service providers empower tenants with the flexibility to implement a diverse array of FPGA circuits, except those containing combinational loops [4]. Additionally, tenants can define precise placement and routing constraints tailored to the unique requirements of their own designs.

B. LeakyDSP Design

The main idea behind LeakyDSP to sense on-chip voltage fluctuations is to design a malicious DSP function that inherently responds to variations in the supply voltage. Our key observation is that the dedicated DSP blocks in FPGAs are composed of three main sub-components: a pre-adder, a multiplier, and an ALU (discussed in Section II-B). The signal propagation delay through these components is sensitive to voltage changes, leading to signal delay variations between their input and output under different voltage levels.

We note that despite the limitations in configurability compared to traditional FPGA logic resources, DSP blocks still offer sufficient flexibility to construct a voltage-sensitive function. Specifically, each of the inputs for these sub-components can be responded asynchronously without intermediate registers (i.e., flip-flops). If we connect the three components in series, their delay will be accumulated and their computation results will also have dependencies. Further, the DSP architecture supports the cascading of multiple DSP blocks. This capability allows LeakyDSP to be composed of an arbitrary number of DSP blocks, much like how a single TDC instance can utilize an arbitrary number of CARRY structures.

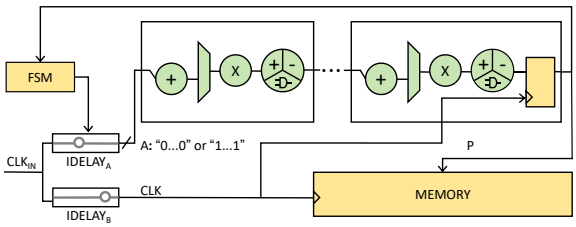


Fig. 2: An overview of LeakyDSP.

Figure 2 shows the overview of the proposed LeakyDSP sensor. Specifically, LeakyDSP contains two parts. First, we carefully configure the DSP blocks to make the signal propagate through the sub-components that are marked as green. The input of the first block (A) consists of multiple bits, and each of the bits equals the delayed CLK_{IN}. Second, we leverage `IDELAY` primitive to introduce a calibration design and dynamically adjust the phase difference between input signal A and the clock signal of the flip-flop within the last DSP block CLK. By doing so, LeakyDSP's output is sensitive to voltage fluctuations and adaptive to different placements.

Malicious DSP function: In this paper, we pick the simplest function that can be instantiated by DSP blocks to analyse the information leakage of LeakyDSP, i.e., $P = A$, where A is varying between '00...0' and '11...1'. To achieve this, we first configure the pre-adder to add a constant '0' to the input (A), and pass the result $(A + 0)$ to the multiplier for further computation. Second, we set the other input of the multiplier to '1', ensuring the multiplication result $(A + 0) \times 1$ remains consistent with the original input A. Third, we configure the ALU as an adder and set the second input as a constant '0', producing a result of $(A + 0) \times 1 + 0$. Last, we configure the output mode based on the position of the DSP block. If it is the last DSP block in a LeakyDSP instance, the output of ALU passes through a register before being stored in memory. Otherwise, the lower bits of output are fed directly into the next DSP block as its input.

Extracting the intensity of the voltage fluctuations: When the PDN is idle, LeakyDSP's outputs remain stable with only a few bit flips. However, in the case of a busy PDN, the number of flipped bits increases in response to fluctuating voltage. Taking `DSP48E1` as an example, its output is 48-bit wide, enabling a fine-grained quantization to voltage levels. To this end, LeakyDSP converts each raw sampled value into a numerical representation based on its Hamming weight [10], [28], referred to as the *readout*.

Calibration: Aligned with existing crafted circuits [10], [14], [28], [31], [32], LeakyDSP contains a calibration part to ensure that the bit-flip number from LeakyDSP's outputs changes accordingly when the voltage in PDN starts fluctuating. To dynamically adjust the initial delay, LeakyDSP incorporates two `IDELAY` components, which are part of the SelectIO resources and designed for signal timing adjustment in FPGA devices. Each `IDELAY` can generate a runtime-adjustable delay of up to

the half of the period of its connected clock signal (i.e., $T/2$) on the signal passing through it. By connecting them to the input signal A and clock signal CLK respectively, LeakyDSP can achieve a phase difference ranging from $-T/2$ to $T/2$ between A and CLK. We adopt a straightforward calibration procedure for LeakyDSP. Specifically, we iteratively increase the initial clock delay by changing the configuration of the two `IDELAY` units, until the readouts of LeakyDSP reach the maximum variation between two consecutive adjustments.

Implementation: Though LeakyDSP can be instantiated with an arbitrary number of used DSP blocks (i.e., n), the optimal choice of n is influenced by the resource budget and sensitivity to voltage fluctuations. In this paper, we select $n = 3$, an empirical value that provides a balance of high sensitivity, acceptable resource usage, and ease of calibration. Besides, we use the `DSP48E1` primitive to map the configuration of DSP blocks on 7-series FPGA boards and `DSP48E2` on Zynq Ultrascale+ FPGAs. For delay configurations, we employ `IDELAYE2` on 7-series FPGAs and `IDELAYE3` on Zynq UltraScale+ FPGAs.

IV. Evaluation

Machine settings: To perform the characterization and AES key extraction attack, we use a Basys3 board connected to a Huawei MateBook 14 laptop via UART and powered through the USB interface. To evaluate the covert channel, we use an ALINX AXU3EGB board, which has the similar architecture as the AWS EC2 instance (i.e., Xilinx UltraScale+ VU9P FPGA) [3]. To deploy hardware code onto the FPGA platforms, we use Vivado version 2020.1 for synthesis, implementation, and bitstream generation.

A. Characterizing LeakyDSP

To show the sensitivity of LeakyDSP to voltage fluctuations, we opt for power virus circuits as a victim to generate varying voltage fluctuations, aligned with [42]. The power virus circuit consists of a number of instances that are implemented as RO circuits. Specifically, within the power virus circuits, each RO circuit consists of a single inverter, an AND gate, and an FF. The output of the inverter is controlled by configuring the enable signal, determining whether it is active to make the output signal frequently switch between '0' and '1'. These switching activities can cause a voltage fluctuation in PDN.

LeakyDSP's sensitivity under different victim activities: Our first experiment is to characterize the readout change of LeakyDSP to various victim activities. Specifically, we create 8,000 instances of power virus to cover over 33.3% routing places of our Basys3 board and activate about 46% of available LUT resources. We then divide them into 8 groups and each group has 1000 evenly-distributed instances. When we activate a varying number of these groups, the voltage varies from 9 levels. For each level, we collect 2,000 readouts from LeakyDSP and take the average of them as the final result to compute Pearson correlation coefficient and regression coefficients. Pearson correlation coefficient is used to assess the strength and direction of the linear association,

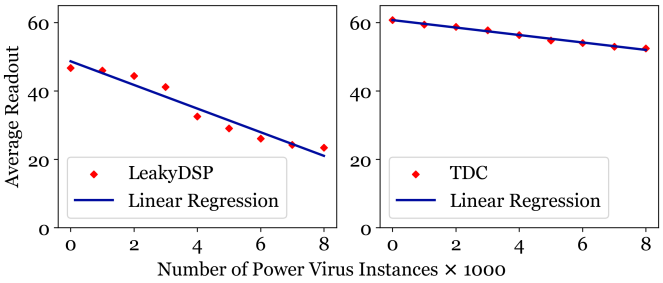


Fig. 3: The sensitivity of LeakyDSP and TDC in the same placement under different victim activities. Note that we implement TDC with 128 FFs, resulting in a 128-bit output, while LeakyDSP produces a 48-bit output.

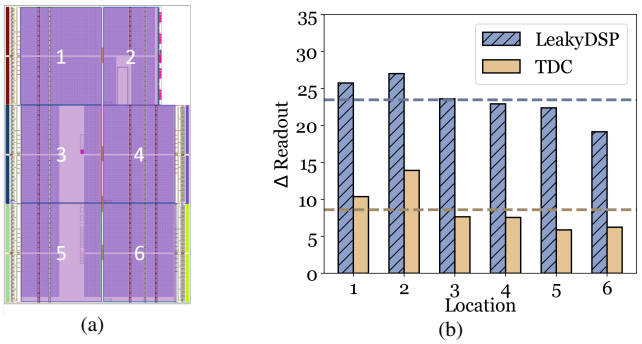


Fig. 4: The sensitivity of LeakyDSP and TDC [10] under different placements. The index as the x-axis in Figure 4(b) corresponds to the region index in Figure 4(a). The dashed line indicates the average value of a sensor.

while the regression coefficient measures the impact of these two values in a linear regression model.

Figure 3 shows the relationship between these readouts (i.e., the number of unflipped bits in LeakyDSP's output) and the number of activated power virus instances. These two values of LeakyDSP exhibit a roughly linear relationship with a Pearson correlation coefficient of -0.974, while TDC achieves -0.996. The close proximity of these correlation coefficients to -1 indicates a strong linear relationship, demonstrating that all the evaluated voltage levels fall within the observable range of LeakyDSP and can be accurately estimated using its readout. Furthermore, compared with the most studied on-chip sensor (i.e., TDC), LeakyDSP achieves a high regression coefficient of -3.45, while TDC achieves -1.09. This indicates that LeakyDSP provides finer granularity under the given sensor placement.

LeakyDSP's sensitivity under different placements: In a multi-tenant FPGA, spatial proximity between a crafted circuit and a victim's circuit is beyond the attacker's control. As such, we characterize the readout change of LeakyDSP with respect to their spatial proximity to victim logic that consumes power. Specifically, we instantiate 8,000 instances of the power virus, and constrain the placement of the power virus to region 1 and region 2 in Figure 4(a). Subsequently, we put LeakyDSP

(resp., TDC) in each of the 6 FPGA regions via a `Pblock` constraint with all the 8,000 power virus instances off and on. We sample 2,000 readouts under each setting and calculate their average.

As shown in Figure 4, although LeakyDSP is influenced by the unevenness of the FPGA's power supply design, it can still sense the given voltage fluctuation under all the above placements. Specifically, LeakyDSP achieves the best performance when it is placed in region 2. Further, when LeakyDSP is placed under the worst-case placements (i.e., region 5 and region 6 which are far away from the victim circuit), its readouts are still sensitive to voltage fluctuations.

B. Stealing Cryptographic Keys

As the most widely adopted symmetric-key cryptographic algorithm, AES has been implemented in different devices, including hardware circuits on FPGA platforms [1], [19], [37]. However, prior work has demonstrated that a remote attacker can extract the full secret key of AES circuits via CPA attacks [7]. These attacks leverage the Pearson Correlation Coefficient [5] to detect linear dependencies between voltage fluctuations during AES circuit execution and hypothetical power calculated from the ciphertext with a guessed key. To achieve this, attackers first use the Hamming weight and Hamming distance power models to estimate the power consumption for each possible key value. Then, they use their own side channel to obtain the power-related information. Last, they sort each guessed value according to its correlation coefficient to determine the ground-truth key value.

Experimental setup: Our end-to-end attack is built upon the open-source code of RDS [28], which implements an AES-128 module [1] that co-resides with the attack circuit on the Basys3 board. By default, the clock frequencies of the attack circuit and the AES are set to 300 MHz and 20 MHz, respectively. Please note that 20 MHz is exactly aligned with the victim setting in VITI [32] and RDS [28], and we just aim to show the general possibility of leakage. Unless otherwise stated, we collect 60 k power side-channel traces for each setting. In each trace collection, we transmit the key `K` and the plaintext `PT` to the AES core, and record the LeakyDSP's readouts obtained during AES encryption. To avoid plaintext repetition, we employ the current ciphertext as the subsequent plaintext. To simplify the process, we utilize the start encryption signal to trigger the collection of a sensor trace. To evaluate the performance of LeakyDSP, we run the key extraction attack under 8 placements and 4 different AES clock frequencies.

Aligned with previous AES key recovery attacks [14], [28], [31], [32], we use the key rank (KR) estimation metric, which evaluates the number of key candidates that an attacker needs to test. Specifically, the key rank is presented as a narrow range with an upper and lower bound, due to the bounded error in current key rank estimation algorithms. For instance, in the absence of any side-channel information, the upper key rank corresponds to the entire key space, which is 2^{128} for AES-128. Conversely, when the full key is compromised, the lower bound of the KR reduces to one, while the upper bound tightly constrains the range, indicating that the correct key can be deduced with minimal guessing. In such scenarios, attackers can arrange each guessed byte value based on correlation coefficient to obtain the genuine AES key. To speed up computation, we employed the open-source analysis tool [7] for conducting our CPA analysis on an NVIDIA GeForce RTX 3090 GPU.

TABLE I: Number of traces required to break the full AES 128-bit key for different sensor placements.

Setting	Traces to Extract Full Key (×1000)									
	P1	P2	P3	P4	P5	P6	P7	P8	Average	Baseline
LeakyDSP	55	43	58	28	48	25	37	41	41	51

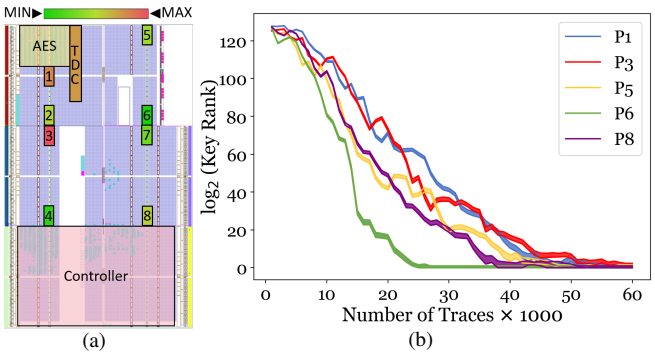


Fig. 5: Key rank estimation for LeakyDSP. The curves in Figure 5(b) are named after the number on each placement shown in Figure 5(a).

Experimental result: Figure 5 shows the average upper and lower bounds of the key rank estimation metric to evaluate the impact of the placement of LeakyDSP. Figure 5(a) is a color gradient that rates all 8 placements based on their key rank with 20 k power traces, revealing that the performance of a power analysis attack relies on the placement of LeakyDSP. This observation aligns with the findings reported in [28], [32], where the location-dependent sensitivity is attributed to the non-uniformity of the PDN across the FPGA board. Moreover, Figure 5(b) shows the key rank variability of LeakyDSP across 5 selected placements out of the total 8 placements, including the best and worst-case placements, as well as the placement closest to the victim circuit. Each placement is identified by its corresponding number, as indicated adjacent to them in Figure 5(a). The full results are summarized in Table I. Compared to TDC, which requires 51 k traces to extract the full key, LeakyDSP achieves a comparable efficiency with 25 k-58 k traces needed. We do not claim that our LeakyDSP is better than TDC in terms of the attack performance, as we only evaluate TDC in one setting. As our LeakyDSP does not use any traditional logic resources inside configurable logic blocks, we cannot place LeakyDSP and TDC in the same position for a solid comparison [28], [32].

We then evaluate the impact of the AES frequency on our attack, utilizing the best-case placement for attackers (i.e.,

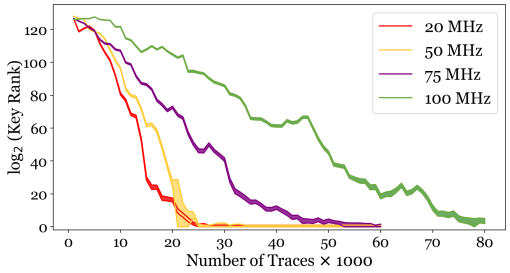


Fig. 6: The impact of the AES frequency on our attack. The efficiency of key extraction decreases as the frequency of the victim circuit increases.

P6). Figure 6 illustrates the corresponding key rank outcomes. We note that the performance of the attack is dependent on the frequency. The efficiency of our key extraction decreases as the frequency of the victim circuit increases. At a high frequency of 100 MHz, the 128-bit AES key cannot be fully recovered even with all 60 k traces. To address this, we collect an additional 20 k traces, achieving full key recovery with a total of 78 k traces.

C. Establishing Covert Channels

Aligned with existing work [9], [12], we apply the covert channel for assessing the bandwidth of our side channels, where the attacker has complete control over both the sender and receiver. We assume the colluding sender and receiver operate on the same FPGA. To induce the voltage fluctuations for the sender, we opt for power virus circuits as to generate varying voltage fluctuations. To transmit a '1', the sender sets its enable signals to '0', making all the power virus instances idle. To transmit a '0', a sender activates all the controlled power virus instances to plunder the voltage of receiver circuits. The receiver is a LeakyDSP-based circuit and loops to monitor voltage level via its readouts. By comparing the averaged readouts to a predefined threshold, the receiver distinguishes between the transmitted bits, '0' or '1'.

Experimental setup: To assess the performance of our covert channel, we vary the time for sending one bit from 2 ms to 7.5 ms and send 10 kb of random data in each configuration. Besides, we employ two metrics that are widely used in previous covert channels [40]: transmission rate (TR) and bit error rate (BER). TR measures the amount of data transmitted per unit time, while BER represents the proportion of received bits incorrectly interpreted relative to the sent bits.

Experimental result: Figure 7 shows the end-to-end results across 10 runs. The experimental results show a negative correlation between the BER and the time for sending a bit. When the sending time is greater than 3.5 ms, the BER stabilizes below 1%. When the sending time is less than 3 ms, the BER gradually increases with the decreasing of sending time due to the limited accuracy of the receiver in distinguishing data. The smaller the time for sending a bit, the greater the TR. So we recommend the sending time of 4 ms with a BER of 0.24%. With such setting, the TR of a single transmission is 247.94 b/s.

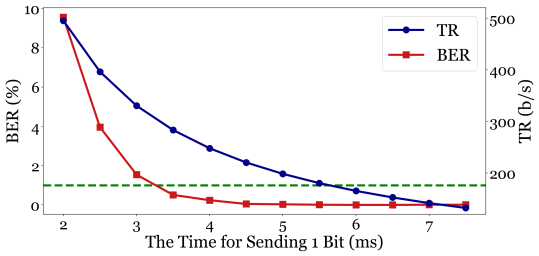


Fig. 7: The performance impact of timing parameters on LeakyDSP-based covert channel.

V. Discussion

Mitigation: Similar to existing on-chip sensors, LeakyDSP relies on precise voltage information from a shared PDN. While enhancing the PDN could mitigate such attacks, this approach is limited by current production technologies [29], [34] and does not protect already-deployed products. An alternative countermeasure is to inject noise into the PDN [11], [16], which obscures power patterns but introduces performance overhead. Besides, the developers can modify their circuits as constant-power implementation with some efforts. Last, after identifying the threats from DSP blocks, cloud providers can prevent the deployment of LeakyDSP by enforcing synchronized inputs or mandatory timing checks on DSP configurations. However, we note that both approaches limit benign tenants' flexibility in DSP reconfiguration. Besides, timing violation checks can be bypassed using programmable clock-generating circuits, as discussed in TDC [10] and RDS [28].

Future work: This paper demonstrates that DSP blocks can indeed be exploited for power analysis attacks through a comprehensive evaluation on two FPGA boards with different architectures. However, while LeakyDSP's readouts vary with voltage levels monotonously, this change is not absolutely uniform. Due to the *black-box* nature of DSP blocks' internal design, we cannot entirely explain LeakyDSP's leakage behavior at this stage. Besides, the number of DSPs within a single LeakyDSP sensor is an empirical value ($n = 3$). We leave the exploration of DSP operating principles and the optimal selection of n as future work.

VI. Conclusion and Acknowledgments

In this paper, we propose LeakyDSP, a novel on-chip sensor that maliciously configures DSP blocks to sense voltage fluctuations on multi-tenant FPGAs. Our characterization shows that LeakyDSP is highly sensitive to voltage fluctuations and robust across different placements. The viability of LeakyDSP is demonstrated by two case studies, which extract full AES keys under various placements of LeakyDSP and operating frequencies of victim circuits, and build FPGA-to-FPGA covert channels with a high transmission rate of 247.94 b/s.

We thank the anonymous reviewers for their feedback. In addition, we thank Yuhui Zhang for her valuable comments that helped us to improve the work. This work is supported by the National Key R&D Program of China (No. 2022YFB2703301) and the Singapore Ministry of Education (Grant No. T1251RES2403).

REFERENCES

[1] AIST and T. University, "AES Encryption Core," 2007. [Online]. Available: http://www.aoki.ecei.tohoku.ac.jp/crypto/

[2] Alibaba. (2024) Compute optimized instance families with FPGAs. [Online]. Available: https://www.alibabacloud.com/help/en/fpga-as-a-service/product-overview/fpga-accelerated-compute-optimized-instance-families

[3] Amazon, "Amazon EC2 F1 Instances," 2024. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1

[4] Amazon Web Services. (2024) AWS EC2 FPGA HDK+SDK errata. [Online]. Available: https://github.com/aws/aws-fpga/blob/master/ERRATA.md

[5] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," *Noise reduction in speech processing*, pp. 1–4, 2009.

[6] G. Dessouky, A.-R. Sadeghi, and S. Zeitouni, "SoK: Secure FPGA Multi-Tenancy in the Cloud: Challenges and Opportunities," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021.

[7] H. Gamaarachchi, H. Ganegoda, and R. Ragel, "The A to Z of building a testbed for power analysis attacks," in *International Conference on Industrial and Information Systems*, 2015.

[8] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Reading between the dies: Cross-SLR covert channels on multi-tenant cloud FPGAs," in *IEEE 37th International Conference on Computer Design (ICCD)*, 2019, pp. 1–10.

[9] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *IEEE Symposium on Security and Privacy*, 2020, pp. 1728–1741.

[10] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1007–1010.

[11] O. Glamočanin, A. Kostić, S. Kostić, and M. Stojilović, "Active Wire Fences for Multitenant FPGAs," in *International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2023, pp. 13–20.

[12] D. R. E. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-Based Covert Channels Using FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, 2021.

[13] M. Gobulukoglu, C. Drewes, W. Hunter, R. Kastner, and D. Richmond, "Classifying Computations on Multi-Tenant FPGAs," in *Design Automation Conference*, 2021, pp. 1261–1266.

[14] D. Jayasinghe, B. Udugama, and S. Parameswaran, "1LUTSensor: Detecting FPGA Voltage Fluctuations using LookUp Tables," *Cryptographic Hardware and Embedded Systems*, pp. 51–86, 2024.

[15] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on FPGAs?" in *USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 991–1010.

[16] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design*, 2019, pp. 1–8.

[17] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems*, 2019.

[18] J. Landgraf, T. Yang, W. Lin, C. J. Rossbach, and E. Schkufza, "Compiler-driven FPGA virtualization with SYNERGY," in *Architectural Support for Programming Languages and Operating Systems*, 2021.

[19] R. Lumbiarres-López, M. López-García, and E. Cantó-Navarro, "Hardware Architecture Implemented on FPGA for Protecting Cryptographic Keys against Side-Channel Attacks," *IEEE Transactions on Dependable and Secure Computing*, pp. 898–905, 2018.

[20] J. Ma, G. Zuo, K. Loughlin, X. Cheng, Y. Liu, A. M. Eneyew, Z. Qi, and B. Kasikci, "A Hypervisor for Shared-Memory FPGA Platforms," in *Architectural Support for Programming Languages and Operating Systems*, 2020, p. 827–844.

[21] J. M. Mbongue, D. T. Kwadjo, A. Shuping, and C. Bobda, "Deploying Multi-tenant FPGAs within Linux-based Cloud Infrastructure," *ACM Trans. Reconfigurable Technol. Syst.*, 2021.

[22] Microsoft. (2024) Project Brainwave. [Online]. Available: https://www.microsoft.com/en-us/research/project/project-brainwave/pricing/details/machine-learning/

[23] ——. (2024) Project Catapult. [Online]. Available: https://www.microsoft.com/en-us/research/project/project-catapult/

[24] Microsoft Azure. (2024) Machine learning. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/machine-learning/

[25] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Remote Power Side-Channel Attacks on BNN Accelerators in FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition*, 2021.

[26] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf Inference Benchmark," in *International Symposium on Computer Architecture*, 2020, pp. 446–459.

[27] B. Ronak and S. A. Fahmy, "Mapping for Maximum Performance on FPGA DSP Blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[28] D. Spielmann, O. Glamočanin, and M. Stojilović, "RDS: FPGA Routing Delay Sensors for Effective Remote Power Analysis Attacks," *Cryptographic Hardware and Embedded Systems*, 2023.

[29] B.-H. Su, J.-S. Tang, H.-J. Lee, and C.-B. Tzeng, "Noise Analysis and Improvement of Power Supply Network Based on Power Integrity," in *International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*, 2023, pp. 317–320.

[30] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, vol. 55, no. 11, pp. 640–642, 2019.

[31] B. Udugama, D. Jayasinghe, H. Saadat, A. Ignjatovic, and S. Parameswaran, "A power to pulse width modulation sensor for remote power analysis attacks," *Cryptographic Hardware and Embedded Systems*, pp. 589–613, 2022.

[32] Udugama, Brian and Jayasinghe, Darshana and Saadat, Hassaan and Ignjatovic, Aleksandar and Parameswaran, Sri, "VITI: A tiny self-calibrating sensor for power-variation measurement in FPGAs," in *Cryptographic Hardware and Embedded Systems*, 2022.

[33] Z. Wang, G. Zhu, Y. Liu, Y. Chang, K. Zhang, and M. Chen, "XUNI: Virtual Machine Abstraction for Self-contained and Multi-tenant Cloud FPGAs," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024.

[34] D. Xiao, X.-j. Li, Y. Li, H. Lin, J. Qian, and C. Hu, "The Screening Method of Board-Level Decoupling Capacitors for Multi-Power Distribution Network," in *Chinese Control Conference*, 2023, pp. 7111–7116.

[35] Xilinx, "Vivado design suite user guide: Dynamic function exchange (ug909)," 2024. [Online]. Available: https://docs.xilinx.com/r/en-US/ug909-vivado-partial-reconfiguration/Introduction-to-Dynamic-FunctioneXchange

[36] Xilinx, "Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency," 2024. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp380_Stacked_Silicon_Interconnect_Technology

[37] J. Zambreno, D. Nguyen, and A. Choudhary, "Exploring area/delay tradeoffs in an AES FPGA implementation," in *International Conference on Field Programmable Logic and Applications*, 2004, pp. 575–585.

[38] Y. Zha and J. Li, "Virtualizing FPGAs in the Cloud," in *Architectural Support for Programming Languages and Operating Systems*, 2020.

[39] F. Zhang, Z. Wang, H. Shen, B. Yang, Q. Wu, and K. Ren, "DARPT: defense against remote physical attack based on TDC in multi-tenant scenario," in *Design Automation Conference*, 2022, p. 559–564.

[40] X. Zhang, Z. Zhang, Q. Shen, W. Wang, Y. Gao, Z. Yang, and Z. Wu, "ThermalScope: A Practical Interrupt Side Channel Attack Based On Thermal Event Interrupts," in *Design Automation Conference*, 2024.

[41] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote FPGA side-channel analysis," *IEEE Transactions on Information Forensics and Security*, 2021.

[42] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *IEEE Symposium on Security and Privacy*, 2018.

Efficient Continuous Logic Optimization with Diffusion Model

Yikang Ouyang[1*], Xiaofei Yu[1*], Jiadong Zhu[1], Tinghuan Chen[2], Yuzhe Ma[1†]
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]The Chinese University of Hong Kong, Shenzhen
Email: yuzhema@hkust-gz.edu.cn

***Abstract*—The logic synthesis optimization flow is crucial to the quality of results (QoR), which applies a sequence of transformations to a design. Recently, there has been a growing focus on the automatic optimization of synthesis flows to improve QoR, utilizing techniques such as Bayesian optimization and reinforcement learning, which may fall short in efficiency due to the exponentially large search space. In contrast, continuous optimization offers notable efficiency advantages by leveraging the explicit gradient. However, despite its potential, several significant concerns remain to be addressed. On one hand, it is essential to obtain a reliable gradient. On the other hand, a major challenge arises from the fact that searching within a continuous space can yield solutions that deviate from feasible ones. In this paper, we propose an efficient approach to optimize synthesis sequences within a continuous latent space. Specifically, the gradient information is derived from a QoR surrogate model, while the discrepancies between solutions and feasible transformations are minimized by a diffusion model. Experimental results on extensive benchmarks demonstrate that the proposed method not only achieves lower area and delay but also improves efficiency by 5X to 130X, compared with previous methods.**

I. Introduction

Logic synthesis is an important process that transforms circuits from register-transfer level design to gate-level netlist. Previous works usually represent circuits as AIG (And-Inverter Graph) [1] or similar boolean networks like MIG [2]. Then the synthesis process applies a sequence of logic optimization transformations like *rewrite*, *restructure* to circuits [1], [3]. For the same circuit, different sequences can lead to a variation in quality-of-result (QoR) of up to 40% [4]. Thus, it is imperative to optimize those synthesis sequences to improve the quality of synthesized netlists.

Previous works are devoted to optimizing sequences with techniques like reinforcement learning (RL) [5]–[7], multi-arm bandit (MAB) [8], and discrete Bayesian optimization (BO) [9]. Both RL and MAB methods treat the sequence optimization task as a sequential decision-making problem. They explore the probability of which transformation to take by updating the Q-function, policy function, etc. The discrete BO [9] has to sample sequences by discretely altering the transformations for evaluating the acquisition function. While automated methods yield significant QoR improvements over general heuristics, their dependence on extensive discrete searches and iterative nested updates leads to considerable overhead and inefficiency during the optimization process, particularly as the number of transformation options increases and the sequence lengthens. This raises a pertinent question: Can we directly optimize sequences without searching in the discrete space to achieve faster turnaround time?

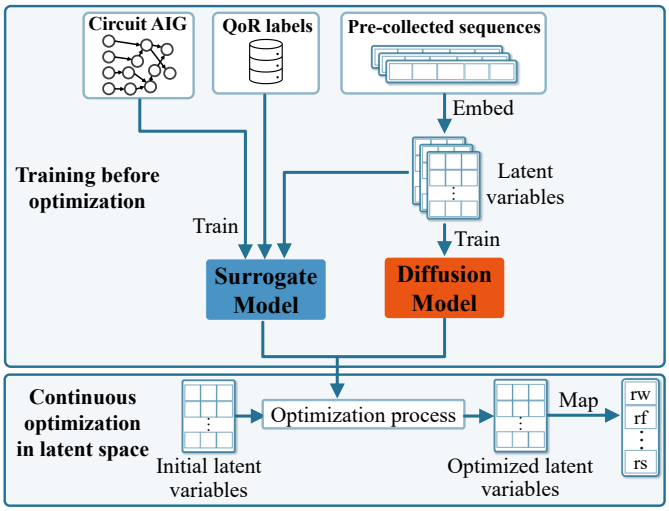


Fig. 1 Overview of the proposed framework. Upper part: Pre-training a surrogate model and a diffusion model. Lower part: Continuous optimization of sequence in latent space.

In contrast to discrete search, continuous search transforms the discrete search space into continuous latent space and leverages numerical optimization techniques like gradient descent to directly optimize the objective function guided by the gradient from a surrogate model [10]. Upon convergence, the final solution is retrieved by mapping the latent representation in continuous latent space back to the original space.

For synthesis sequence optimization, a deep learning-based surrogate model should first be built, which takes the circuit and sequence embeddings as input and predicts the QoR. However, it is observed that solely relying on the gradient from the surrogate model is problematic [11], [12]. Note that the final sequence for synthesis should be retrieved based on the optimized sequence embedding in the latent space. Unfortunately, even though the continuous optimization converges in the first phase, the sequence may hardly be retrieved due to a large discrepancy between the obtained embeddings and the feasible transformation embeddings. Some previous works explored to

*Equal contribution
†Corresponding author

conduct an additional mapping based on a distance metric [13] or a neural network-based decoder [14]. Nevertheless, neither of these methods can achieve the desired performance, as revealed in [15]–[17], since the additional mapping operation is already out of the optimization loop, offering no assurance against performance degradation. Therefore, to facilitate the retrieval of final transformations from continuous space, it is essential to ensure that the embedding discrepancy between the final solutions and feasible transformations is minimized, such that the final optimized sequence can be instantly retrieved.

To address the above challenges, we incorporate a novel generative diffusion model together with a surrogate model into the continuous logic optimization framework. The overview of it is shown in Fig. 1. Particularly, the diffusion model is designed to minimize that discrepancy. Different from the previous two-stage workflows, the proposed method can seamlessly integrate the two stages into a unified stage. Moreover, all the computations are within the optimization loop, which provides a higher level of assurance regarding the quality of the final solution. The discrepancy is minimized explicitly through a series of denoising steps, each of which is performed together with a gradient descent step. Upon convergence, the discrepancy between the optimized embedding and the feasible embedding is minimized, and thus the final optimized sequence can be retrieved instantly without degrading QoR. Our contributions are as follows:

1) To the best of our knowledge, it is the first work to propose a continuous optimization approach for the discrete logic sequence optimization problem.
2) We optimize the QoR together with the discrepancy between optimized embeddings and feasible transformations to achieve a unified framework.
3) We leverage the gradient obtained from a surrogate model for QoR optimization and the denoising steps based on a diffusion model for discrepancy minimization.
4) Experimental results demonstrate that the proposed continuous optimization framework achieves substantial improvements in runtime and QoR compared with previous baseline methods on logic optimization.

II. Preliminaries

Some frequently used notations are shown in TABLE I for a clearer illustration.

A. Logic Synthesis Sequence Optimization

In logic synthesis, a sequence of logic transformations is applied to the circuit to achieve a better QoR. A sequence $\boldsymbol{s}$ with length L is composed of transformations $[s_1, \dots s_L]$, where each transformation s_i belongs to an available set $\mathbb{S}$, i.e., $s_i \in \mathbb{S}$. In our work, $\mathbb{S} = \{\texttt{rw}, \texttt{rwz}, \texttt{rf}, \texttt{rfz}, \texttt{rs}, \texttt{rsz}, \texttt{b}\}$. These transformations have different effects on circuits represented in AIG by reducing the number of nodes and reducing the levels. Although ABC [1] is a relatively fast synthesis tool, the exponentially increasing search space ($|\mathbb{S}|^L$) still hinders efficient optimization. Given the fact that longer sequences may not necessarily lead to better QoR [8], we focus on finding a sequence with a fixed length L that leads to minimal area or delay, or both. It is formulated as:

$$\begin{aligned} \min_{\boldsymbol{s}} \quad & F(\boldsymbol{s}) \\ \text{s.t.} \quad & s_i \in \mathbb{S}, \quad i = 1, 2 \dots, L. \end{aligned} \tag{1}$$

TABLE I Notations in this work.

Notation	Explanation
$\boldsymbol{s}$	A sequence of discrete transformations with length L.
$g(\cdot)$	The function that embeds sequence into latent space.
$\boldsymbol{x}$	The sequence of transformation embeddings.
$\boldsymbol{x}_t$	The sequence of latent variables at denoising step t. We use $t = T$ to denote the start of denoising steps and $t = 0$ as the end to align with notations in diffusion models.
$\hat{F}(\cdot)$	Surrogate model to predict QoR.
$\boldsymbol{\epsilon}_t$	The added noise at the t-th step in diffusion process.
$\boldsymbol{\epsilon}_\theta(\cdot, t)$	The predicted noise at the t-th step by the diffusion model.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$	A Gaussian distribution with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$.
$\boldsymbol{\sigma}_t, \boldsymbol{\alpha}_t$	Some constants dependent on t in the diffusion model.

B. Diffusion Models

Diffusion models have become the mainstream generative models [18]–[20]. Such popularity is mainly due to their extraordinary ability to capture the distribution of training data and then generate data with a similar distribution as the training data. Specifically, diffusion models will convert the training data into pure Gaussian noise by gradually adding Gaussian noise $\boldsymbol{\epsilon}_t$. Then they will learn to restore the data from pure Gaussian through denoising steps by removing noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, t)$ predicted by a neural network parameterized with θ.

III. Methodologies

Our continuous optimization framework involves two models, including a surrogate model to predict QoR and a diffusion model to learn the distribution of feasible transformation embeddings. The optimization process will optimize latent variables with both the gradient from the surrogate model and the denoising update from the diffusion model. Finally, it maps optimized latent variables back to discrete transformations. We illustrate the details in the following sections.

A. Training Surrogate Model

Since QoR after logic synthesis cannot be analytically formulated, a surrogate model is required to bridge the gap. Moreover, it is imperative that the surrogate model can generate the gradient information to enable continuous optimization. Recently, various works have investigated deep learning-based QoR prediction for logic synthesis [4], [21], [22]. Generally, these frameworks will first encode the sequence into an embedding in the latent space, i.e., $\boldsymbol{x} = g(\boldsymbol{s})$. Then a deep neural network is trained to predict the QoR. For each design, given a dataset comprising N data points that include sequences and QoR labels, the surrogate model is trained by minimizing the mean squared error between the predicted values and the corresponding labels as shown in the following equation:

$$\min \mathcal{L} = \frac{1}{N} \sum_{j=1}^{N} (\hat{y}^{(j)} - y^{(j)})^2, \tag{2}$$

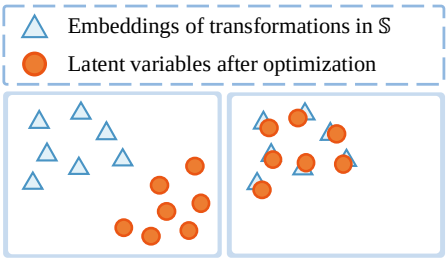


Fig. 2 (a) Latent variables optimized only w.r.t QoR. (b) Latent variables optimized w.r.t QoR and discrepancy.

where the $\hat{y}^{(j)}$ and $y^{(j)}$ are the predicted QoR and the label of the sample j, respectively.

B. Optimizing Sequences in Continuous Space

After a surrogate model is trained to predict the delay and/or area after synthesis, we can minimize the predicted delay or area by finding a sequence embedding in the latent space. For simplicity, we use $\hat{F}(\cdot)$ to represent the surrogate model, and $\boldsymbol{x}$ to represent the embedded sequence to be optimized.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \hat{F}(\boldsymbol{x}). \tag{3}$$

Since the surrogate model is based on a DNN, we can get the gradient by back-propagation. However, relying on gradient may lead to a large discrepancy between optimized latent variables and embeddings of feasible transformations. Fig. 2 presents an illustrative example that visualizes the embeddings in latent space by t-SNE [23]. It poses a great challenge for sequence retrieval, i.e., mapping the embedding in latent space to the real sequence for logic synthesis, given the results with large discrepancy shown in Fig. 2(a).

In this work, we propose to bridge the gap by introducing another auxiliary objective $H(\boldsymbol{x})$ that minimizes the discrepancy. Specifically, minimizing $H(\boldsymbol{x})$ implies that the optimized embedding $\boldsymbol{x}^*$ follows the same distribution as valid transformation embeddings so that we can eliminate the additional yet intricate mapping to retrieve the sequence, as depicted in Fig. 2(b). To this end, we reformulate our optimization problem in the latent space as:

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}[\hat{F}(\boldsymbol{x}) + H(\boldsymbol{x})], \tag{4}$$

where $H(\boldsymbol{x})$ represents the discrepancy between an embedding $\boldsymbol{x}$ and the valid transformation embeddings in latent space.

A widely used option for $H(\boldsymbol{x})$ is the negative log-likelihood (NLL). It is a commonly used objective to be optimized in generative models to ensure the generated data follows the same distribution as the training data. If minimized, the optimized latent variables should be in the distribution of the feasible transformation embeddings and the discrepancies between them are minimized. Ideally, we can directly compute the gradient of the objective in Equation (4) w.r.t $\boldsymbol{x}$, i.e., $\nabla_{\boldsymbol{x}}\hat{F}(\boldsymbol{x}) + \nabla_{\boldsymbol{x}}H(\boldsymbol{x})$.

The gradient $\nabla_{\boldsymbol{x}}\hat{F}(\boldsymbol{x})$ is easy to obtain since it can be computed via back-propagation. However, minimizing NLL through gradient descent leads to tractability issues for computing $\nabla_{\boldsymbol{x}}H(\boldsymbol{x})$. Instead, a variational bound is optimized [24], which is the mainstream approach to train generative models, including diffusion models [18]. Therefore, minimizing $H(\boldsymbol{x})$ can be resolved through training a diffusion model.

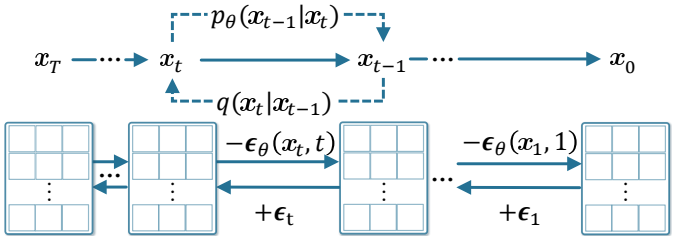


Fig. 3 The diffusion process (from right to left) and denoising process (from left to right) of a diffusion model.

C. Optimization with Diffusion Model

Diffusion models have shown extraordinary abilities in generating high-quality data in the training set distribution by minimizing a variational bound. The diffusion model encompasses a diffusion process and a denoising process as shown in Fig. 3. The diffusion process gradually adds noise to the feasible transformation embeddings as shown in Equation (5) with $\boldsymbol{\alpha}_t$ as some constants.

$$\boldsymbol{x}_t = \sqrt{\boldsymbol{\alpha}_t}\boldsymbol{x}_{t-1} + \sqrt{1-\boldsymbol{\alpha}_t}\boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{5}$$

This diffusion process converts the original data into pure Gaussian noise following the conditional distribution:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\boldsymbol{\alpha}_t}\boldsymbol{x}_{t-1}, \sqrt{1-\boldsymbol{\alpha}_t}\boldsymbol{I}). \tag{6}$$

To generate in-distribution data, the denoising process tries to remove the same scale of Gaussian noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) \sim \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\sigma}_t^2\boldsymbol{I})$ generated with a mean $\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t)$ parameterized by θ:

$$\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\boldsymbol{\alpha}_t}}(\boldsymbol{x}_t - \frac{1-\boldsymbol{\alpha}_t}{\sqrt{1-\bar{\boldsymbol{\alpha}}_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)) + \boldsymbol{\sigma}_t\boldsymbol{z}, \tag{7}$$

where $\boldsymbol{\sigma}_t$ are some constants, $\bar{\boldsymbol{\alpha}}_t = \sum_{s=1}^{t}\boldsymbol{\alpha}_s$ and $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. This process follows the conditional distribution:

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\sigma}_t^2\boldsymbol{I}). \tag{8}$$

The objective of training the diffusion model is to minimize the variational bound over the negative log-likelihood through the denoising process, as shown in Equation (9).

$$\min \mathcal{L}_d = \mathbb{E}[-\log q(\boldsymbol{x}_T) - \sum_{t=1}^{T}\log\frac{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}] \geq \mathbb{E}[-\log p_\theta(\boldsymbol{x}_0)]. \tag{9}$$

Since the diffusion process with $q(\cdot)$ is irrelevant to parameters θ of the neural network, optimizing the loss function in Equation (9) is equivalent to learning a denoising process with parameters θ that can minimize the variational bound over the NLL. In practice, the diffusion model is trained by minimizing

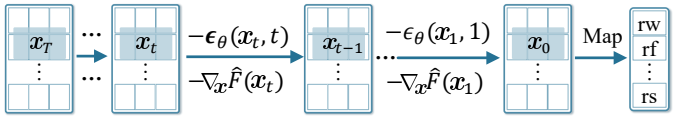


Fig. 4 The optimization process that gradually subtracts gradient from the surrogate model and noise from the diffusion model.

Algorithm 1 Training Diffusion Model

Input: N embedded latent sequences of transformations.
Output: A trained diffusion model parameterized by θ.

1: **repeat**
2: $t \leftarrow \text{Random}(1, T)$, $j \leftarrow \text{Random}(1, N)$.
3: $\bar{\boldsymbol{\alpha}}_t \leftarrow \sum_{s=1}^{t} \boldsymbol{\alpha}_s$.
4: $\mathcal{L}_d \leftarrow \text{Loss}(t, j, \boldsymbol{x}_0^{(j)})$. ▷ Equation (10).
5: $\theta \leftarrow \text{Update}(\theta, \mathcal{L}_d)$.
6: **until** Diffusion model converges.

the difference between the added noise and the predicted noise on the N pre-collected embedded sequences [18]:

$$\min \mathcal{L}_d = \mathbb{E}_{t,j,\boldsymbol{x}_0,\boldsymbol{\epsilon}_t}[||\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\boldsymbol{\alpha}}_t}\boldsymbol{x}_0^{(j)} + \sqrt{1-\bar{\boldsymbol{\alpha}}_t}\boldsymbol{\epsilon}_t, t)||^2], \tag{10}$$

where $\boldsymbol{x}_0^{(j)}$ means the original embedded sequence of sample j. The training details are also illustrated in Algorithm 1. The predicted noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ with a mean $\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t)$ that dependent on θ is updated by gradient descent on computed loss as shown in Line 5.

Given a well-trained diffusion model, the generated latent variable $\boldsymbol{x}_0$ through the denoising process should be in the distribution of embeddings of feasible transformations. The discrepancies between them in the latent space are also minimized. To minimize $\hat{F}(\boldsymbol{x}) + H(\boldsymbol{x})$ in Equation (4), the corresponding update rule can be calculated by back-propagation and Equation (7), respectively. Therefore, we can combine them as the optimization process starting from a sequence of initial latent variables following pure Gaussian distribution, i.e., $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and proceed with the iterative optimization in a continuous latent space as:

$$\boldsymbol{x}_{t-1} = \boldsymbol{x}_t - \eta(\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) + \omega\nabla_{\boldsymbol{x}}\hat{F}(\boldsymbol{x}_t)) + \boldsymbol{\sigma}_t\boldsymbol{z}, \tag{11}$$

where the constant coefficients are combined into η and ω for brevity. This optimization process is also shown in Fig. 4. At each optimization step, we will subtract the predicted noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ computed by the diffusion model and the gradient w.r.t predicted QoR $\nabla_{\boldsymbol{x}}\hat{F}(\boldsymbol{x}_t)$.

Arguably, the surrogate model may not provide accurate prediction on a sequence consisting of *noisy* latent variables. Following [25], for each optimization step, we revert the *noisy* variables $\boldsymbol{x}_t$ to a noise-free version $\hat{\boldsymbol{x}}_t$ to obtain a more accurate gradient to optimize the QoR. This can be done by the reparameterization trick used in [25], [26]:

$$\hat{\boldsymbol{x}}_t = \frac{\boldsymbol{x}_t - (\sqrt{1-\bar{\boldsymbol{\alpha}}_t})\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)}{\sqrt{\bar{\boldsymbol{\alpha}}_t}}. \tag{12}$$

Algorithm 2 Continuous Logic Optimization

Input: Trained surrogate model and diffusion model; target circuit.
Output: Optimized sequence for logic synthesis on the target circuit.

1: Randomly sample a sequence of latent variables from Gaussian noise: $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
2: **for** $t = T$ **to** 1 **do**
3: Run optimization step with Equation (13).
4: **end for**
5: Retrieve a sequence of transformations. ▷ Section III-D.

Combining Equation (12) and Equation (11), we can have the final update rule for solving the problem in Equation (4):

$$\boldsymbol{x}_{t-1} = \boldsymbol{x}_t - \eta(\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) + \omega\nabla_{\boldsymbol{x}}\hat{F}(\hat{\boldsymbol{x}}_t)) + \boldsymbol{\sigma}_t\boldsymbol{z}. \tag{13}$$

This equation illustrates our idea of optimizing sequences in continuous space with the optimized variables in the distribution of valid transformation embeddings. It generates latent variables in a sequence to minimize QoR and prevent them from having large discrepancies from the feasible transformation embeddings. The optimization process is illustrated in Algorithm 2.

D. Retrieval of Optimized Transformations

Upon convergence of the above optimization, we obtain the optimized $\boldsymbol{x}^*$ in latent space. Owing to the denoising process involved, the separate embeddings in $\boldsymbol{x}^*$ will be almost aligned with the embeddings of feasible transformation from $\mathbb{S}$ with minimal discrepancy. Therefore, the final optimized transformation sequence can be retrieved instantly. Our framework can be concluded in Algorithm 1. This optimized sequence will be sent to ABC [1] for validating the actual QoR.

IV. Experimental Results

In this section, we demonstrate the efficiency of our continuous optimization method with several baselines in terms of QoR and runtime. We also ablate the effectiveness of our continuous optimization with different choices of surrogate models and the necessity of the diffusion model.

A. Experiment Setup

Target circuits are taken from EPFL15 and ISCAS89 benchmarks [27], [28]. The PDK for technology mapping is the open-source ASAP7 PDK [29]. Our surrogate model is an MTL-based from [22], which consists of a 2-layer GNN, an LSTM, and two 2-layer attention modules. It is trained by 20000 randomly generated sequences synthesized by ABC [1]. Our diffusion model has a 1-D U-Net structure [18], [30]. It has 500 diffusion/denoising steps. The experiments are conducted on a machine with two AMD EPYC 7543 32-core CPUs and one Nvidia RTX 3090 GPU. We target optimizing synthesis sequences with length $L = 20$. For each circuit, we train a corresponding surrogate model and a diffusion model, which takes about 11 minutes and is a one-time effort.

We choose several state-of-the-art methods of logic optimization as baselines, including DRiLLS [5], abcRL [6],

TABLE II Delay and Area Comparison. Unit of area: μm^2. Unit of delay: ps.

Circuit	Original		DRiLLS [5]		abcRL [6]		BOiLS [9]		FlowTune [8]		**Ours**	
	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay
adder	867.43	1006.42	807.56	1005.60	941.10	1004.28	945.74	1005.69	**797.05**	1004.28	**797.05**	**1003.60**
arbiter	**1299.64**	**347.26**	1301.25	**347.26**	**1299.64**	**347.26**	**1299.64**	**347.26**	**1299.64**	**347.26**	**1299.64**	**347.26**
bar	2077.62	**75.37**	1680.94	76.39	**1669.79**	**75.37**	1680.94	76.39	1680.94	**75.37**	**1669.79**	76.39
cavlc	213.22	76.18	235.02	74.17	206.06	73.36	229.57	76.18	208.65	**72.97**	**198.77**	73.77
ctrl	66.29	40.44	54.33	35.34	53.86	32.94	56.68	40.49	48.03	**31.09**	**41.78**	**31.09**
dec	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**
div	12888.45	17251.69	12531.56	16871.04	12130.13	16766.32	13488.61	16898.22	13768.78	**16756.87**	**11900.00**	16765.37
hyp	176053.57	89237.98	168154.61	89319.01	**167755.80**	89207.74	181291.68	89269.65	176048.90	89237.98	168070.20	**89168.95**
i2c	556.44	73.08	491.11	61.86	491.81	**60.78**	549.74	**60.78**	469.11	61.11	**464.22**	**60.78**
int2float	90.45	71.06	91.16	70.00	75.98	71.06	85.82	71.06	**74.37**	**68.04**	78.10	71.06
log2	**20590.64**	1614.88	22084.77	1613.17	23092.16	1571.92	24947.17	1571.80	22412.07	**1570.75**	20997.60	1606.55
max	1968.73	896.81	1332.00	747.25	1369.65	681.15	1580.82	637.20	1340.26	657.73	**1317.22**	**612.07**
mem_ctrl	13377.19	439.94	11705.69	371.81	11138.74	360.04	10517.57	363.30	11016.25	360.52	**10289.00**	**353.37**
multiplier	**19738.85**	1100.52	21870.08	1098.25	21924.85	1049.30	23028.97	1093.97	22468.65	**1046.04**	19747.90	**1046.04**
priority	382.75	969.31	254.58	555.05	**208.43**	**331.00**	237.33	436.86	218.65	337.10	211.90	331.10
router	246.68	114.32	103.09	74.01	100.51	74.53	95.63	82.41	95.83	69.46	**82.35**	**67.57**
sin	4576.88	752.97	4478.85	726.67	4619.88	706.06	4896.39	714.22	4486.30	**702.59**	**3921.83**	704.74
sqrt	12109.51	21604.47	11683.06	21318.41	13991.26	21330.84	13685.81	21385.74	11736.63	**21316.52**	**11569.62**	21320.39
square	11284.68	984.70	13611.74	935.58	12658.78	934.25	13382.14	978.47	12637.56	977.54	**11086.65**	**929.45**
voter	17367.28	333.12	11398.64	294.34	11359.52	305.71	12254.71	290.33	**10986.33**	285.26	11150.58	**283.87**
c17	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**
c432	165.77	128.00	137.13	119.37	**88.42**	118.82	135.29	124.42	112.19	119.55	88.67	**108.55**
c499	487.36	105.30	524.26	98.59	514.01	98.59	487.36	105.30	487.65	97.51	**480.45**	**97.25**
c880	207.21	92.22	230.74	91.97	221.61	91.09	226.78	94.59	**197.17**	90.54	206.74	**89.90**
c1355	494.16	107.81	524.88	100.12	557.81	97.51	507.74	114.78	502.96	**97.25**	**483.95**	99.89
c1908	465.92	142.06	452.96	130.87	436.59	131.66	447.51	131.73	**410.87**	129.25	426.20	**127.96**
c2670	631.64	88.95	522.35	94.14	506.68	88.95	554.93	99.59	491.30	86.28	**463.09**	**83.37**
c3540	871.97	175.82	705.59	164.89	692.83	160.31	777.00	169.21	**627.92**	158.88	677.16	**157.01**
c5315	1069.64	175.32	822.22	128.71	824.82	121.18	938.61	131.33	758.96	120.25	**726.85**	**116.76**
c6288	2379.73	424.29	2577.50	417.55	2594.44	**415.73**	2298.16	424.29	2546.76	416.58	**2269.26**	424.39
c7552	1635.09	151.14	1221.87	118.99	1072.60	107.16	1252.75	146.18	1058.71	105.75	**1027.64**	**103.44**
Mean	9821.22	4471.81	9415.57	4422.75	9448.09	4401.93	10070.14	4418.91	9386.73	4400.48	**9097.98**	**4396.99**
Ratio	1.079	1.017	1.035	1.006	1.038	1.001	1.107	1.005	1.032	1.001	**1.000**	**1.000**
Geo. mean	1212.81	319.57	1088.54	292.09	1056.72	281.97	1117.81	296.23	1029.10	279.15	**983.51**	**276.69**
Ratio	1.233	1.155	1.107	1.056	1.074	1.019	1.137	1.071	1.046	1.009	**1.000**	**1.000**

BOiLS [9], and FlowTune [8], all of which have open-source implementations. We spare all 64 CPU cores when running them for each circuit and one GPU if needed. We modify them meticulously so that both baseline methods and ours use the same set of transformations $\mathbb{S}$ to form a sequence with a length of 20 for a fair comparison. The numbers of optimization iterations of baselines are properly set according to their original papers. We repeat the evaluation 30 times for all methods and select the best sequence for comparison. For DRiLLS [5] and BOiLS [9], which support multi-objective optimization (delay and area), we evaluate them in a multi-objective scenario. For abcRL [6] and FlowTune [8], designed for single-objective optimization, we select the best results for area and delay separately.

B. QoR Comparison

Here we compare the delay and area after logic synthesis between our method and baseline methods. The results are shown in TABLE II. Note that the circuit scale in these benchmarks may vary significantly, and thus computing the mean value over all results may undermine the significance of small circuits (e.g., `ctrl`) and be biased towards large circuits (e.g., `hyp`). To mitigate this bias, we also list the geometric mean results, which reduce the disproportionate impact of substantially large values on the overall relative comparison. It can be seen that the proposed method can always dominate all baselines. Particularly, we can achieve a 4% to 13% reduction in the area and a 1% to 7% reduction in delay. This result validates that our continuous optimization is effective and can find superior sequences compared with baseline methods.

C. Runtime Comparison

For a fair comparison, for DRiLLS [5], abcRL [6], and BOiLS [9], we subtract the runtime of ABC [1] interleaved in the optimization loop so that the comparison is solely on the optimization algorithm itself. For our method, despite that model training may take time in our approach, it is a one-time effort. Therefore, the actual sequence optimization process in our approach is highly efficient. The runtime of our method compared with previous methods is shown in Fig. 5. It can be seen that the proposed continuous optimization method can achieve a 5X to 130X speedup compared with previous methods. In contrast, baseline methods like RL and Bayesian optimization require the model to be updated during the optimization process, and MAB searches a discrete space without explicit gradient information. Thus the efficiency of the baseline methods is not so high. The abcRL [6] is much slower as it requires building graphs from AIG being synthesized every after a transformation is applied. It is clearly shown that our continuous space optimization method is much more efficient than previous discrete optimization methods as it directly optimizes the sequence in a continuous space that avoids extensive discrete search.

D. Effectiveness of the Diffusion Model

We devise an ablation study to further validate the effectiveness of the diffusion model. To this end, we perform extensive

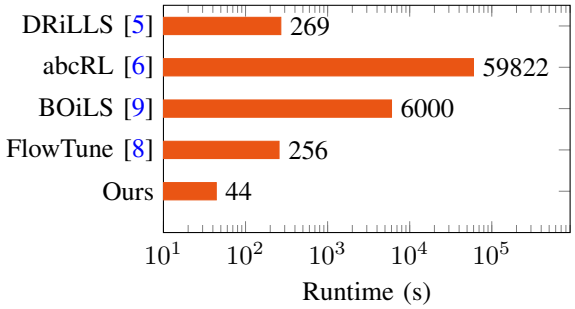


Fig. 5 Comparison of runtime.

validation not only on the MTL-based model [22], but also hybrid graph-based model (LOSTIN) [21] and convolutional neural network-based model (CNN) [4]. Regarding the counterpart without diffusion model, we perform the optimization using only the gradient from the surrogate model, i.e.,

$$\boldsymbol{x}_{t-1} = \boldsymbol{x}_t - \omega \nabla_{\boldsymbol{x}} \hat{F}(\boldsymbol{x}_t). \tag{14}$$

Fig. 6 presents the QoR results of continuous optimizations with various surrogate models, with and without the incorporation of the diffusion model. It can be seen that no matter which surrogate model we choose, incorporating the diffusion model can always result in a substantial performance improvement, which validates the general applicability of the proposed framework. Moreover, it is noted that solely utilizing a surrogate model to perform continuous optimization can hardly exceed the performance of baseline methods, e.g., FlowTune [8]. When the diffusion model is integrated, all the results can surpass the baselines. Please note that our continuous optimization method equipped with different surrogate models has approximately the same runtime (∼40 s) and is still much faster than baselines.

Next, we demonstrate a visualization of the optimized embeddings in latent space with t-SNE [23] in Fig. 7. With the diffusion model, the optimized variables are aligned with the embeddings of feasible transformations, allowing them to be mapped back to transformations instantly. The corresponding sequence in Fig. 7 is [`rw; rw; rsz; rfz; rw; rf; rf; rfz; b; rs; rw; rsz; rwz; rw; rw; rsz; rsz; rsz; rw; rsz`], which yields an area of 11900.00 μm^2. In contrast, without the diffusion model, optimized variables diverge from embeddings of feasible transformations. Mapping these latent variables back to the nearest feasible transformations with large discrepancies gives a sequence as [`rfz; rfz; rwz; rs; rs; rsz; rs; rs; rs; rs; rs; rwz; rs; rwz; rs; rs; rwz; rw; rwz; rwz`] and leads to an area of 22654.15 μm^2, which is 1.9 times larger than the area with a diffusion model.

V. Conclusion

In this paper, we propose to efficiently optimize logic synthesis sequences in a continuous space. The gradient-based optimization in continuous space alleviates the heavy runtime burden of extensive discrete search of transformations. We additionally incorporate a diffusion model to keep latent variables in feasible embedding distributions, allowing instant retrieval of feasible transformations without degrading QoR. Our experimental results show that our work is not only significantly faster than previous methods but also reaches better QoR.

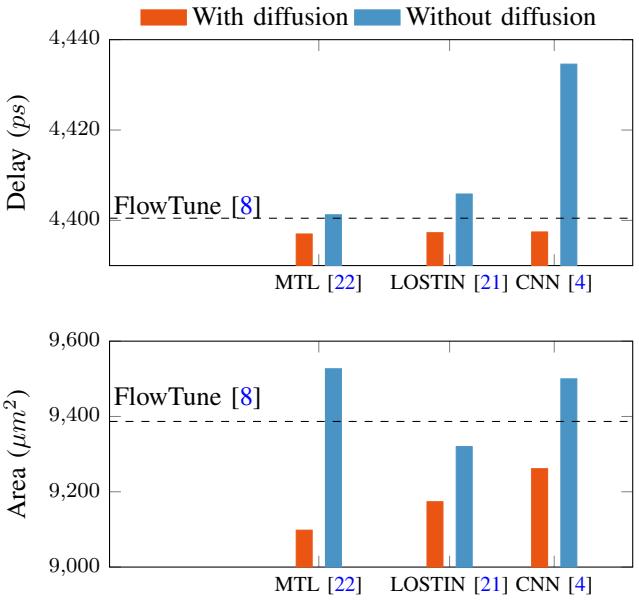


Fig. 6 Area and delay with and without diffusion model for various surrogate models.

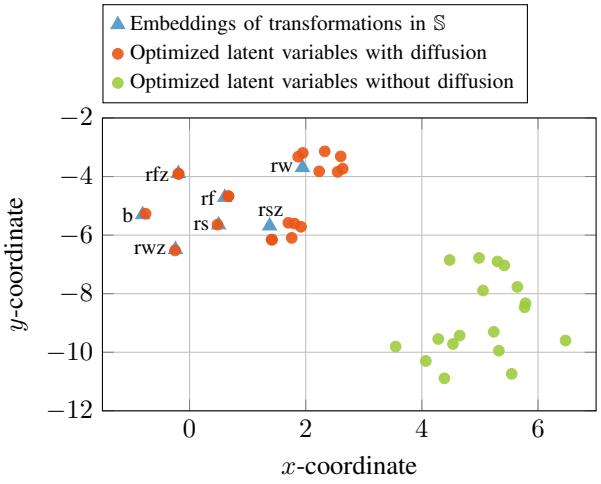


Fig. 7 The t-SNE projection of transformation embeddings from `div` design. The optimized latent variables without diffusion model deviate from feasible transformation embeddings.

Acknowledgment

This work is supported in part by the Department of Education of Guangdong Province (No. 2024KTSCX037) and the Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project, Grant No.2023A03J0013).

REFERENCES

[1] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Proc. CAV*. Springer, 2010, pp. 24–40.

[2] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE TCAD*, vol. 35, no. 5, pp. 806–819, 2015.

[3] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "Lsoracle: a logic synthesis framework driven by artificial intelligence: Invited paper," in *Proc. ICCAD*, 2019, pp. 1–6.

[4] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *Proc. DAC*, 2018, pp. 1–6.

[5] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *Proc. ASPDAC*, 2020, pp. 581–586.

[6] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *Proc. MLCAD*, 2020, p. 145–150.

[7] C. Lv, Z. Wei, W. Qian, J. Ye, C. Feng, and Z. He, "Gpt-ls: Generative pre-trained transformer with offline reinforcement learning for logic synthesis," in *Proc. ICCD*, 2023, pp. 320–326.

[8] W. L. Neto, Y. Li, P.-E. Gaillardon, and C. Yu, "Flowtune: End-to-end automatic logic optimization exploration via domain-specific multiarmed bandit," *IEEE TCAD*, vol. 42, no. 6, pp. 1912–1925, 2023.

[9] A. Grosnit, C. Malherbe, R. Tutunov, X. Wan, J. Wang, and H. B. Ammar, "Boils: Bayesian optimisation for logic synthesis," in *Proc. DATE*, 2022, pp. 1193–1196.

[10] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[11] T. Wu, T. Maruyama, L. Wei, T. Zhang, Y. Du, G. Iaccarino, and J. Leskovec, "Compositional generative inverse design," *arXiv preprint*, 2024.

[12] Q. Zhao, D. B. Lindell, and G. Wetzstein, "Learning to solve pde-constrained inverse problems with graph networks," *arXiv preprint*, 2022.

[13] J. Jiang and J. A. Fan, "Global optimization of dielectric metasurfaces using a physics-driven neural network," *Nano letters*, vol. 19, no. 8, pp. 5366–5372, 2019.

[14] J. Lu, L. Lei, F. Yang, L. Shang, and X. Zeng, "Topology optimization of operational amplifier in continuous space via graph embedding," in *Proc. DATE*, 2022, pp. 142–147.

[15] T. White, "Sampling generative networks," *arXiv preprint*, 2016.

[16] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "Cktgnn: Circuit graph neural network for electronic design automation," *arXiv preprint*, 2023.

[17] J. Chu, J. Park, S. Lee, and H. J. Kim, "Inversion-based latent bayesian optimization," in *Proc. NIPS*, 2024.

[18] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Proc. NIPS*, vol. 33, pp. 6840–6851, 2020.

[19] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2021.

[20] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8633–8646, 2022.

[21] N. Wu, J. Lee, Y. Xie, and C. Hao, "Lostin: Logic optimization via spatio-temporal information with hybrid graph models," in *Proc. ASAP*, 2022, pp. 11–18.

[22] Y. Ouyang, S. Li, D. Zuo, H. Fan, and Y. Ma, "Asap: Accurate synthesis analysis and prediction with multi-task learning," in *Proc. MLCAD*, 2023, pp. 1–6.

[23] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[24] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint*, 2013.

[25] A. Bansal, H.-M. Chu, A. Schwarzschild, S. Sengupta, M. Goldblum, J. Geiping, and T. Goldstein, "Universal guidance for diffusion models," in *Proc. CVPR*, 2023, pp. 843–852.

[26] L. Wei, P. Hu, R. Feng, H. Feng, Y. Du, T. Zhang, R. Wang, Y. Wang, Z.-M. Ma, and T. Wu, "A generative approach to control complex physical systems," *arXiv preprint*, 2024.

[27] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proc. IWLS*, 2015.

[28] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[29] X. Xu, N. Shah, A. Evans, S. Sinha, B. Cline, and G. Yeric, "Standard cell library design and optimization methodology for asap7 pdk," in *Proc. ICCAD*, 2017, pp. 999–1004.

[30] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.

MetaDSE: A Few-shot Meta-learning Framework for Cross-workload CPU Design Space Exploration

Runzhen Xue[1,2,*], Hao Wu[3,*], Mingyu Yan[1,2,†], Ziheng Xiao[1], Xiaochun Ye[1,2], Dongrui Fan[1,2]
[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China;
[2]University of Chinese Academy of Sciences, Beijing, China;
[3]University of Electronic Science and Technology of China, Chengdu, China;
{xuerunzhen21s, yanmingyu, xiaoziheng, yexiaochun, fandr}@ict.ac.cn, wh.pyjnqd@gmail.com

***Abstract*—Cross-workload design space exploration (DSE) is crucial in CPU architecture design. Existing DSE methods typically employ the transfer learning technique to leverage knowledge from source workloads, aiming to minimize the requirement of target workload simulation. However, these methods struggle with overfitting, data ambiguity, and workload dissimilarity.**

To address these challenges, we reframe the cross-workload CPU DSE task as a few-shot meta-learning problem and further introduce MetaDSE. By leveraging model agnostic meta-learning, MetaDSE swiftly adapts to new target workloads, greatly enhancing the efficiency of cross-workload CPU DSE. Additionally, MetaDSE introduces a novel knowledge transfer method called the workload-adaptive architectural mask algorithm, which uncovers the inherent properties of the architecture. Experiments on SPEC CPU 2017 demonstrate that MetaDSE significantly reduces prediction error by 44.3% compared to the state-of-the-art. MetaDSE is open-sourced and available at this anonymous GitHub.

***Index Terms*—Design Space Exploration, Cross-workload, Prediction Model, Meta-learning**

I. Introduction

Design Space Exploration (DSE) is pivotal in modern CPU architecture design, enabling the optimization of configurations to balance multiple objectives like performance, power, and area (PPA) [1], [2]. The increasing complexity of CPU architectures has led to an exponential expansion of the design space, presenting significant challenges in efficiently identifying optimal configurations.

Prior DSE frameworks [3]–[11] have utilized learning-based surrogate models to expedite the DSE process, achieving substantial time savings compared to exhaustive simulations. However, these workload-specific DSE frameworks rely heavily on extensive simulations to generate sufficient data for training machine learning models, necessitating the evaluation of numerous design points. This approach can be exceedingly time-intensive, especially for complex architectural spaces.

To mitigate this challenge, various studies [12]–[18] have adopted the transfer learning technique to reduce simulation overhead. By leveraging previously collected data from source workloads, transfer learning enables the model to adapt to unseen target workloads, a process known as cross-workload DSE. Cross-workload DSE generally follows a two-stage training process: first, pre-training on data from source workloads, and then fine-tuning on limited data from the target workload. However, current cross-workload DSE frameworks have notable limitations in both upstream pre-training stage and downstream adaptation stage:

- **Upstream Pre-training Issues:** Existing methods typically rely on machine learning based surrogate models [12]–[14], [17], [18] combined with standard supervised learning, which are prone to overfitting the pre-training data. Additionally, the inherent data ambiguity across multiple workloads introduces optimization challenges for transfer learning, often hindering effective convergence.
- **Downstream Adaptation Issues:** Given the high cost of simulations, training samples for new workloads are often limited, leading most existing frameworks to rely on knowledge transfer for adaptation. However, these approaches typically assume workload similarity as a prerequisite [12], [15], [16], which is not always the case. When this assumption fails, the adaptation performance degrades significantly, limiting the effectiveness of traditional transfer learning methods in cross-workload CPU DSE tasks.

To tackle these issues, we rethink cross-workload CPU DSE as a few-shot problem and propose MetaDSE, an open-source framework featuring two key stages: upstream pre-training and downstream adaptation. **For the upstream pre-training stage,** we adopt a meta-learning approach [19]–[21], well-suited for addressing few-shot problems. Unlike normal supervised learning, meta-learning emphasizes "learning how to learn" across tasks, making it ideal for tackling data ambiguity in DSE. By treating each workload as a distinct task during meta-training, the model avoids overfitting to specific instances and instead captures shared patterns and relationships across tasks. **For the downstream adaptation stage,** we design an adaptation algorithm called the Workload-adaptive Architectural Mask (WAM). WAM provides a novel approach to knowledge transfer, emphasizing the inherent properties of the architecture rather than relying on workload similarity.

To summarize, we list our contributions as follows:

- We conduct a comprehensive analysis of existing CPU cross-workload DSE frameworks, identifying their limi-

*Runzhen Xue and Hao Wu contributed equally to this work.
†The corresponding author is Mungyu Yan (*yanmingyu@ict.ac.cn*).

tations in handling data ambiguity and enabling efficient knowledge transfer.
- We reframe the CPU cross-workload DSE task as a few-shot meta-learning problem, using the Model-Agnostic Meta-Learning (MAML) algorithm to enhance model adaptability with limited samples from new workloads.
- We introduce a novel adaptation algorithm named WAM, which captures inherent properties of the architecture, enabling efficient and generalized DSE framework across diverse workloads.
- Extensive experiments on the SPEC CPU 2017 benchmark suite show that our proposed MetaDSE framework significantly outperforms state-of-the-art methods, achieving 44.3% reductions in prediction errors across various workloads.

II. Background & Related Work

A. Cross-workload DSE framework

1) Taxonomy: Most cross-workload CPU DSE methods [12]–[18] use transfer learning [22], [23] to reduce sample requirements for new workloads by leveraging source knowledge, typically following three strategies. **Linear Fitting.** Work [18] uses linear regression to combine predictions from fixed source models, mapping them to the target workload label space using a few labeled samples. It merely builds a linear mapping between the source and target label spaces, relying heavily on the unrealistic assumption of identical distributions and linearity. **Data Augmentation.** Work [17] employs the Gaussian mixture model [24] to model the original data and perform augmentation. By swapping the coefficients of high-probability and low-probability Gaussian components, the method enhances rare data distributions, aiming for more balanced data augmentation while preserving the overall distribution. To some extent, this work stems from the premise of similarity in the distribution of workloads. **Similarity Analysis.** Studies [12]–[16] explore workload similarity for model adaptation using distinct strategies. [15], [16] leverage workload signatures, representing workloads during pretraining and identifying the most similar signature for downstream adaptation. This approach transfers knowledge through signature-based representation. In contrast, methods like TrEnDSE [12], TrDSE [13], and TrEE [14] adopt a data-driven approach. TrEnDSE uses Wasserstein distance [25] to measure workload similarity, while TrDSE clusters workloads using orthogonal array (OA) sampling and distributional features. TrEE further refines this with an OA-based foldover sampling strategy. These methods select similar samples from the pre-trained dataset to enhance downstream adaptation, directly integrating relevant upstream data without relying on explicit signature representation.

2) Re-evaluating Similarity-based Approaches: In this section, we examine the frameworks of incorporating data from source workload into downstream adaptation. One of the earliest works to apply this concept is TrDSE [13], which influenced subsequent frameworks such as TrEE [14] and TrEnDSE [12]. However, these methods always include a one-time training process using data from the target workload. Actually, concerns over data leakage make it uncommon and impractical to access pre-trained data in many real-world scenarios. These methods can be better described as data augmentation rather than transfer learning [23].

B. Few-shot Problem & Meta-learning

The few-shot learning problem involves training models to generalize effectively using only a small number of labeled samples [26], [27], typically just a few dozen. This scenario is common in situations where labeling data is costly. To address this challenge, meta-learning has emerged as a powerful approach. In meta-learning [20], [20], a model is trained across a diverse set of tasks and is optimized to generalize well over a distribution of tasks. Here, each task is associated with a dataset $\mathcal{D}$ containing features and labels, where the entire dataset acts as a single data sample. The goal of meta-learning can be formalized similarly to supervised learning, with the optimal model represented as $\theta^* = \arg\min_\theta \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D})}[\mathcal{L}_\theta(\mathcal{D})]$.

As depicted in Fig. 1, the meta-learning model is trained on a variety of curated tasks from different datasets, where the supervised signal from these tasks guides the model to perform well across a wide range of unseen tasks. One common meta-learning technique is MAML [19], which aims to learn an optimal initialization of model parameters. This initialization enables the model to rapidly adapt to any new task with just a few gradient updates, providing efficient solutions for downstream few-shot problems.

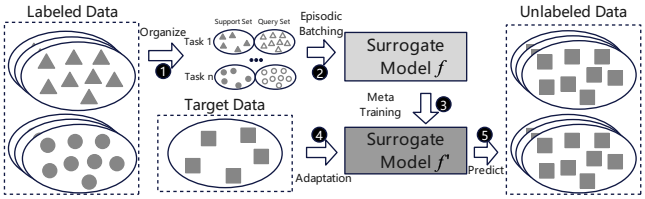


Fig. 1. Typical workflow of Model-Agnostic Meta-Learning.

III. Motivation

A. Limitations in Prior Arts

Current cross-workload DSE frameworks have notable limitations in both the pre-training stage and the adaptation stage.

1) Upstream Pre-training Issues: In the pre-training stage, the surrogate model is prone to overfitting the pre-training data and faces data ambiguity. For instance, previous methods [12]–[14] typically utilize machine learning based surrogate models, such as tree-based ones. These models tend to capture patterns specific to the source workloads and lead to overfitting with standard supervised training. However, due to the variability and ambiguity inherent in cross-workload CPU DSE scenarios, similar architectural configurations may yield vastly different performance metrics across workloads. This discrepancy leads the model to learn misleading or unreliable features during pre-training. Consequently, these features may not generalize well when the model is fine-tuned on new target workloads, resulting in poor predictive accuracy and limited adaptability.

2) Downstream Adaptation Issues: For the downstream adaptation stage, the primary challenge lies in the method of knowledge transfer. Many prior works assume that workloads are similar [12], [15], [16], employing methods like Wasserstein distance [25] to compare metric space distributions between source and target workloads, but this assumption does not always hold. In real-world scenarios, workloads exhibit significant variability in computational behaviors, data access patterns, and performance requirements, making this assumption unreliable.

For instance, our experiments on the SPEC CPU 2017 benchmark suite (as shown in Fig. 2) demonstrate that the similarities across workloads are inconsistent, with many showing substantial differences. This variability undermines the effectiveness of transfer learning methods like TrEnDSE when faced with diverse and complex workload characteristics. As a result, these methods may struggle to deliver accurate performance predictions for real-world DSE tasks, underscoring the need for a more adaptive, workload-independent approach to tackle the challenges of cross-workload CPU DSE.

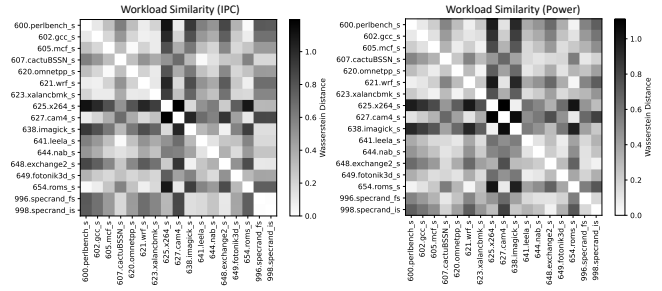


Fig. 2. Wasserstein distances among SPEC CPU 2017 workloads. The darker the color, the less similar the two workloads.

B. Rethinking Cross-workload CPU DSE

To overcome the limitations of prior arts, we introduce two innovative solutions tailored to the distinct stages.

For the pre-training stage, we address the challenges of overfitting and data ambiguity by reframing the task as a few-shot meta-learning problem. This perspective shift allows the model to be trained on a variety of tasks using only a small number of labeled samples, thereby reducing the risk of overfitting and improving its ability to generalize across different workloads. By leveraging the meta-learning technique, the surrogate model can capture shared patterns across workloads, eliminating the impact of data ambiguity.

For the adaptation stage, we recognize the limitations of conventional methods that rely on workload similarity for knowledge transfer. Instead, we propose shifting the focus from workload similarity to the intrinsic properties of the architectural design. To harness these inherent characteristics, we introduce the WAM algorithm. WAM uncovers key relations between architectural parameters and leverages them during adaptation. This novel approach enhances the model's robustness and adaptability, improving the overall effectiveness of cross-workload CPU DSE.

IV. Design of MetaDSE

A. Overview of MetaDSE

In this section, we introduce MetaDSE, a framework designed to enhance the efficiency of cross-workload CPU DSE. The workflow of MetaDSE, illustrated in Fig. 3, is divided into two main stages: pre-training (❶–❾) and adaptation (①–③). For the surrogate model f, we employ a transformer-based predictor, inspired by the strong performance of AttentionDSE [7] in CPU DSE tasks.

In the pre-training stage, the meta-training process begins by creating a copy of the surrogate model (parameterized by $\hat{\theta}$). This model is first fine-tuned on the support set of each task through a few gradient descents (❶, ❷). Subsequently, the fine-tuned model evaluates the loss on the corresponding query set (❸), and this computed loss contributes to the overall meta-loss (❹). The meta-loss is then used to update the parameters of the original surrogate model (parameterized by θ). After each training epoch, a meta-validation step is conducted to evaluate the model's performance and to identify the optimal parameters for downstream tasks. In the adaptation stage, the pre-trained surrogate model (parameterized by $\hat{\theta^*}$) undergoes fine-tuning with the WAM algorithm (②). WAM focuses on the inherent properties of the architecture, adjusting the model to emphasize relevant interactions between architectural parameters. Further technical details on the WAM adaptation process are elaborated in the subsequent sections.

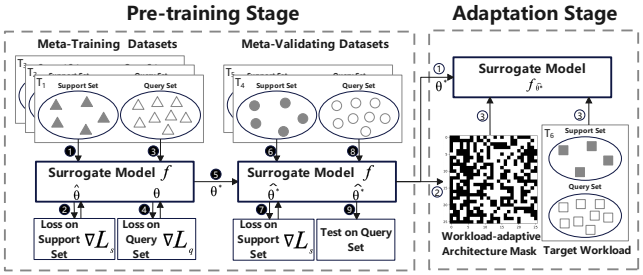


Fig. 3. The overview workflow of MetaDSE.

B. MAML-based Pre-training Stage

The pre-training stage in MetaDSE typically aims to build a model that performs well across a wide range of tasks. Instead, our MAML-based approach focuses on finding an optimal model initialization, θ, which can rapidly adapt to new downstream tasks using few data. This strategy allows the predictor to generalize effectively to diverse workloads without extensive retraining, tackling the core challenge of the few-shot problem in cross-workload CPU DSE.

Algorithm 1 describes the MAML-based pre-training process for the surrogate model. Before the training stage, each iteration begins by sampling tasks from the task distribution $P(T)$, representing different workloads. For each task, the model is trained using a support set S and evaluated on a query set Q. The training in MAML can be divided into two phases: the inner loop and the outer loop. The inner loop (from line 4 to line 12) focuses on task-specific adaptation, simulating the fine-tuning process during model deployment. Here, the model updates its parameters using the support set from a

Algorithm 1: The Pre-Training Procedure of Predictors

Input: f_θ: the predictor with random parameters; $P(T)$: task distribution of one workload; s, q: sample amount of support set and query set in one task, correspondingly; α, β: learning rate in inner and outer loop;
Output: f_{θ^*}: the meta-trained predictor

1 randomly initialize θ
2 **while** *not done* **do**
3 Sample tasks from some workloads' distribution $T_i \sim P(T)$
4 **for** *each task* t *in* T_i **do**
5 $\hat{\theta} = \theta$
6 Split out support set $S = (x_j, y_j)$ and query set $Q = (x_k, y_k)$ from t, $S, Q \leftarrow Spilt(t, s, q)$
7 **for** *each inner step* **do**
8 Evaluate $\nabla_{\hat{\theta}} \mathcal{L}(f_{\hat{\theta}})$ with respect to S
9 Compute adapted parameters with gradient: $\hat{\theta} = \hat{\theta} - \alpha \nabla_{\hat{\theta}} \mathcal{L}(f_{\hat{\theta}})$
10 **end**
11 Save Q and $f_{\hat{\theta}}$ temporarily
12 **end**
13 Evaluate and average meta gradients of every task from T_i, $\nabla_\theta \mathcal{L}(f_{\hat{\theta}})$ with respect to each Q in every t
14 Compute adapted meta parameters with meta gradient: $\theta = \theta - \beta \nabla_\theta \mathcal{L}_{T_i}(f_{\hat{\theta}})$
15 **end**
16 $\theta^* = \theta$
17 return final predictor f_{θ^*}

Algorithm 2: The Adaptation Procedure of Predictors

Input: f_{θ^*}: the meta-trained predictor; $P(T)$: task distribution of test workload; s, q: sample amount of support set and query set in one task, correspondingly; γ: learning rate during adaptation; M: workload-adaptive architectural mask.
Output: $f_{\hat{\theta^*}}$: the adapted predictor

1 Equip f_{θ^*} with M in its self-attention operator
2 Set M to learnable state: $M.required_grad = True$
3 Sample tasks from test workload's distribution $T_i \sim P(T)$
4 **for** *each task* t *in* T_i **do**
5 $\hat{\theta^*} = \theta^*$
6 Split out support set $S = (x_j, y_j)$ and query set $Q = (x_k, y_k)$ from t, $S, Q \leftarrow Spilt(t, s, q)$
7 **for** *each step* **do**
8 Evaluate $\nabla_{\hat{\theta^*}} \mathcal{L}(f_{\hat{\theta^*}}, M)$ with respect to S
9 Compute adapted parameters with gradient: $\hat{\theta^*} = \hat{\theta^*} - \gamma \nabla_{\hat{\theta^*}} \mathcal{L}(f_{\hat{\theta^*}})$
10 **end**
11 return adapted predictor $f_{\hat{\theta^*}}$
12 **end**

specific task. This phase aims to find temporary parameters $\hat{\theta}$ that adapt well to the current task. The update rule in this phase is: $\hat{\theta} \leftarrow \hat{\theta} - \alpha \nabla_{\hat{\theta}} \mathcal{L}(\hat{\theta})$. where α is the inner loop learning rate. This gradient descent adjusts the parameters to minimize the task-specific loss $\mathcal{L}$, effectively mimicking how the model would adapt when exposed to new data. The outer loop (from line 2 to line 15) aims to optimize the initial parameters θ by aggregating the feedback from multiple tasks. After the inner loop updates for each task, the model evaluates its performance on the query set to calculate the meta-loss. This meta-loss reflects how well the adapted model (with parameters $\hat{\theta}$) generalizes across different tasks. The meta-parameters θ are then updated based on the meta-gradient: $\theta = \theta - \beta \nabla_\theta \mathcal{L}_{T_i}(f_{\hat{\theta}})$ where β is the meta-learning rate. This outer-loop update ensures that the initial parameters θ are optimized to quickly adapt to new tasks with minimal data.

This two-level optimization process enables MetaDSE to efficiently adapt to new workloads. It ensures the pre-trained model generalizes well to diverse, unseen tasks, minimizing the need for extensive fine-tuning on large datasets.

C. Workload-adaptive Architectural Mask Adaptation Stage

In cross-workload CPU DSE tasks, workload dissimilarity poses a significant challenge for knowledge transfer. To tackle this, we propose WAM adaptation, which leverages inherent architectural properties to enhance the pre-trained model's generalization to unseen workloads.

A critical issue is that not all architectural parameters are fully correlated. Previous efforts [12]–[15], [18] have largely overlooked this, leading to noise from irrelevant parameter interactions and reduced prediction accuracy. To address this, we introduce an algorithm that generates a mask for the surrogate model, utilizing intermediate attention weights from the transformer-based predictor during pre-training. This mask filters out irrelevant parameter interactions, enabling the model to focus on key factors and improving prediction accuracy.

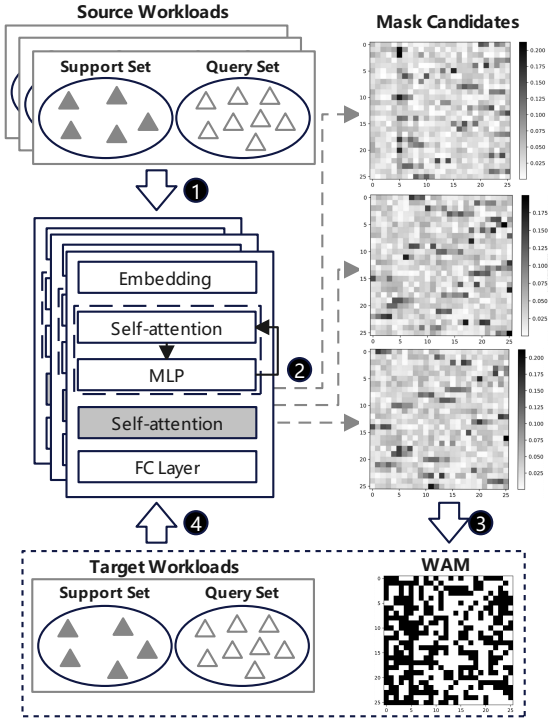


Fig. 4. The example of workload-adaptive architectural mask generation.

As Fig. 4 shows, we utilize attention weights obtained during the pre-training process to analyze the relationships between architectural parameters (❶, ❷). We extract all the attention weights from the last self-attention layer, referring to them as mask candidates. By identifying parameter interactions that frequently occur across diverse workloads, we construct a WAM (❸) that highlights high-frequency correlations, effectively filtering out less relevant interactions.

After the mask is generated, we leverage the mask and the target workloads to the adaptation stage (❹). The procedure is illustrated in Algorithm 2. The process also begins by sampling

tasks from the test workload's distribution $P(T)$. For each sampled task t, we initialize the adapted model parameters as $\hat{\theta}^* = \theta^*$ and split the task data into a support set S and a query set Q. During the adaptation process, we iteratively update the model parameters $\hat{\theta}^*$ using gradients computed based on the loss $\mathcal{L}$ evaluated on the support set S. The update rule $\hat{\theta}^* = \hat{\theta}^* - \gamma \nabla_{\hat{\theta}^*} \mathcal{L}(f_{\hat{\theta}^*})$ ensures that the model learns with mask M to minimize the error on the specific task at hand. Finally, the adapted predictor $f_{\hat{\theta}^*}$ is returned for each task, incorporating the updated parameters that better capture the features of target workloads.

WAM adaptation allows the surrogate model to focus on critical parameter interactions, reducing noise from irrelevant features and enhancing prediction accuracy. Moreover, it introduces a novel approach to knowledge transfer, prioritizing architectural properties over workload similarity.

V. Experimental Setup

Datasets Generation. We utilize GEM5 [28] as the timing-accurate simulator and McPAT [29] as the power modeling tool. We extend GEM5 using Python to generate different CPU cores with varying configurations. To bolster the credibility of our experiments, we employ SPEC CPU 2017 [30] as the evaluation benchmark. We utilize Simpoints [31] for each workload evaluation. Each workload is divided into at most 30 clusters, with each cluster containing ten million instructions.

For dataset splitting, we iteratively and randomly designated seven datasets for training, five for validation, and five for testing. This approach ensured balanced and robust training across various dataset combinations.

Baselines. We evaluate our proposed MetaDSE framework against several baselines from different perspectives. The first baseline is TrEnDSE [12], a state-of-the-art cross-workload CPU DSE framework. We then introduce TrEnDSE-Transformer, where the ensemble model in TrEnDSE is replaced with a Transformer-based predictor. Lastly, MetaDSE-w/o WAM is a variant of MetaDSE that excludes the WAM adaptation.

The Explored Design Space. The design space of the out-of-order processor is listed in Table I. The values in the third column are formatted as "start number: end number: stride". A two-level cache hierarchy with an 8192MB DRAM main memory configuration is employed.

Evaluation Metrics. The accuracy of the model is primarily measured using the Root Mean Squared Error (RMSE), which is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{1}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and N is the number of data points.

Additionally, other metrics such as Mean Absolute Percentage Error (MAPE) and Explained Variance (EV) are used for a more comprehensive evaluation:

TABLE I
Microarchitecture design space specification.

Parameters	Description	Candidate Value
Core Frequency	the frequency of CPU core in GHz	1/1.5/2/ 2.5/3
Pipeline Width	fetch/decode/rename/ dispatch/issue/writeback/ commit width	1:12:1
Fetch Buffer	fetch buffer size in bytes	16/32/64
Fetch Queue	fetch queue size in μ-ops	8:48:4
Branch Predictor	predictor type	BiModeBP/ TournamentBP
RAS Size	return address stack size	16:40:2
BTB Size	branch target buffer size	1024/2048/4096
ROB Size	reorder buffer entries	32:256:16
Int/Fp RF Number	number of physical integer/floating-point registers	64:256:8
Inst Queue	number of instruction queue entries	16:80:8
Load/Store Queue	number of load/store queue entries	20:48:4
IntALU	number of integer ALUs	3:8:1
IntMultDiv	number of integer multipliers and dividers	1:4:1
FpALU	number of floating-point ALUs	1:4:1
FpMultDiv	number of floating-point multipliers and dividers	1:4:1
Cacheline	cacheline size	32/64
L1 Cache Size	size of ICache in KB	16/32/64
L1 Cache Assoc.	associative sets of ICache	2/4
L2 Cache Size	size of L2 Cache in KB	128/256
L2 Cache Assoc.	associative sets of L2 Cache	2/4

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\left|y_i - y_i^{\text{real}}\right|}{y_i^{\text{real}}}\right) \times 100. \tag{2}$$

$$\text{EV} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{3}$$

where $\bar{y}$ is the mean of the actual values. Lower RMSE and MAPE values, and higher EV values, indicate better model performance.

VI. Experiment Results

A. Overall Analysis

The implementation of MetaDSE has two stages: upstream pre-training and downstream WAM adaptation. For pre-training, we run 15 epochs with 200 tasks per workload, each task having five support samples and forty-five query samples. Inner and outer loop learning rates are 1×10^{-5} and 1×10^{-4}, respectively. In each iteration, every model copy is optimized via five inner-loop steps on the support set using SGD, and the accumulated meta-loss is used to update the original model with the Adam optimizer. In the downstream stage, we fine-tune the pre-trained model using ten gradient steps with a

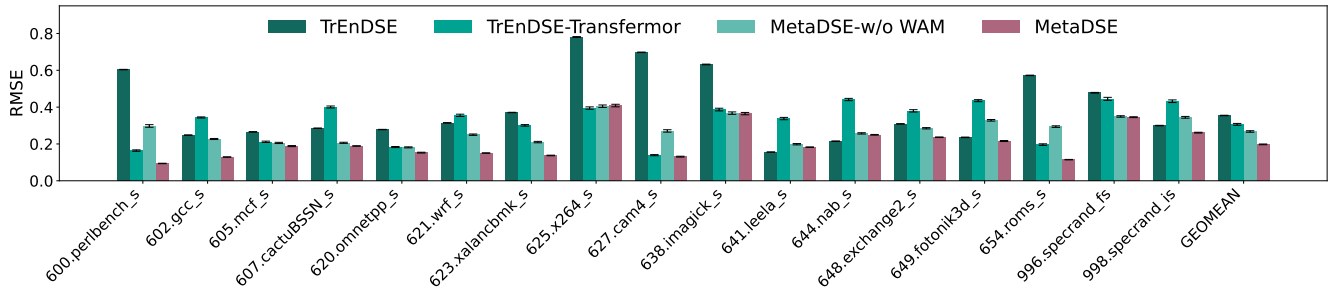


Fig. 5. Comparison of IPC RMSE per workload on SPEC CPU 2017 with the SOTA cross-workload CPU DSE framework (lower is better for RMSE).

TABLE II
THE OVERALL RESULTS, AVERAGED ACROSS FIVE DATASETS (600.PERLBENCH_S, 605.MCF_S, 620.OMNETPP_S, 623.XALANCBMK_S, AND 627.CAM4_S), ARE EVALUATED USING RMSE, MAPE, AND EXPLAINED VARIANCE.

Models / Metrics	RMSE ↓		MAPE ↓		EV ↑	
	IPC	Power	IPC	Power	IPC	Power
RF	0.4389±0.0150	0.5344±0.0198	1.1624±0.0827	0.3356±0.0178	-0.7997±0.1231	0.4470±0.0509
GBRT	0.3637±0.0125	0.4539±0.0171	0.9486±0.0678	0.2667±0.0144	-0.5152±0.1137	0.4634±0.0529
TrEnDSE	0.3270±0.0109	0.3990±0.0146	0.8386±0.0595	0.2348±0.0123	-0.5142±0.1079	**0.5711±0.0410**
MetaDSE	**0.2204±0.0072**	**0.3969±0.0095**	**0.5909±0.0460**	**0.2330±0.0045**	**-0.0471±0.0056**	0.3189±0.0065

learning rate of 1×10^{-5} and cosine annealing. We evaluate 1000 tasks per workload for mean and confidence intervals.

Fig. 5 illustrates the prediction accuracy comparison between MetaDSE and various baseline models on the SPEC CPU 2017 dataset. The last column presents the geometric mean (GEOMEAN) of prediction errors across all tasks. MetaDSE achieves a significant reduction in prediction error, lowering it by 44.3% compared to the state-of-the-art cross-workload CPU DSE framework, TrEnDSE. When comparing TrEnDSE with TrEnDSE-Transformer, we observe that simply replacing the ensemble learning model with a more powerful Transformer model provides some performance improvement. However, this improvement is relatively modest and inconsistent across different tasks. In contrast, within the meta-learning-based DSE framework, the inclusion of our proposed WAM adaptation technique leads to a substantial performance boost, reducing the average prediction error by 27%, and highlighting the effectiveness of adapting to workload-specific characteristics.

We further compare MetaDSE with several commonly used models in transfer learning for predicting IPC and power, including Random Forest (RF) and Gradient Boosting Regression Trees (GBRT), shown in Table II. Our results reveal that MetaDSE consistently outperforms TrEnDSE and other baselines across all evaluated workloads, demonstrating lower RMSE and MAPE values while achieving higher EV scores. These results further highlight the superiority of MetaDSE over traditional transfer learning approaches, emphasizing its enhanced ability to generalize and adapt effectively across different workloads in cross-workload CPU DSE tasks.

B. Exploring Data Impact

We further investigate the impact of pre-training and adaptation data on MetaDSE performance.

1) Sensitivity of Pre-training: For the upstream pre-training stage, we fix the downstream adaptation support set size to ten and change the support set size of the pre-training stage from five to forty. Fig. 6 illustrates that when the upstream and downstream data are aligned, the model achieves optimal transfer performance, with the highest EV and lowest RMSE. This improvement is attributed to the matched knowledge pattern and the higher learning efficiency when the data distribution is consistent across pre-training and adaptation phases, leading to better generalization and model adaptation.

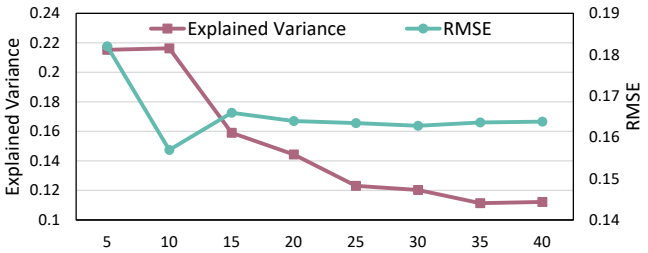


Fig. 6. The variation in explained variance and RMSE with respect to changes in the source sample size.

2) Sensitivity of Adaptation: We further investigate the impact of downstream adaptation data and compare MetaDSE to commonly used machine learning models, including RF, GBRT, and TrEnDSE. In this experiment, we fix the support set size of the upstream pre-training stage to ten while varying the adaptation support set size from five to forty. The results presented in Table III show that when the pretraining data size is fixed, increasing the adaptation data size results in modest performance improvements for MetaDSE. Notably, MetaDSE achieves high performance even with a smaller amount of adaptation data.

TABLE III
RMSE ON IPC PREDICTION AS THE DOWNSTREAM ADAPTATION SUPPORT SIZE (K) CHANGES.

Models / K	5	10	20	30	40
RF	0.4409	0.4397	0.4390	0.4386	0.4380
GBRT	0.2577	0.2390	0.2356	0.2321	0.2299
Baseline	0.2616	0.2397	0.2229	0.2147	0.2076
MetaDSE	0.1580	0.1562	0.1485	0.1471	0.1466

VII. CONCLUSION

In this work, we introduce meta-learning to cross-workload CPU DSE tasks and propose MetaDSE. MetaDSE utilizes the MAML and proposes the WAM algorithm to uncover the inherent architectural properties, offering a novel approach to prior knowledge transfer. Experiments on SPEC CPU 2017 benchmarks demonstrate that MetaDSE significantly reduces prediction errors compared to state-of-the-art methods, highlighting its effectiveness in improving model adaptability.

ACKNOWLEDGMENT

This work was supported by National Key Research and Development Program (Grant No. 2022YFB4501400), the National Natural Science Foundation of China (Grant No. 62202451 and No. 62302477), CAS Project for Young Scientists in Basic Research (Grant No. YSBR-029), and CAS Project for Youth Innovation Promotion Association.

REFERENCES

[1] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pp. 340–351, IEEE, 2007.

[2] A. Deshwal, S. Belakaria, G. Bhat, J. R. Doppa, and P. P. Pande, "Learning pareto-frontier resource management policies for heterogeneous socs: An information-theoretic approach," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 607–612, IEEE, 2021.

[3] D. Li, S. Yao, Y.-H. Liu, S. Wang, and X.-H. Sun, "Efficient design space exploration via statistical sampling and adaboost learning," in *Proceedings of the 53rd Annual Design Automation Conference*, pp. 1–6, 2016.

[4] D. Wang, M. Yan, Y. Teng, D. Han, X. Ye, and D. Fan, "A high-accurate multi-objective ensemble exploration framework for design space of cpu microarchitecture," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, GLSVLSI '23, (New York, NY, USA), p. 379–383, Association for Computing Machinery, 2023.

[5] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Proceedings of the 47th Design Automation Conference*, DAC '10, (New York, NY, USA), p. 120–125, Association for Computing Machinery, 2010.

[6] H. Wang, J. Shi, and Z. Zhu, "An expected hypervolume improvement algorithm for architectural exploration of embedded processors," in *Proceedings of the 53rd Annual Design Automation Conference*, pp. 1–6, 2016.

[7] R. Xue, H. Wu, M. Yan, Z. Xiao, X. Ye, and D. Fan, "Multi-objective optimization in cpu design space exploration: Attention is all you need," 2024.

[8] H. Fan, Y. Wang, S. Li, T. Liang, and W. Zhang, "Explainable fuzzy neural network with multi-fidelity reinforcement learning for micro-architecture design space exploration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

[9] D. Wang, M. Yan, X. Liu, M. Zou, T. Liu, W. Li, X. Ye, and D. Fan, "A high-accurate multi-objective exploration framework for design space of cpu," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023.

[10] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2021.

[11] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 1, pp. 1–23, 2023.

[12] D. Wang, M. Yan, Y. Teng, D. Han, H. Dang, X. Ye, and D. Fan, "A transfer learning framework for high-accurate cross-workload design space exploration of cpu," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023.

[13] D. Li, S. Wang, S. Yao, Y.-H. Liu, Y. Cheng, and X.-H. Sun, "Efficient design space exploration by knowledge transfer," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 1–10, 2016.

[14] D. Li, S. Yao, S. Wang, and Y. Wang, "Cross-program design space exploration by ensemble transfer learning," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 201–208, 2017.

[15] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra, "Using predictivemodeling for cross-program design space exploration in multicore systems," in *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*, pp. 327–338, 2007.

[16] Q. Guo, T. Chen, Y. Chen, L. Li, and W. Hu, "Microarchitectural design space exploration made fast," *Microprocessors and Microsystems*, vol. 37, no. 1, pp. 41–51, 2013.

[17] Y. Ding, N. Mishra, and H. Hoffmann, "Generative and multi-phase learning for computer systems optimization," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 39–52, 2019.

[18] C. Dubach, T. M. Jones, and M. F. O'Boyle, "An empirical architecture-centric approach to microarchitectural design space exploration," *IEEE Transactions on Computers*, vol. 60, no. 10, pp. 1445–1458, 2010.

[19] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.

[20] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-learning for semi-supervised few-shot classification," *arXiv preprint arXiv:1803.00676*, 2018.

[21] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 403–412, 2019.

[22] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI global, 2010.

[23] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[24] D. A. Reynolds *et al.*, "Gaussian mixture models.," *Encyclopedia of biometrics*, vol. 741, no. 659-663, 2009.

[25] S. Vallender, "Calculation of the wasserstein distance between probability distributions on the line," *Theory of Probability & Its Applications*, vol. 18, no. 4, pp. 784–786, 1974.

[26] X. Luo, H. Wu, J. Zhang, L. Gao, J. Xu, and J. Song, "A closer look at few-shot classification again," in *International Conference on Machine Learning*, pp. 23103–23123, PMLR, 2023.

[27] J. Yang, H. Wu, J. Zhang, L. Gao, and J. Song, "Effective and efficient few-shot fine-tuning for vision transformers," in *2024 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2024.

[28] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, p. 1–7, aug 2011.

[29] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*, pp. 469–480, 2009.

[30] Standard Performance Evaluation Corporation, "Spec cpu 2017." https://www.spec.org/cpu2017/, 2022.

[31] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program phase analysis," *Journal of Instruction Level Parallelism*, vol. 7, no. 4, pp. 1–28, 2005.

CHORD: Composable Hybrid Optical Reconfigurable Diffractive Framework For Optical Neural Network

Ziang Yin[1], Yu Yao[1], Jeff Zhang[1], Jiaqi Gu[1†]
[1]Arizona State University,
†*jiaqigu@asu.edu*

***Abstract*— Diffractive optical neural networks (DONNs), leveraging free-space light wave propagation for ultra-parallel, high-efficiency computing, have emerged as promising artificial intelligence (AI) accelerators. However, their inherent lack of reconfigurability due to fixed optical structures post-fabrication hinders practical deployment in the face of dynamic AI workloads and evolving applications. To overcome this challenge, we introduce, *for the first time*, a composable hybrid optical reconfigurable diffractive framework (CHORD), a physically composable architecture that unlocks a new degree of freedom and unprecedented versatility in DONNs. By leveraging full-system learnability, CHORD repurposes fixed fabricated optical hardware, achieving exponentially expanded functionality and superior task adaptability through the differentiable learning of system variables. Furthermore, CHORD adopts a hybrid optical/photonic design, combining the reconfigurability of integrated photonics with the ultra-parallelism of free-space diffractive systems. Extensive evaluations demonstrate that CHORD has digital-comparable accuracy on various task adaptations with 74× faster speed and 194× lower energy. Compared to prior DONNs, CHORD shows exponentially larger functional space with 5× faster training speed, paving the way for a new paradigm of versatile, composable, hybrid optical/photonic AI computing. Our code is open-sourced at link[1].**

I. Introduction

Emerging technologies promise efficiency and performance breakthroughs, and optical neural networks (ONNs) are at the forefront of this movement. They are poised to revolutionize next-generation AI computing by harnessing the inherent parallelism and speed of light. Diffractive ONNs (DONNs) [1], [2], [3], [4], [5] represent a compelling realization of optical AI hardware, employing free-space optical signal modulation, diffraction, and interference to achieve ultra-parallel intelligent information processing. Different from integrated photonic tensor cores (PTCs) [6], [7], [8], [9], [10], [11], [12] based on programmable photonic integrated circuits (PICs) designed for matrix-vector multiplication, free-space DONNs eliminate the dimension limitations of 2-D silicon chips and operate directly on light waves in 3-D free space, offering unprecedented parallelism through a global-view spatial linear operation.

However, DONNs face a significant challenge in their lack of reconfigurability after fabrication. The fixed manufactured phase masks limit their adaptability to new ML tasks, hindering their deployment in dynamic, real-world AI workloads [13], [14]. While research into reconfigurable phase masks based on tunable metasurfaces shows promise, these approaches remain in the immature stages. They struggle with limitations such as low endurance and reliability, which make them unsuitable for adoption in the near term.

To address these limitations, we shift our focus from device-level reconfigurability to system-level reconfigurability. Optical systems inherently feature adjustable parameters, such as wavelength, phase mask spacing, and element size, that significantly influence system behavior but are often manually determined based on experience in the early stage and fixed during the optimization process later. This raises a key question: **can these built-in hardware variables be automatically optimized for better expressivity?** However, for true task adaptation, adjusting these scalar variables alone lacks sufficient degrees of freedom. A fundamental system redesign is necessary to unlock the versatility needed for modern AI workloads. In this work, we address this challenge: **how to construct a versatile DONN system with exponentially many reconfigurable functionalities by re-purposing the same set of fabricated phase masks?** Furthermore, considering the strengths of both diffractive ONNs and integrated PTCs, we explore **how to hybridize free-space optics and integrated photonics organically** to leverage both the ultra-parallel processing capability of DONNs and the superior reconfigurability of integrated PTCs for joint NN acceleration.

[1]https://github.com/ScopeX-ASU/CHORD

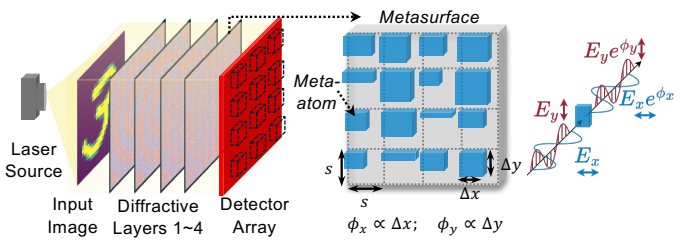


Fig. 1: Diffractive optical neural networks (DONN) based on metasurfaces with a pixel size of s. The rectangular meta-atom shape can be designed to realize a polarization-dependent phase shift for orthogonally polarized light.

To answer those questions, we investigate the intrinsic learnability of DONNs from the system level and propose a multi-dimensional reconfigurable hybrid DONN design paradigm, dubbed CHORD, with a physically composable architecture that can flexibly switch the orientation and placement order of phase masks for exponentially larger functional space. The main contributions are as follows,

- We present *the first* in-depth analysis of multi-dimensional learnability in DONNs, introducing a physically composable hybrid optical system CHORD for extreme multi-functionality.
- **Auto-Learned Multi-Dimensional System Variables**: we enable differentiable learning of DONNs across multiple dimensions (wavelength, spacing, orientation, permutation), exponentially enhancing expressivity and the functional space.
- **Hybrid Optical/Photonic System**: We explore the possibility of hybridizing free-space diffractive optics and reconfigurable integrated photonics for joint NN acceleration.
- **Superior Task Adaptability**: Extensive evaluations show that our system and differentiable learning algorithms enable digital-comparable task adaptation performance with 74× faster speed and 194× lower energy, outperforming prior DONNs with exponentially higher functional space and 5× higher training efficiency.

II. Background

A. Diffractive Optical Neural Networks (DONN)

As shown in Fig. 1, the main trainable parameters are weights on the phase mask, i.e., $\mathbf{\Phi} \in \mathbb{R}^{K\times K}$, where the phase shift of one unit cell is $\phi_{i,j} \in [0, 2\pi)$. When incident light X passes through the phase mask, it applies pixel-wise phase rotation, i.e., $e^{\Phi} \odot X$, where $\odot$ is element-

wise multiplication. Typically, to increase the number of trainable parameters, a DONN will have multiple cascaded phase masks, e.g., L layers [15]. Between two masks, the light will propagate in the free space with diffraction. This can be described using the Rayleigh-Sommerfeld diffraction, a dense linear transformation where each light on the output plane is the superposition of all light from the input plane with a distance-related intensity/phase modulation coefficient h. The transmission, including a phase mask and diffraction, is denoted as $X_{l+1} = \mathcal{H}(e^{\Phi_l} \odot X_l)$, formulated as [16]:

$$
\begin{aligned}
X_{l+1}(k_x^{l+1}, k_y^{l+1}) &= \sum_{k_x}^{K} \sum_{k_y}^{K} h_l(R, z, \lambda) \cdot e^{\phi_{k_x^l, k_y^l}} \cdot X_l(k_x^l, k_y^l), \\
h(R, z, \lambda) &= \frac{1}{2\pi} \frac{z}{R} \left(\frac{1}{R} - j\mathbf{k}\right) \frac{e^{j\mathbf{k}R}}{R}, \\
R &= \sqrt{((k_x^{l+1} - k_x^l) \cdot s)^2 + ((k_y^{l+1} - k_x^l) \cdot s)^2 + z^2}.
\end{aligned} \tag{1}
$$

In the diffraction coefficient h, $\mathbf{k} = \frac{2\pi}{\lambda}$ is the wave vector in free space, and λ is the wavelength, and z is the distance between two layers, $j = \sqrt{-1}$. R is the distance from the unit cell located at $(k_x{}^l, k_y{}^l)$ in the l-th layer to the unit cell located at $(k_x{}^{l+1}, k_y{}^{l+1})$ in the $(l+1)$-th layer. s is the pixel size of the phase mask unit cell.

As nanofabrication technology advances, a promising approach to implement DONN is using a sub-wavelength metasurface. Metasurfaces offer a compact solution, where each unit cell is called a meta-atom and consists of sub-wavelength nanostructures [2], [17], [18]. A typical meta-atom is a rectangular nano-pillar with precisely defined width, length, and thickness to achieve a specific phase shift, in Fig. 1. This structure is polarization-dependent and thus can induce independent phase changes in orthogonal light polarizations, e.g., ϕ_x for x-polarized and ϕ_y for y-polarized light. This is the foundation of our polarization-differential computing technique introduced later.

B. Integrated Photonic Tensor Cores

A different approach to implementing ONNs is through photonic tensor cores (PTCs). Unlike DONNs based on passive diffractive components, PTCs integrate photonic devices to actively modulate light to achieve high-speed matrix-vector multiplication (MVM) [6], [19], [20], [21], [11], [9], [10], [22]. DONNs are more natural for ultra-parallel low-energy spatial processing, while PTCs are more suitable in generic reconfigurable MVMs. This indicates complementary capabilities between the two hardware platforms for joint computing.

III. Related Works

Previous works have proposed solutions to build multi-functional DONNs. For instance, prior methods have introduced rotation into diffractive neural networks, enabling certain layers to be rotatable and thereby enhancing task adaptability [13], [14]. Prior work [18] also explored training different tasks on separate regions of the diffractive layer. This approach suffers from area limitations, constrained to a limited number of fixed functions (e.g., 2-3). Similarly, metasurface-based DONNs [17] leverage polarization-dependent phase responses to train multiple tasks, with each task corresponding to a different polarization state that the metasurface can distinguish. This approach faces limited expressivity as only a small number of orthogonal polarization states are supported by metasurfaces.

IV. Proposed Composable Hybrid Optical Reconfigurable Diffractive Framework CHORD

We introduce a hybrid DONN with multi-dimensional reconfigurability and physical composability, dubbed CHORD (see Fig. 2). By delving into the system-level learnability, CHORD introduces new degrees of freedom to overcome the reconfigurability limitations of traditional DONNs, enabling superior task adaptability and training efficiency.

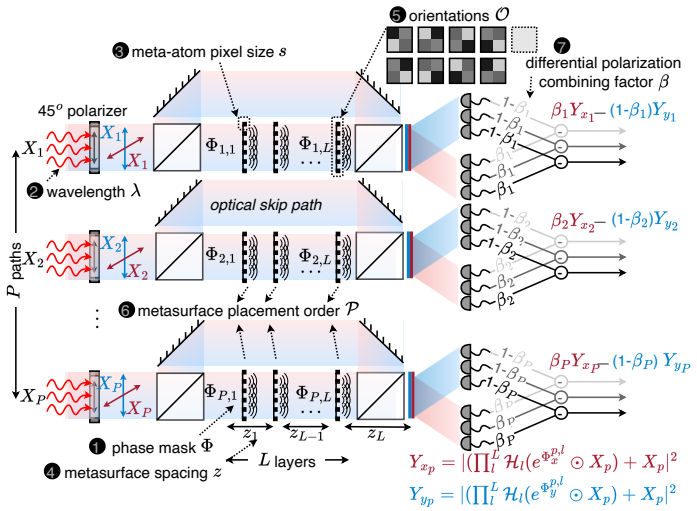


Fig. 2: Multi-path diffractive layer (DiffLayer) with P paths. Each has L cascaded metasurfaces and polarization-differential photodetection. Learnable parameters ❶-❼ are marked.

TABLE I: Parameter tuning mechanism (structural, electrical, mechanical) and reconfiguration frequency. ❶-❼ belong to diffractive optical hardware. ❽ is on integrated photonic chips.

Learnable System Parameters	Stru.	Elec.	Mech.	Reconfig. Freq.
❶ Phase Response Φ	√			Fixed after fab.
❷ Source Laser Wavelength λ		√		Tunable per task
❸ Meta-atom Pixel Size s	√			Fixed after fab.
❹ Metasurface Spacing z			√	Tunable per task
❺ Orientation $\mathcal{O}$			√	Tunable per task
❻ Placement Order $\mathcal{P}$			√	Tunable per task
❼ Differential Polarization Factor β		√		Runtime tunable
❽ Channel-Mixing Factors α_{pre}, α_{post}		√		Runtime tunable

A. Overview of Learnable System Parameters

For a generic expressive DONN, our CHORD is assumed to have a **multi-path** architecture with P parallel paths, each with L cascaded phase masks, shown in Fig. 2. We summarize programming mechanisms and properties of all variables in our CHORD architecture in Table I, including the *conventional* parameter phase masks Φ, and our **7 newly introduced dimensionalities**: light source wavelength λ, meta-atom pixel size s, diffraction distance between metasurfaces z, metasurface orientation $\mathcal{O}$, metasurface placement ordering $\mathcal{P}$, pre- and post-channel-mixing factor α_{pre} and α_{post}, and differential polarization combining factor β, visualized in Fig. 2.

1) ❶ Phase Masks Φ: Phase masks are the core parameters of DONNs. If phase elements (meta-atoms) are designed to be polarization-dependent, e.g., a rectangle, we can adjust meta-atom width/length to independently change its phase responses $\phi_x, \phi_y \in [0, 2\pi)$ for x- and y-polarized light. Once a phase mask is trained and fabricated, it is fixed and typically cannot be reconfigured.

2) ❷ Light Source Wavelength λ: The wavelength λ impacts the optical system's diffraction behavior $\mathcal{H}(\cdot|\lambda)$ while maintaining a near-constant phase response ϕ if using broadband meta-atoms. A reasonable range for λ is from 400 nm in the visible spectrum to 1600 nm in the near-infrared. Given a preferred λ, e.g., 532 nm, we consider local fine-tuning with $\pm$20 nm linewidth given the tunability of the current laser with a 0.1 nm tuning resolution.

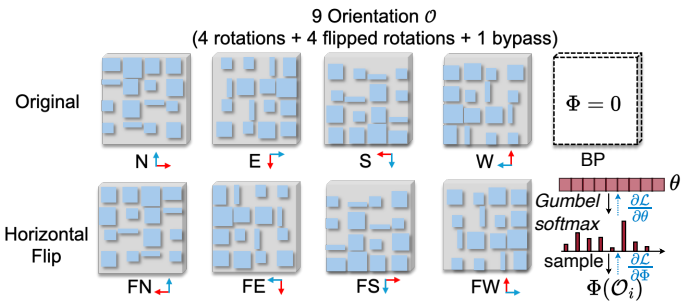


Fig. 3: 9 orientation states for each bi-directional phase mask, which can be learned differentiably using Gumbel-softmaxd.

3) ❸ *Meta-Atom Pixel Size s:* The diffraction operator $\mathcal{H}(\cdot)$ is a function of pixel size s, which is the period of the metasurface grid. Considering all metasurfaces have a shared scalar pixel size $s \in \mathbb{R}$, e.g., 300 nm, we are allowed to freely determine this variable during the initial design stage for better performance. The pixel size is lower-bounded by the meta-atom width required to realize 2π phase response plus minimum manufacturable linewidth, e.g., $s > s_{2\pi,\lambda_0} \cdot \lambda/\lambda_0 + \Delta s$. For example, given a reference λ_0=532 nm, the meta-atom width needs to be $s_{2\pi,\lambda_0}$=180 nm to realize 2π phase shift, then we assume a linear scaling w.r.t. λ. With a Δs=20 nm gap between pixels, the pixel period needs to honor the constraint $s > 200$ nm.

4) ❹ *Metasurface Spacing z:* The spacing between two phase masks determines the diffraction behavior of the light, i.e., $\mathcal{H}(\cdot|z)$ Intuitively, a small z passes the light directly to the next metasurface with weak diffraction and cross-pixel interactions, yielding a local linear operator with a nearly diagonal transfer matrix. A large spacing introduces all-to-all pixel interaction. However, most light spreads to spaces outside the next metasurface plane, which causes significant light energy loss and, thus, a dim and blurred image projected onto the next metasurface. Hence, a carefully optimized diffraction spacing z is critical to the receptive field and expressivity of the DONN. We explore a shared learnable spacing z for all layers and independently adjustable spacings $(z_1, z_2, \cdots, z_L)$ for different layers.

5) ❺ *Metasurface Orientation $\mathcal{O}$:* Before metasurface fabrication, the phase mask is freely optimized to realize an arbitrary phase distribution. After fabrication, the metasurface is fixed. Thanks to the physical composability of our CHORD design, the phase mask orientation becomes an additional degree of freedom to enrich the functionality while reusing the same piece of hardware, as shown in Fig. 3. The phase mask can be rotated at four angles, i.e., R0 (N), R90 (E), R180 (S), and R270 (W). Besides, a bidirectional phase mask can also be horizontally flipped to introduce chirality, i.e., shine the light reversely through the mask. Then, we can augment to another 4 states: FN, FE, FS, and FW, shown in Fig. 3. Replacing the phase mask with an identity mask $\Phi = 0$ is another allowable *bypass* state given the system flexibility, denoted as BP. Orientation reconfiguration can be implemented mechanically. Each metasurface can select its orientation out of 9 states. For a DONN with P parallel paths, each with L cascaded masks, there are 9^{PL} unique states. Though this approach might not be suitable for real-time frequent reprogramming, it is promising for less frequent task adaptation. It enables **exponentially many new system responses for potential lifelong hardware reusing**. Later, we will introduce how to learn the best state from discrete orientations in a differentiable way.

6) ❻ *Metasurface Placement Order $\mathcal{P}$:* To fully leverage the physical composability of our CHORD system, we further explore the location swapping of metasurfaces for extreme multi-functionality. Inspired by the concept of card shuffling, phase masks can be reordered and replugged to any valid slots. A permutation of all phase masks can be used to describe this placement ordering. In a $P \times L$ multi-path DONN, the total permutation reaches $(PL)!$, enabling an even larger design space than orientation.

7) ❼ *Differential Polarization Combining Factor β:* Polarization-multiplexed DONN [17] is previously demonstrated to map two tasks onto orthogonal polarization channels. Differently, we use two independent polarization directions in a **differential fashion to enhance the linear operation expressivity and realize full-range outputs**. Each forward pass through our CHORD system generates two non-negative outputs Y_x and Y_y for x(y)-polarized channel, respectively. We use balanced photodetection to obtain a full-range feature map $Y = \beta Y_x - (1-\beta)Y_y$ with *a non-negative combining factor* $\beta \in \mathbb{R}_+$. β is electronically implemented with low overhead and thus can be dynamically reconfigured in real time.

8) ❽ *Pre/Post Channel-Mixing Coefficients α_{pre} and α_{post}:* Simulating such systems in a multi-channel DONN layer is time-consuming and memory-hungry. With C_{in} input feature map channels and C_{out} output channels, it requires to simulate the P-path CHORD system by $C_{in}C_{out}/P$ times, which is not scalable. To make DONN training more scalable and efficient, we borrow the concept of depth-wise convolution, which performs **spatial-only channel-wise diffractive projection** without aggregating the computing results from C_{in} channels. To enable channel-wise feature extraction, we add a channel-mixing coefficient α_{pre} and α_{post} before and after the metasurface DONN layer, same as point-wise convolution. The pre-channel-mixing layer projects the features from C_{in}-channel to C_{mid}-channel. The DONN system forwards C_{mid} channels of images through the metasurface system and obtains C_{mid} output channels, which are further fed into α_{post} layer and projected to C_{out} channels.

As the channel-mixing operation only involves parallel MVM, it can be efficiently mapped to integrated PTCs for ultra-fast processing with real-time reconfigurability. By leveraging the ultra-parallel global-view spatial processing capability of diffractive optical systems and ultra-fast reconfigurable MVM power of integrated photonic accelerators, our hybrid optical-photonic CHORD system can realize **highly adaptable and efficient** optical computing. Meanwhile, this depthwise separable operation, with comparable expressivity to traditional multi-channel convolution-like DONN layers, **significantly lowers the training cost** of simulating diffractive layers.

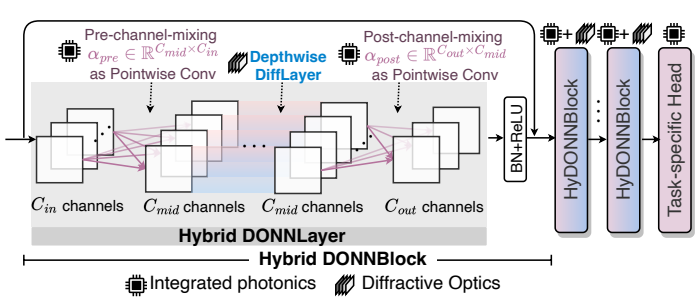


Fig. 4: Hybrid DONNLayer with pointwise Convolution mapped to integrated PTCs and depthwise diffractive layers mapped to free-space diffractive optics.

9) Hybrid DONNLayer: Figure 4 shows our hybrid DONNLayer with the channel-mixing operation mapped to integrated PTCs for ultra-fast point-wise Conv and diffractive layers (DiffLayer) in the middle

mapped to metasurface systems for ultra-parallel depth-wise spatial information extraction. `DONNResBlock` contains BatchNorm, activation, and a residual connection. Multiple `DONNResBlocks` construct the backbone of our `CHORD` system. The task-specific head at the end is mapped onto integrated PTCs.

B. Pre-Fab Training for Initial Task

In the initial training phase before metasurface fabrication, all system variables can be freely optimized. Note that since the phase response of the metasurfaces Φ can be freely adjusted, it is unnecessary to learn the orientation or placement order at this stage. The permutation $\mathcal{P}$ and metasurface rotation mask $\mathcal{M}$ will not be optimized during this phase. Therefore, we focus on optimizing parameters such as the Φ, α_{pre}, α_{post}, β, λ, z, and s on the initial machine learning task.

$$\min_{\Phi,\lambda,z,s,\alpha,\beta} \mathcal{L}((\Phi,\lambda,z,s,\alpha,\beta),\mathcal{D}_{init}). \tag{2}$$

C. Post-Fab Training for Tasks Adaptation

Once the first training is completed, DONN phase masks will be fabricated. Phases Φ and pixel size s are fixed. We now introduce how to enable fully-differentiable learning of the rest of the architecture variables to adapt the fabricated system to new tasks $\mathcal{D}_{adapt}$,

$$\begin{aligned} \min_{\lambda,z,\alpha,\beta,\mathcal{O},\mathcal{P}} \ & \mathcal{L}((\lambda,z,\alpha,\beta,\mathcal{O},\mathcal{P}),\mathcal{D}_{adapt}|\Phi^*,s^*) \\ \text{s.t.} \quad & \mathcal{P} \in \text{Permutation matrices} \\ & \lambda^* - \Delta\lambda \le \lambda \le \lambda^* + \Delta\lambda, \quad \beta \ge 0. \end{aligned} \tag{3}$$

1) Adapt Wavelength λ: Similar to the initial training, we allow local adjustment of wavelength with $\pm$20nm range to adapt the diffraction operation to new tasks.

2) Adapt Metasurface Spacing z: Similar to the initial training stage, we can either learn a shared spacing $z \in \mathbb{R}$ or a dedicated layer-wise spacing vector $\mathbf{z} \in \mathbb{R}^L$ with the standard gradient descent method.

3) Learn Metasurface Orientation $\mathcal{O}$ with Gumbel-Softmax Method: Selecting the orientation of metasurfaces is a highly discrete problem. We model the orientations of each metasurface as a 9-category parametric distribution. For the (p,l)-th metasurface, its probability of having orientation $\mathcal{O}_{p,l} \in \mathbf{O} = \{N, W, S, E, FN, FW, FS, FE, BP\}$ is

$$P_{\theta_{p,l}}(\mathcal{O}_{p,l} = \mathbf{O}[i]) = e^{\theta^i_{p,l}} \Big/ \sum_{i=1}^{9} e^{\theta^i_{p,l}}. \tag{4}$$

We employ Gumbel-softmax to learn the distribution parameter θ with a gradient-based method in Fig. 3. The phase mask Φ is the superposition of all 9 states weighted by a mask m, i.e., $\Phi_{p,l} = \sum_{i=1}^{9} m^i_{p,l}\Phi^i_{p,l}$, where the mask is sampled from the following distribution,

$$m^i_{p,l} = \texttt{GumbelSoftmax}(\theta^i_{p,l}|\theta_{p,l}) = \frac{e^{(\theta^i_{p,l}+g^i_{p,l})/\tau}}{\sum_i e^{(\theta^i_{p,l}+g^i_{p,l})/\tau}}, \tag{5}$$

where g and τ are the Gumbel noise and temperature. The final orientation $\mathcal{O}_{p,l}$ can be selected by $\mathcal{O}_{p,l} = \mathbf{O}[\text{argmax}_i\, \theta^i_{p,l}]$.

4) Learn Metasurface Placement Order $\mathcal{P}$ with Differentiable Permutation Learning: We introduce an automatic learning method to permute all PL metasurfaces in a differentiable way. Without loss of generality, we use the learning of a permutation across P metasurfaces as an example. Exploring the factorial permutation space is very challenging. Inspired by a differentiable circuit routing learning algorithm [23], as illustrated in Fig. 5, we use a $P \times P$ permutation matrix $\mathcal{P}$ to represent the mapping from metasurface indices to slot indices. A permutation matrix is a Boolean matrix $\mathcal{P} \in \{0,1\}^{P\times P}$ where each row and column only has one 1. $P_{ij} = 1$ means the i-th metasurface is placed at slot j. We convert the permutation constraint with its equivalent continuous format, i.e., the row-wise and column-wise ℓ_1-norm equals the squared ℓ_2-norm. We solve the constrained optimization problem with the augmented Lagrangian method (ALM) to relax the constraint as an objective term,

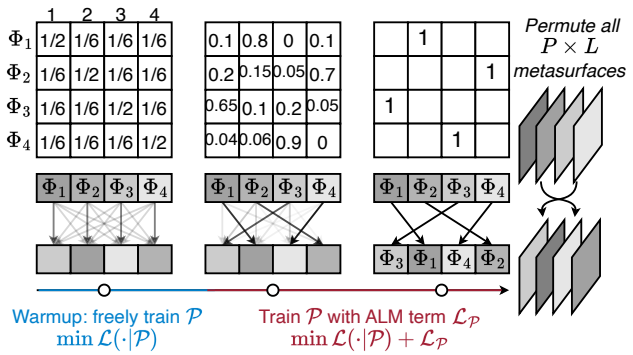


Fig. 5: Illustration of metasurface placement order permutation. Two-stage training is adopted with warmup training and ALM-based permutation learning.

$$\mathcal{L}_{\mathcal{P}} = \sum_{i=1}^{P} \mu^T_{i,r}\big(\Delta\widetilde{\mathcal{P}}^{i,:} + \frac{\rho}{2}(\Delta\widetilde{\mathcal{P}}^{i,:})^2\big) + \mu^T_{i,c}\big(\Delta\widetilde{\mathcal{P}}^{:,j} + \frac{\rho}{2}(\Delta\widetilde{\mathcal{P}}^{:,j})^2\big), \tag{6}$$

where $\mu^r \in \mathbb{R}^P$ and $\mu^c \in \mathbb{R}^P$ are row-wise and column-wise Lagrangian multipliers, and ρ is the quadratic penalty weight. $\Delta\widetilde{\mathcal{P}}^{i,:} = \|\widetilde{\mathcal{P}}^{i,:}\|_1 - \|\widetilde{\mathcal{P}}^{i,:}\|_2^2$. At every step when we update $\mathcal{P}$, we also perform the dual update on the multiplier $\mu \leftarrow \mu + \rho(\Delta\widetilde{\mathcal{P}}^{i,:} + \frac{1}{2}(\Delta\widetilde{\mathcal{P}}^{i,:})^2)$ to gradually increase the penalty to push it to a feasible permutation.

5) Train Task-specific Head Layers: After the backbone, task-specific head layers, such as the final linear classifier or convolutional regression layers, are re-initialized and trained on the new task.

V. Evaluation Results

A. Settings

1) Model and Dataset: As a case study, we assume our `CHORD` system has up to 4 paths and up to 4 cascaded metasurface layers. The DONN model backbone based on this hardware contains two Hybrid `DONNBlocks`, with depthwise `DiffLayers` mapped to the same `CHORD` hardware. All structural and mechanical parameters that cannot be switched in runtime, i.e., λ, z, s, Φ, $\mathcal{O}$, $\mathcal{P}$, **are shared** across all `DiffLayers`. Runtime reconfigurable parameters, i.e., α_{pre}, α_{post}, and β, **are independently learned** for each `DiffLayer`. Each metasurface has 32×32 meta-atoms. The initial values for wavelength $\lambda = 532$ nm, meta-atom size $s = 400$ nm, and metasurface spacing $z = 8.42\ \mu m$ are typical values used in the literature [17]. We evaluate on three benchmarks:

- Pre-fab training on Fashion-MNIST [24] 10-class recognition and adapt the model to QuickDraw [25] 10-class image recognition. A classification head is used with AdaptiveAvgPool5-FC10.
- Pre-fab training on CIFAR-10 [26] and adapt the model to CIFAR-100 [26]. The classification head is Adaptive AvgPool5-FC10(100).
- Pre-fab training on Darcy flow [27] 1-channel PDE solving and adapt to Navier-Stokes [28] 10-channel PDE solving. A regression head is used with C64K3-BN-GELU-C1(10)K1.

2) Training Settings: General Setting: We train all DONNs for 100 epochs using the AdamW [29] optimizer with a learning rate (lr) of 1E-2, a cosine decay scheduler, a weight decay rate of 1E-2, and data augmentation techniques such as random cropping and flipping for both

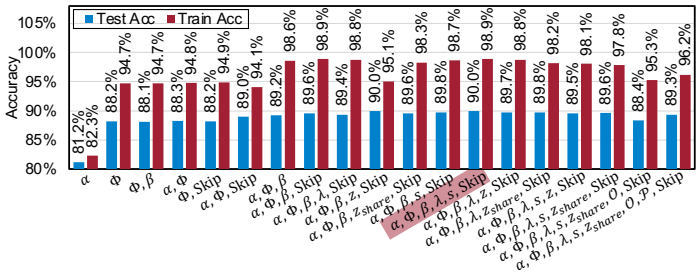
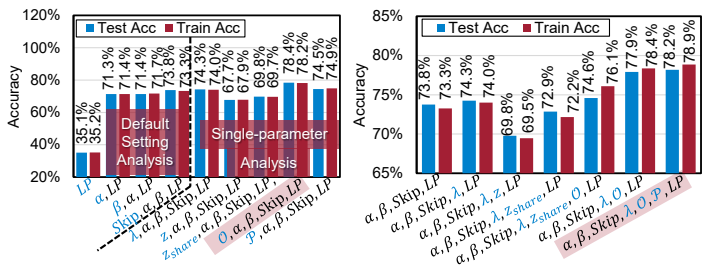


Fig. 6: Expressivity of different combinations of learnable system parameters. Training and test accuracy are shown on 2-layer DONN on FMNIST.



(a) (b)

Fig. 7: (a) Effectiveness of individual parameter (❷-❻) for task adaptation, LP: Linear Probing. (b) Effectiveness of progressively enabling parameters (❷-❻) in adaptation.

pre-fabrication training tasks and adaptation tasks. Weight decay is only applied to Φ, which is reparameterized in the range of $[-\pi, \pi)$.

Schedule τ in Orientation and ρ in Permutation: For orientation learning, to prioritize early-stage exploration and later-stage stable convergence, we apply exponential decay on the Gumbel-softmax temperature $\tau = \tau_{init} \cdot \gamma_\tau^{(epoch-1)}$, where τ_{init} is 5 and γ_τ is 0.956. To encourage early-stage free permutation optimization and gradually force the constraints, ALM quadratic penalty term ρ is scheduled as $\rho = \rho_{init} \cdot \gamma_\rho^{(epoch-warm\ up\ epoch)}$, with a warm-up period of 5 epochs, ρ_{init} set to 1E-5, and γ_ρ set to 1.2.

Parameter Quantization: λ can be adjusted within ± 20 nm around initial value, with 1 nm precision. To ensure that the meta-atom can generate a $-\pi$ to π phase response for a given λ, the pixel size's lower bound is calculated following Sec. IV-A3 equation: $s > \frac{\lambda}{\lambda_0} \cdot s_{2\pi,\lambda_0} + \Delta_s$, where $\lambda_0 = 532$ nm, $s_{2\pi,\lambda_0} = 300$ nm, and $\Delta_s = 20$ nm. The precision for z is set to 10 nm, while Φ is reparameterized $\in [-\pi, \pi)$ with a precision of 1°. We also employ learned step-size quantization-aware training [30]. A b_w=8-bit quantization is applied to weights and activations in layers utilizing PTCs, such as α and the final head.

Evaluation Metrics: Accuracy is used as the primary metric for the classification tasks. The Mean Squared Error (MSE) was utilized as the evaluation metric for PDE-solving tasks. Both training and test performance are shown to indicate each setting and parameter's contribution to the DONN's expressivity, trainability, and generalizability.

B. Ablation Study on CHORD's Learnable Parameters

1) Parameter Expressivity and Learnability in Pre-fab Training: In initial training, we conduct a thorough ablation study on parameters in Fig. 6. A single-path DONN is assumed ($P = 1$, $L = 4$). We train (α, Φ, β, λ, s) with the optical skip path enabled as the optimal default pre-fab training settings.

2) Individual Effectiveness of Params. ❷-❼ in Post-fab Task Adaptation: We explore the impact of each parameter on task adaptation

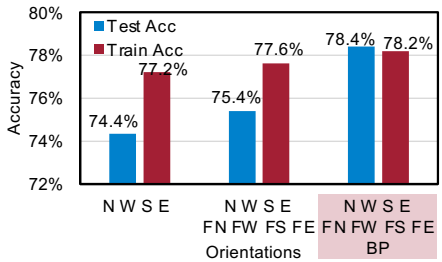


Fig. 8: Nine orientations show the best expressivity and accuracy on QuickDraw adaptation tasks.

TABLE II: Evaluate CHORD (P=4, L=4) on pre-fab initial tasks and post-fab adapted tasks. A 2-layer digital CNN (C32K3)$_{\times 2}$ is shown as a reference. In each group, DONN is trained on the first task and adapted to the second task. We use initial training results of the second task as a reference. For classification, test Acc. is shown. For Darcy/NavierStokes, MSE is shown.

Tasks	Digital 2-Layer CNN C32K3-C32K3	Pre-fab Initial Training P=4, L=4, α, Φ, β, *Skip*, s, λ	Post-fab Task Adaptation P=4, L=4, α, β, *Skip*, λ, $\mathcal{O}$, $\mathcal{P}$
FMNIST	0.9122	0.9014	-
Quickdraw	0.8978	0.8646	0.7992 (adapt from FMNIST)
CIFAR10	0.7633	0.6182	-
CIFAR100	0.4347	0.3815	0.2513 (adapt from CIFAR10)
Darcy	0.0466	0.055	-
NavierStokes	0.1085	0.1007	0.1753 (adapt from Darcy)

in Fig. 7(a). Simply training the head, called linear probing (LP), with the backbone frozen, is insufficient. As we enable the optimization of *electrically* tunable parameters (β and α), along with the optical skip path and linear probing, we observe a significantly enhanced accuracy.

Mechanically tunable parameters (λ, z, $\mathcal{O}$, $\mathcal{P}$) further improve task performance, with phase mask orientation $\mathcal{O}$ showing the largest impact due to its additional $9^4\times$ larger solution space. Permutation $\mathcal{P}$ also contributes positively to task adaptation, though not as impactful as orientation due to the 4! solution space with only 4 metasurfaces (single path), $273\times$ smaller than $\mathcal{O}$. The effectiveness of $\mathcal{P}$ is expected to factorially grow as the number of paths increases.

The diffractive distance z and z_{share} contribute less to performance, as they are too sensitive to be trained alongside other parameters, leading to an increase in training difficulty and thus impacting the overall optimality, which is not trained during adaptation.

3) Combinational Effectiveness of Params. ❷-❼ in Post-fab Task Adaptation: Building on the individual parameter analysis, we progressively enable these parameters to evaluate their combined effectiveness in task adaptation, as shown in Fig. 7(b). The results are consistent with those in Sec. V-B2, confirming that the diffractive distance z is highly sensitive and tends to reach suboptimal states when trained alongside other parameters. However, fixing z to its initial learned value significantly improves adaptation accuracy as additional parameters, λ, $\mathcal{O}$, and $\mathcal{P}$, are progressively introduced.

4) Orientation Effectiveness: Figure 8 provides strong evidence for the effectiveness of the 9-option orientations. Compared to 4-direction rotation [13], [14], the inclusion of 4 flipped states and the identity state improves test accuracy by +4%, demonstrating their significant contribution to CHORD's task adaptability.

C. Multi-Path, Multi-Layer Exploration

We investigate the impact of multiple parallel paths in CHORD, exploring a more generic design space to enable greater functionality.

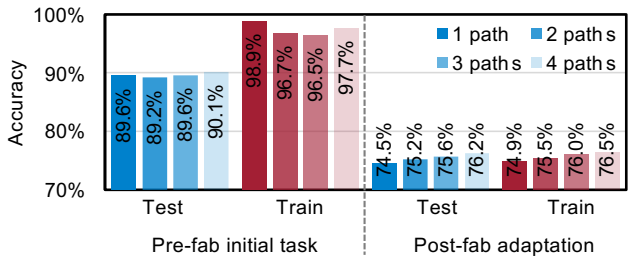


Fig. 9: Adapt four DONNs with L=4 and different paths P from pre-fab FMNIST task to post-fab Quickdraw task. Only permutation $\mathcal{P}$ is trained during adaptation to show the learnability gain by enlarging the solution space from 4! to 16!.

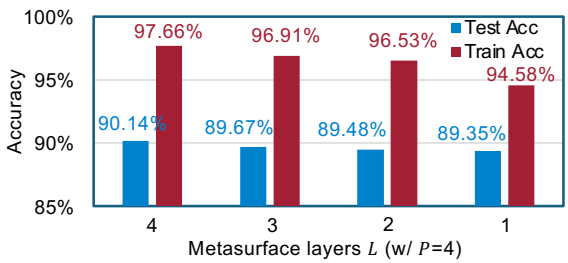


Fig. 10: Different layers L with P=4 on pre-fab FMNIST.

1) Number of Paths P in Pre-Fab Initial Training: In Fig. 9, we fix the layer count to L=4 and increase the number of diffractive paths P. Test accuracy on the initial task improves by approximately 0.5% with an increasing number of paths. However, fluctuations in training accuracy suggest that adding too many parallel phase masks, similar to large-kernel convolutions, introduces learning challenges.

2) Number of Paths P in Post-Fab Task Adaptation with Permutation: In Fig. 9(*Right*), we show task adaptation only with permutation learning enabled. By increasing the path from 1 to 4, the space of permutation increases from 4! to 16!, leading to a notable learnability boost and thus +1.7% improvement on test and training accuracy.

3) Number of Layers L in Pre-Fab Initial Training: We evaluate the impact of layer depth L on learnability by conducting pre-fab training with 4 parallel paths and varying the number of layers L. Figure 10 shows a significantly enhanced expressivity with more layers cascaded, demonstrating the clear benefit of deeper CHORD.

D. Main Results

We conduct three pre-fab training tasks on Fashion-MNIST, CIFAR-10, and Darcy, and transfer the CHORD system to new tasks, i.e., QuickDraw, CIFAR-100, and Navier-Stokes, respectively. Table II demonstrates that CHORD exhibits competitive expressivity compared to the digital counterpart in FMNIST and QuickDraw. However, we observe accuracy gaps on CIFAR-10 and CIFAR-100 due to the increased difficulty of those tasks. Notably, CHORD **excels in much harder PDE prediction tasks such as Darcy flow and Navier-Stokes. These tasks benefit from a global receptive field, aligning perfectly with the global spatial processing capabilities of diffractive ONNs**.

For task adaptation, when all system variables ❶-❽ are enabled to learn, we can largely resume the accuracy even when the major weights Φ are frozen. To understand the individual contribution of each variable to task adaptability, we show a detailed discussion below.

E. Discussion

1) Parameter Contribution to Adaptability: Figure 11 shows the insights of parameter contribution to adaptability.

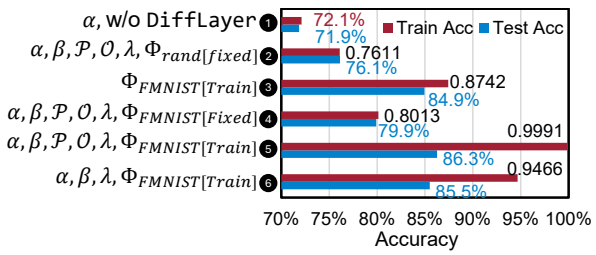


Fig. 11: Training different parameters during adaptation.

TABLE III: Device info in energy and latency calculation [12].

Device	8-bit ADC	8-bit DAC	TIA	MZM	PS	PD
Power (mW)	14.8	50	3	50	0.15	0

Compare (❶, ❹): Besides α_{pre}, α_{post} on integrated photonics, DiffLayers are also critical in DONN learning.

Compare (❷, ❹): The trained Φ from initial tasks carries transferable knowledge that helps new task learning.

Compare (❹, ❺): In the future, with advancements in *reconfigurable metasurfaces*, enabling Φ to be trainable on new tasks post-fabrication could significantly enhance performance, unlocking even greater potential for dynamic task adaptability and learnability.

Compare (❸, ❺): We want to highlight that even if Φ becomes trainable in the future, our introduced $\alpha, \beta, \mathcal{P}, \mathcal{O}, \lambda$ parameters can still enhance the network's expressivity and learnability.

Compare (❺, ❻): It validates that our introduced permutation $\mathcal{P}$ and orientation $\mathcal{O}$ effectively expand the solution space and are crucial for enhancing the model's adaptability.

2) Training Efficiency of Our Depth-wise DiffLayer: Our depth-wise DiffLayer achieves significant efficiency improvements over prior DONNs with 64-channel Conv-like diffractive layers, reducing memory by 29% (from 9807 MB to 6997 MB) and accelerating training by 4.92× per iteration (from 329.59 ms to 66.99 ms).

3) Energy and Speed Advantages of CHORD vs. Digital CNNs: Via behavior-level architecture simulation, on average, CHORD delivers 74× faster single-batch inference speed (1.48 ms to 0.02 ms) and consumes 194× less energy (134.9 mJ to 0.696 mJ) compared to a digital CNN on A6000 GPU. For the DONN part, we assume a photodetection (PD) speed of 10 GHz [31]. Given that P channels are processed in parallel, this results in an 8E-7 ms latency with 32/P=8 (P=4) cycles and 0.1 ns per cycle. Thus, the latency of the diffractive layer is negligible compared to the overall time.

For the PTC part, we employ a single-core 16×16 PTC design [12], with device power listed in Table III. The energy and latency are simulated using an open-source photonic accelerator simulator [20].

VI. Conclusion

We present the first in-depth analysis of multi-dimensional learnability in DONNs, introducing a physically composable hybrid optical/photonic system, CHORD. Our results demonstrate that CHORD with multi-dimensional learnable system variables provides digital-comparable accuracy on various task adaptations, especially challenging PDE-solving tasks, with 74× faster inference speed and 194× lower energy. Compared to prior DONNs, our hybrid CHORD shows exponentially larger functional space and extreme multi-functionality with 5× faster training speed. CHORD shows the potential to enable versatile and high-efficiency optical AI systems with repurposed hardware, paving the way for practical deployment in dynamic AI applications.

REFERENCES

[1] Tao Yan, Jiamin Wu, Tiankuang Zhou, Hao Xie, Feng Xu, Jingtao Fan, Lu Fang, Xing Lin, and Qionghai Dai. Fourier-space diffractive deep neural network. *Phys. Rev. Lett.*, 123:023901, Jul 2019.

[2] Kaixuan Wei, Xiao Li, Johannes Froech, Praneeth Chakravarthula, James Whitehead, Ethan Tseng, Arka Majumdar, and Felix Heide. Spatially varying nanophotonic neural networks, 2023.

[3] Jingyang Peng, Li Fang, Min Gu, and Qiming Zhang. Dynamically reconfigurable all-optical neural network based on a hybrid graphene metasurface array. *Opt. Continuum*, 3(5):704–713, May 2024.

[4] Hang Chen, Jianan Feng, Minwei Jiang, Yiqun Wang, Jie Lin, Jiubin Tan, and Peng Jin. Diffractive deep neural networks at visible wavelengths. *Engineering*, 7(10):1483–1491, 2021.

[5] Xing Lin, Yair Rivenson, Nezih T Yardimci, Mert Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018.

[6] Yichen Shen, Nicholas C. Harris, Scott Skirlo, et al. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, 2017.

[7] Q. Cheng, J. Kwon, M. Glick, M. Bahadori, L. P. Carloni, and K. Bergman. Silicon Photonics Codesign for Deep Learning. *Proceedings of the IEEE*, 2020.

[8] Bhavin J. Shastri, Alexander N. Tait, et al. Photonics for Artificial Intelligence and Neuromorphic Computing. *Nature Photonics*, 2021.

[9] Alexander N. Tait, Thomas Ferreira de Lima, Ellen Zhou, et al. Neuromorphic photonic networks using silicon photonic weight banks. *Sci. Rep.*, 2017.

[10] Xingyuan Xu, Mengxi Tan, Bill Corcoran, Jiayang Wu, Andreas Boes, Thach G. Nguyen, Sai T. Chu, Brent E. Little, Damien G. Hicks, Roberto Morandotti, Arnan Mitchell, and David J. Moss. 11 TOPS photonic convolutional accelerator for optical neural networks. *Nature*, 2021.

[11] H.H. Zhu, J. Zou, H. Zhang, et al. Space-efficient optical computing with an integrated chip diffractive neural network. *Nature Commun.*, 2022.

[12] Ziang Yin, Nicholas Gangi, Meng Zhang, Jeff Zhang, Rena Huang, and Jiaqi Gu. Scatter: Algorithm-circuit co-sparse photonic accelerator with thermal-tolerant, power-efficient in-situ light redistribution. In *International Conference on Computer-Aided Design (ICCAD)*, 2024.

[13] Yingjie Li, Weilu Gao, and Cunxi Yu. Rubik's optical neural networks: multi-task learning with physics-aware rotation architecture. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023.

[14] Yingjie Li, Ruiyang Chen, Weilu Gao, and Cunxi Yu. Physics-aware differentiable discrete codesign for diffractive optical neural networks. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery.

[15] Xing Lin, Yair Rivenson, Nezih T. Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018.

[16] Joseph W Goodman. *Introduction to Fourier Optics*. Roberts and Company Publishers, 3rd edition, 2005.

[17] Xuhao Luo, Yueqiang Hu, Xiangnian Ou, Xin Li, Jiajie Lai, Na Liu, Xinbin Cheng, Anlian Pan, and Huigao Duan. Metasurface-enabled on-chip multiplexed diffractive neural networks in the visible. *Light: Science & Applications*, 11(1):158, May 2022.

[18] Hanyu Zheng, Quan Liu, Ivan I. Kravchenko, Xiaomeng Zhang, Yuankai Huo, and Jason G. Valentine. Multichannel meta-imagers for accelerating machine vision. *Nature Nanotechnology*, 19(4):471–478, Apr 2024.

[19] Zhihao Xu, Tiankuang Zhou, Muzhou Ma, ChenChen Deng, Qionghai Dai, and Lu Fang. Large-scale photonic chiplet taichi empowers 160-tops/w artificial general intelligence. *Science*, 384(6692):202–209, 2024.

[20] Hanqing Zhu, Jiaqi Gu, Hanrui Wang, Zixuan Jiang, Zhekai Zhang, Rongxin Tang, Chenghao Feng, Song Han, et al. Lightening-transformer: A dynamically-operated photonic tensor core for energy-efficient transformer accelerator. In *Proc. HPCA*, 2024.

[21] Chenghao Feng, Jiaqi Gu, Hanqing Zhu, Zhoufeng Ying, Zheng Zhao, et al. A compact butterfly-style silicon photonic–electronic neural chip for hardware-efficient deep learning. *ACS Photonics*, 9(12):3906–3916, 2022.

[22] Johannes Feldmann, Nathan Youngblood, Maxim Karpov, Helge Gehring, Xuan Li, Maik Stappers, Manuel Le Gallo, Xin Fu, Anton Lukashchuk, Arslan Raja, Junqiu Liu, David Wright, Abu Sebastian, Tobias Kippenberg, Wolfram Pernice, and Harish Bhaskaran. Parallel convolutional processing using an integrated photonic tensor core. *Nature*, 2021.

[23] Jiaqi Gu, Hanqing Zhu, Chenghao Feng, Zixuan Jiang, Mingjie Liu, Shuhan Zhang, Ray T. Chen, and David Z. Pan. Adept: Automatic differentiable design of photonic tensor cores. In *Proc. DAC*, 2022.

[24] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *Arxiv*, 2017.

[25] Google. The quick, draw! dataset, 2017.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[27] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.

[28] Hyung-Chun Lee John Burkardt, Max Gunzburger. 2d navier stokes flow datasets. 2006.

[29] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

[30] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *Proc. ICLR*, 2020.

[31] Maxwell G. Anderson, Shi-Yuan Ma, Tianyu Wang, Logan G. Wright, and Peter L. McMahon. Optical transformers, 2023.

BirdMoE: Reducing Communication Costs for Mixture-of-Experts Training Using Load-Aware Bi-random Quantization

Donglei Wu*†‡, Weihao Yang§, Xiangyu Zou§, Jinda Jia‖, Dingwen Tao¶, Wen Xia§, Zhihong Tian*†‡
*Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, China
†Guangdong Key Laboratory of Industrial Control System Security, Guangzhou, China
‡Huangpu Research School of Guangzhou University, Guangzhou, China
§School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China
¶Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
‖Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN, USA

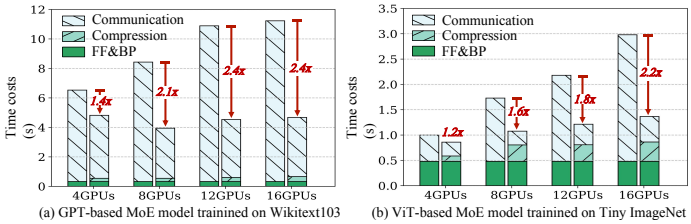
***Abstract*—Mixture-of-Experts (MoE) model parallelism is prevalent in training Large Language Models (e.g., ChatGPT). However, the intensive *all-to-all* collective communication of the MoE layer's intermediate computing results substantially degrades MoE training efficiency. In this paper, we propose BirdMoE, a novel load-aware communication compression technique with Bi-random quantization for MoE training with two core modules. Specifically, BirdMoE employs a lightweight Random Quantization (RQ) with expectation invariance property to efficiently map the floating-point intermediate computing results into integers while maintaining the MoE training quality. Additionally, BirdMoE utilizes a Mixed Precision (MP) strategy to dynamically balance the communication loads among expert nodes, significantly improving all-to-all communication efficiency for the MoE training system. Experiments on four typical MoE training tasks demonstrate that BirdMoE achieves higher 4.06×-10.44× total communication compression ratios and 1.18×-5.27× training speedup compared with the state-of-the-art compression techniques while maintaining the MoE training quality.**

***Index Terms*—Mixture-of-Experts training, communication compression, load balance.**

I. Introduction

The remarkable success of deep learning can be attributed to the combination of large-scale training datasets and increasingly expansive model architectures. For example, in domains like language generation, machine translation, and computer vision, augmenting model capacity significantly boosts model performance when datasets are suitably large [1]–[5]. However, dense neural networks process every sample using all parameters, resulting in escalating training costs. This includes increased demands on memory and computing power, proportionate to the growth in both model and dataset scales [6].

Mixture-of-Experts (MoE), a form of conditional computation technique, has recently proven effective in expanding model capacity with sub-linear cost [7]. Different from directly scaling small models to a large dense model, the MoE model consists of many small feed-forward networks (FFN), which are named *experts*. Training samples are dynamically selected by a trainable gate network and fed into different experts. Because experts are activated sparsely, MoE notably boosts the number of samples trained within the same time, improving model accuracy with lower computational expenses compared to conventional dense models [8]–[10].



(a) GPT-based MoE model trained on Wikitext103 (b) ViT-based MoE model trained on Tiny ImageNet

Fig. 1. The runtime breakdown per iteration of naive no-compression MoE training (left bar) and BirdMoE training (right bar).

Communication Bottleneck in MoE: Although the flexible MoE structure makes it more feasible to train a giant model beyond a trillion scale, the training process is still extremely costly. For instance, training a 600-billion-parameter MoE model requires 2048 TPUs and four days [11]. To improve training efficiency, *expert parallelism* is presented to distributedly train MoE models, where experts are partitioned into different workers (e.g., GPU nodes) and each worker processes a different batch of training samples in parallel. In MoE layers, the *intermediate computing results* of each worker are sent to different desired experts during the forward and backward propagation using '*all-to-all*' collective communication [11]. However, the distinct communication characteristics of MoE require the intermediate computing results to be intensively and unevenly transferred across workers in every iteration, making the imbalanced communication overheads a significant bottleneck in MoE training. To closely examine the communication bottleneck, we conduct four typical MoE training tasks on a cloud computing platform (detailed in Section IV). The left bars of Figure 1 indicate that communication times not only take up a large proportion but also increase significantly with the number of expert nodes.

Issues of Existing Solutions: To improve MoE training efficiency, several studies focused on modifying the MoE model architecture [8], [12] or optimizing the training workflow and resource scheduling [9], [10]. However, these works intrude on the standard MoE training workflow, potentially restricting their use in a wide range of practice scenarios. Additionally, as the MoE architecture scales up, the growth of the intensive traffic becomes explosive, making compressing the communication in MoE training a promising direction.

Recently, communication compression techniques, such as sparsification and quantization, have been successfully employed to reduce gradient volume in distributed learning due to the favorable compression ratios and scalability [13]–[20]. However, we noticed that the aforementioned distinct communication characteristics of MoE introduce three specific compression challenges to existing communication compression techniques: ❶ *Overhead Amplification Issue*: Forward and backward propagation of expert parallelism incurs intensive communication across multiple nodes demanding hundreds of compressions per iteration, dramatically amplifying the compression overhead up to 89.6% (as see Figure 2 (a)). ❷ *Error Propagation Issue*: Compression of intermediate computing results spans multiple MoE layers, causing information distortion to accumulate layer-by-layer during MoE training, significantly compromising 18.9%-66.7% training accuracy (as see Figure 2 (b)). ❸ *Communication Imbalance Issue*: existing communication compression techniques address all data uniformly without considering the imbalanced communication load among experts, thereby ineffective in resolving the dilemma of low resource utilization and congested all-to-all communication (as see Figure 2 (c)).

In this paper, we propose a novel load-aware **Bi-random** quantization technique tailored for **MoE** training system, named BirdMoE, to solve the aforementioned issue based on several observations and techniques as below.

To solve the **Overhead Amplification Issue**, we leverage the lightweight advantage of the random process and introduce a Probabilistic Thresholding-based Random Quantization (RQ) for MoE layer, which significantly reduces compression overheads by probabilistically mapping the floating-point intermediate computing results of the all-to-all communication into low-bit integers at a low $O(N)$ time complexity.

To solve the **Error Propagation Issue**, we exploit the convergence efficiency of unbiased representation [18], [19] and build the expectation invariance within RQ, which can preserve an unbiased estimation for the quantized intermediate computing results, thereby effectively reducing the information distortion and maintaining the MoE training quality.

To solve the **Communication Imbalance Issue**, we study the specific data routing characteristic of MoE layer and introduce a Mixed Precision (MP) strategy upon the above RQ, which can dynamically adjust the bit-widths of data routed to different experts, effectively balancing the communication load and improve efficiency to the overall MoE training system.

To the best of our knowledge, BirdMoE is the first compression technique designed for balancing and reducing the communication costs of intermediate computing results in all-to-all communication while maintaining model accuracy in the MoE training context. The right bars of Figure 1 indicate that BirdMoE successfully reduces communication costs at low overheads, making MoE training accessible in platforms with limited computing and communication resources. Our main contributions are summarized as follows:

- We identify three issues when applying communication compression techniques to MoE training: (*i*) Compressing the intensive communications incurs significant computational overhead; (*ii*) The compression error of intermediate results severely degrades the MoE training quality; (*iii*) Compression cannot address the low training efficiency caused by imbalanced communication loads.
- We propose BirdMoE, a novel load-aware MoE communication compression technique with two key modules: *Mixed Presicion* and *Random Quantization*, to dynamically balance and reduce all-to-all communication costs at low $O(N)$ time complexity. Additionally, we establish expectation invariance within BirdMoE to ensure the unbiasedness of the compressed intermediate computing results, effectively maintaining MoE training quality.
- We conduct comprehensive evaluations on mainstream MoE training tasks, involving multiple model structures and datasets at different scales. The evaluation demonstrate that compared with the state-of-the-art approaches, BirdMoE can achieve higher 4.06×-10.44× communication compression ratios with 1.18×-5.27× training speedup while maintaining the MoE training quality.

II. Background and Motivation

A. MoE Structure and observed Communication Bottleneck

MoE Structure: MoE is found to have a strong ability in modern large-scale pre-trained models. The key idea of MoE is to constitute numerous small sub-models, namely experts, to a large model, given the intuition that different small models are experts in different domains, and can be only activated when the data in its domain are input. The most common usage of MoE is to extend the MLP layer of the state-of-the-art sequences Transformer [1], [21], [22] by a large number of Feed Forward Network (FFN) (i.e., experts) [7], [11]. For each training iteration, samples are fed into a few experts, dynamically selected by a trainable gate network. because experts are sparsely activated, much computation is saved. Hence MoE can significantly increase the number of samples trained in the same period and improve the model performance compared with conventional dense models [7], [23].

Data Flow: As introduced in Section I, the MoE layer are shared across multiple worker devices (e.g., GPU), while all other non-MoE layers are replicated. To route the data (e.g., *tokens*) among expert nodes, the '*all-to-all*' collective communication is required to perform four times communication between MoE and non-MoE layers per training iteration. Specifically, during the Feed-Forward (FF) propagation, two all-to-all communications are employed to '*dispatch*' and '*combine*' the *feature activation* between source devices of data and target device of desired experts, according to the results of the *gate network*. In the Back-Propagation (BP) propagation, two reverse all-to-all communication are used to transfer the *feature gradient* of corresponding feature activation along the reverse trace. As mentioned before, these intensive and imbalanced communications dominate the overall runtime of MoE training, significantly degrading MoE training efficiency.

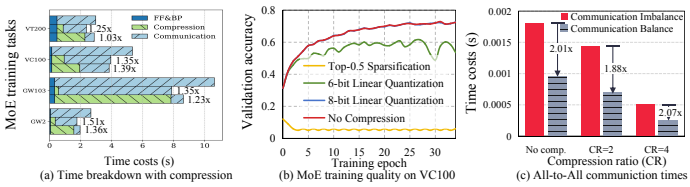


Fig. 2. Challenges of MoE communication compression. (a) Time breakdown of no-compression MoE training (top bar), linear quantization (middle bar), and Top-K sparsification (bottom bar). (b) Popular compression techniques degrade the model training quality to different degrees. (c) Compression with different levels cannot solve the communication imbalance issue.

B. Observed Compression Challenges in MoE

To examine the *Overhead Amplification* and *Error Propagation* issues discussed in Section I, we conduct various MoE training tasks on a distributed cloud-computing platform with 16 GPUs (detailed in Section IV-A). Then we apply two common communication compression techniques in distributed learning: Top-K sparsification and linear quantization, to reduce the communication costs of all-to-all stages during the MoE training.

Overhead Amplification issue: As illustrated in Figure 2 (a), time breakdown bars from up to bottom within a group represent the communication overheads of MoE training with no-compression, 8-bit linear quantization, and Top-0.1 sparsification. Experimental results show that sparsification reduces communication time by $63.5\% - 92.2\%$, but its naive implementation with $O(N \log K)$ complexity [24], [25] incurs significant compression overhead, yielding only a modest $1.03 \times - 1.39\times$ overall training speedup. Although 8-bit linear quantization has low computational overhead, it achieves a limited $4\times$ compression ratio, resulting in limited $1.25 \times - 1.51\times$ overall training speedup.

Error Propagation issue: Figure 2 (b) shows that even setting a high 0.5 sparsity for Top-K sparsification (Top-0.5), the MoE training collapse occurred, resulting in the model failing to converge. On the other hand, while low bit-width linear quantization (e.g., 6-bit) exhibits instability in training and reduced accuracy, sufficiently high bit-width (e.g., 8-bit) can ensure the MoE accuracy comparable to the no-compression counterpart (e.g., yet low compression ratio). This is because excessive compression error in intermediate computing results flow through the network, causing information distortion of feed and backward propagation.

Communication Imbalance issue: Popular expert nodes accept more data requiring longer communication time, thereby forcing other nodes to wait for synchronization and increasing the overall communication time. Figure 2 (c) shows the communication times with and without quantization in balanced and imbalanced expert communication loads. We observe that the ratio of balanced and imbalanced communication runtime remains similar under different compression ratios, indicating that evenly compress to all data does not alleviate the congestion in the imbalanced MoE communication.

C. Motivation of Our Solution

Driven by the above observations, we aim to solve the aforementioned issues by developing a load-aware communication compression technique, BirdMoE, tailored for the imbalanced and intensive all-to-all communication in MoE training based on the following motivations: ① Motivated by the lightweight computational advantages of random process and quantization operation, we introduce a lightweight random quantization algorithm to compress all-to-all communication by mapping the intermediate results to the closer ends of the quantization interval at a larger probability; ② Motivated by the convergence efficiency of unbiased gradient compression techniques in data parallelism context, we establish the expectation invariance within the above random quantization algorithm by maintaining the unbiasedness for the quantized intermediate computing results; ③ Motivation by the compression ratio of quantization is determined by the bit-width, we construct a mixed precision mechanism upon the above random quantization, which effectively improves the communication efficiency of the overall MoE training system by dynamically assigning different quantization bit-widths to the data routed to different expert nodes. The detailed methodology of BirdMoE are detailed in the next Section III.

III. Design and Implementation

A. Overview

In this paper, we propose BirdMoE, a load-aware Bi-random quantization technique for the MoE training with two key modules: *Mixed Precision* and *Random Quantization*.

1) *Mixed Precision (MP):* To balance the communication loads among expert nodes: (i) MP establishes a probabilistic negative correlation between bit-width and quantity of data routed to each expert. (ii) MP randomly assigns specific bit-widths to data routed to various experts according to a constructed probability distribution.
2) *Random Quantization (RQ):* For the intermediate computing results of experts in all-to-all communication: (i) RQ lowers compression overhead by randomly mapping the intermediate computing results to the closer integers at a larger probability. (ii) RQ maintains convergence efficiency of MoE training by building unbiasedness within the quantized intermediate computing results.

The detailed design and implementation of MP and RQ are introduced in the next subsections.

B. Mixed Precision (MP)

To balance the communication load in MoE training, MP assigns various bit-widths for the data routed to different expert nodes. The insight is that the quantization bit-width assigned to the data routed to a particular expert should be negatively proportional to the quantity.

Bit-width Assignment: For any expert node, assuming that a pre-set fixed bit-width is L (as in traditional fixed-precision quantization methods), the data quantity and bit-width routed to i-th node are N_i and L_i, respectively. As illustrated in

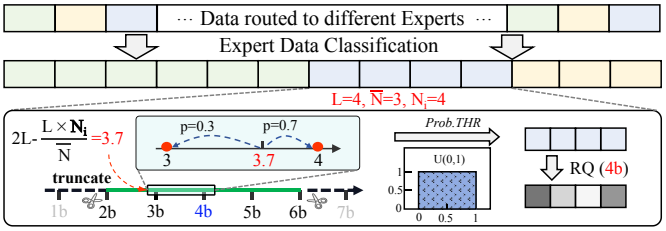


Fig. 3. A general example of Mixed Precision module.

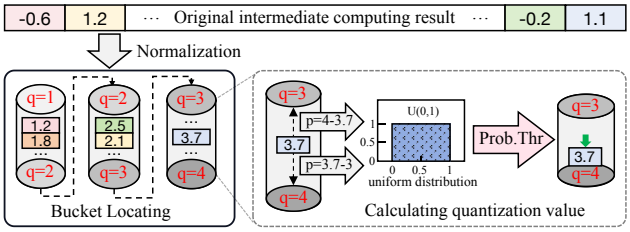


Fig. 4. A general example of Random Quantization module.

Figure 3, we first construct a bit-width intervals by expanding L to a bit-width range with multiple intervals as $\widetilde{L} \in \{1, 2, ..., L, ..., 2L-1\}$, centered at L. After that, MP randomly assigns L_i to the item of $\widetilde{L}$ by the following five steps: ① Calculating a scaler $s = \frac{N_i}{\overline{N}}$ to represent the factor of N_i relative to $\overline{N}$, where $\overline{N}$ is the average data quantity routed to all expert nodes. ② Determining the located bit-width interval $[b, b+1]$ within $\widetilde{L}$ according to s, where $b = 2L - floor(L \times s)$ is the left endpoint of the interval and $b+1$ is the right endpoint. As a result, a negative correlation is built between the data quantity routed to a specific expert node and the corresponding assigned bit-width. ③ Calculating the distance from $2L - L \times s$ to b and $b+1$ as $d_l = (b+1) - (2L - L \times s)$ and $d_{l+1} = (2L - L \times s) - b$, respectively, where $2L - L \times s$ is the location of N_i in the interval $[b, b+1]$. ④ Due to both d_l and d_{l+1} belonging to $[0, 1]$ while summing to 1, we can set the probability of assigning L_i to b and $b+1$ as the d_l and d_{l+1}, respectively. In doing so, L_i will be assigned to the closer endpoint (e.g., b or $b+1$) at a larger probability. ⑤ Assigning L_i to the specific endpoint as the bit-width by comparing the probability with a random number generated from the uniform distribution (i.e., *Probabilistic Thresholding (Prob. THR)* method) at a low computational cost. Notably, it can be derived that the average expectation of bit-width $\mathbb{E}[L_i] = L$ with $\forall N_i \leq 2\overline{N}$. Therefore, although data transferred to different nodes are randomly assigned with different bit-width, the average expectation of bit-width could be maintained at L.

Interval Truncating: In an extremely imbalance situation, data routed to one expert node may be very huge that $\exists N_i > 2\overline{N}$ (e.g., $N_1 = 3, N_2 = 5, N_3 = 1000$), thus the corresponding bit-width will be assigned to 1. While excessively low 1-bit quantization significantly reduces communication costs, it potentially degrades MoE training quality. To balance the communication costs and quantization error, MP bounds the lowest and highest bit-widths by truncating $\widetilde{L}$ as $\{L-\delta, ..., L-1, L, L+1, ..., L+\delta\}$. In doing so, we can flexibly balance communication costs and MoE accuracy while maintaining the average bit-width approximately equal to L. Note that the mixed precision strategy is degraded to the conventional fixed precision quantization if δ is set to 0.

Essentially, the Mixed Precision strategy balances the communication load at the cost of reducing the quantization precision of more data routed to several experts, thus there is a trade-off between δ values and MoE training quality. By evaluating multiple mainstream MoE training tasks, we notice that when δ is set to 2, the all-to-all communication congestion could be effectively eliminated without significantly compromising model training accuracy. After assigning the specific bit-width for the data routed to each expert, BirdMoE utilizes next RQ module to randomly quantize the original floating-point intermediate computing results to integers as follows.

C. Random Quantization (RQ)

As introduced in subsection II-B, lossy compression techniques potentially incur *Overhead Amplification* and *Error Propagation* issues. By exploiting the lightweight advantage of the random process and the convergence efficiency of the expectation invariance, we propose an unbiased Random Quantization (RQ) algorithm to efficiently compress the intermediate computing results of all-to-all communication in MoE layers. As illustrated in Figure 4, the insight of random quantization is to map the floating-point into the closer integer end of the quantization interval with a larger probability.

All-to-all Compression: Given the original floating-point intermediate computing result set $\mathcal{X}$ and the assigned bit-width L, the quantization counterpart of $\mathcal{X}$ is denoted by $\mathcal{Q}$. There is a natural range of quantization level $[1, 2, ..., L]$. A similar methodology as MP is employed to map the i-th elements x_i from $\mathcal{X}$ into $\mathcal{Q}$ as below: ① Calculating the normalized floating points as $f_i = \frac{|x_i|}{||\mathcal{X}||_\infty}$, which is used to reflect the proportion of x_i within the entire range of $\mathcal{X}$; ② Locating the bucket $[l, l+1]$ for x_i according to f_i, where $l = floor(f_i \times L)$ is the left endpoint of the bucket and $l+1$ is the right endpoint. ③ Calculating the distance from f_i to each endpoint as $d_l = f_i \times L - l$ and $d_{l+1} = l + 1 - f_i \times L$, where d_l and d_{l+1} are located in $[0, 1]$ while summing to 1. ④ Establishing a negative correlation between distance and probability by setting the probability of mapping x_i to $l+1$ and l as d_l and d_{l+1}, respectively. In doing so, x_i will be mapped to the closer endpoint (e.g., l or $l+1$) at a larger probability. ⑤ Quantizing x_i to the specific bucket endpoint (e.g., l or $l+1$) as the quantized value q_i using the *Probabilistic Thresholding* method with a low $O(N)$ computational complexity.

Unbiased Recovery: After the above random quantization, the sender node only needs to transfer (*i*) the signed quantized integer q_i, each element of $L+1$-bit; (*ii*) one floating-point factor $||\mathcal{X}||_\infty$, to the target expert node. The receiver node can recover the unbiased and trainable floating-point data with $\mathbb{E}[(\mathcal{Q})] = \mathcal{X}$. The convergence efficiency of similar algorithms has been theoretically proven in gradient compression of data parallelism scenario. Empirically evaluations suggest that RQ

module can effectively mitigate the *Error Propagation* issue in the compression of intermediate computing results using the recommended 3-5 bit-widths, thereby effectively maintaining the MoE training quality (detailed in subsection IV-B).

IV. Experiments

A. Experimental Setup

Tasks, Models, and Datasets: Our experimental workloads cover four typical MoE training tasks, which are trained on various models and datasets at different scales: ② GW2: Training a GPT-based MoE [22] model with 19.2 million parameters on WikiText2 [26] dataset using the Adam optimizer with a learning rate of 1e-2. ③ GW103: Training a larger GPT-based MoE model with 315.1 million parameters on WikiText103 [26] dataset using the Adam optimizer with a learning rate of 1e-2. ④ VC100: Training a ViT-based MoE model [23] with 19.0 million parameters on CIFAR100 [27] dataset using the Adam [28] optimizer with a learning rate of 1e-3. ⑤ VT200: Training a larger ViT-based MoE model with 181.3 million parameters on Tiny ImageNet [29] dataset using the Adam optimizer with a learning rate of 1e-3.

Baseline and compared methods: To demonstrate the superiority of our proposed BirdMoE, we select three popular state-of-the-art quantization-based communication compression techniques: QSGD [14], PWLQ [30], and ShapeShifter [31] to compare the MoE training quality, compression ratio, compression throughput, and training speedup across the above four MoE training tasks. Additionally, we also provide the performance of the original no-compression baseline to refer.

Experimental environment: We implement the MoE training environment upon a distributed cloud-computing platform with 16 GPUs within 4 nodes, and these nodes are connected by a limited 10 Gbps inter-node Ethernet. Within each node, 4 NVIDIA Tesla V100-SXM2-16GB GPUs are connected by a 400 Gbps intra-node NVLink. The compression operations of different methods are implemented using the `Pytorch v.2.2.1` [32] package in Python 3.8.13. The all-to-all communication is built on the collective communication of `torch.distributed` [32] package, utilizing the `NCCL v.2.19.3` backend.

B. Evaluation of MoE Training Quality

The most important premise in communication compression is to ensure that the model training quality remains unaffected significantly. Thus we first explore the superiority of BirdMoE in maintaining MoE training quality (i.e., model accuracy and convergence rate). To study the impact of our proposed mixed precision module on model training quality, we provide the evaluation results of BirdMoE with mixed and fixed precision (referred to as $\text{BirdMoE}_{\text{MP}}$ and $\text{BirdMoE}_{\text{FP}}$). For the sake of fairness, we first compare BirdMoE with the state-of-the-art quantization-based compression approaches QSGD, PWLQ, and ShapeShifter under the same bit-width as BirdMoE.

Figure 5 shows that when we assign different compression methods to the same bit-width, BirdMoE with fixed precision (i.e., $\text{BirdMoE}_{\text{FP}}$) achieves the best MoE training quality, which has a similar performance to the original no-compression MoE training solution. Specifically, ShapeShifter combines a global linear quantization with a lossless encoding to compress the flatten intermediate computing results without considering the distribution difference among data dimension, thereby failing to maintain the convergence stability of MoE training and degrading the model accuracy in GW103 and VT200 tasks. Although QSGD can compress gradients while maintaining the convergence efficiency in the data parallelism by preserving the expectation invariance of L_2-norm, the same measurement is ineffective for quantizing the intermediate results compression, thus causing a significant MoE training divergence. PWLQ employs a piece-wise quantization strategy to reduce the information distortion of intermediate computing results, effectively maintaining the MoE model training quality in GW2 and VC100 tasks. However, PWLQ cannot achieve a stable convergency in the other tasks with the larger model capacity due to its disregard for the distribution differences in data routed to different MoE layer. In contrast, BirdMoE with a Mixed Precision strategy (i.e., $\text{BirdMoE}_{\text{MP}}$) reduces the quantization precision of more data, slightly degrading model accuracy of $\text{BirdMoE}_{\text{FP}}$, while its craft unbiasedness enables it to achieve comparable and stable MoE training quality with further improving all-to-all communication efficiency.

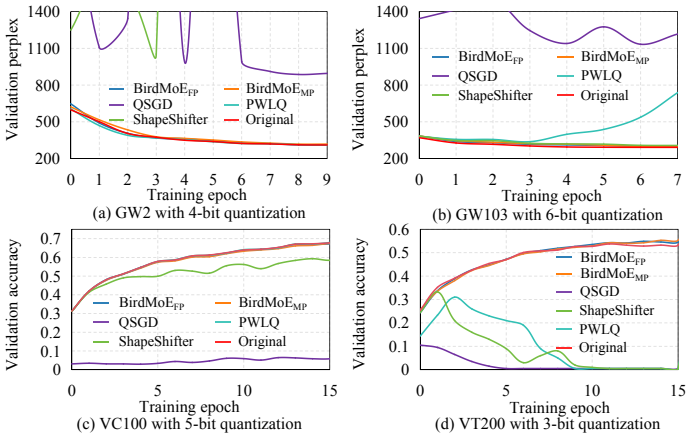


Fig. 5. $\text{BirdMoE}_{\text{FP}}$ and $\text{BirdMoE}_{\text{MP}}$ achieve better MoE training quality than other methods under the same bit-width.

C. Evaluation of Compression Ratio

Figure 6 (a) shows the communication compression ratio of different methods, calculated by the original communication costs of the no-compression solution divided by the compressed communication costs of each method under their **best practice**. Note that the best practice refers to the maximum compression ratio a particular method can achieve while maintaining the similar original MoE accuracy, given the same number of MoE communication rounds as the no-compression solution. Evaluation results indicate that $\text{BirdMoE}_{\text{FP}}$ achieves 5.3×-7.4× compression ratios in vision tasks, outperforming competing methods, while PWLQ and ShapeShifter have a comparable 2.2×-7.2× compression ratio in language tasks. This is because the compressibility of intermediate computing

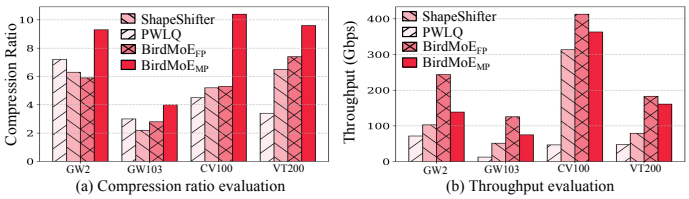


Fig. 6. The evaluation of communication compression ratio and compression throughput (Gbps) across various tasks.

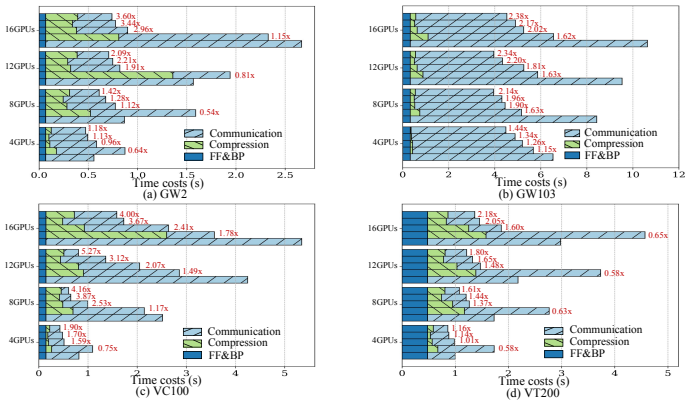


Fig. 7. The average time breakdowns per iteration on 4, 8, 12, and 16 GPU nodes. Bars from top to bottom are time breakdowns of BirdMoE$_{MP}$, BirdMoE$_{FP}$, ShaperShfter, PWLQ, and no-compression.

results differs among the various model structures, leading to different compression performances across tasks. In contrast, BirdMoE$_{MP}$ achieves superior 4.1×-10.4× compression ratios across all tasks by further reducing the bit-width of intermediate computing results with large quantities, thereby significantly increasing the overall compression ratio.

D. Evaluation of Throughput

Figure 6 (b) illustrates the compression throughput of different techniques, which is defined as $tp = \frac{CS}{CT}$, where CS is the amount of reduced data volume by compression, CT is the compression runtime. Compression throughput can effectively reflect the overall performance of an approach by comprehensively considering the compression overheads and gains. Experimental results suggest that although PWLQ and Shapeshifter can achieve a slightly higher compression ratio in partial tasks, their piece-wise calculation and lossless coding incur significant computational overheads, thereby limiting their compression throughput to 12.6-313.5 Gbps. In contrast, BirdMoE$_{FP}$ achieves the highest 125.7-612.8 Gbps compression throughput by employing lightweight random quantization to efficiently achieve a comparable compression ratio. On the other hand, BirdMoE$_{MP}$ utilizes MP module to further reduce the communication costs with an acceptable extra computational cost, achieving the second-best compression 75.1-363.2 Gbps compression throughput.

E. Scalability and Time Breakdown Evaluation

The essential purpose of communication compression is to speed up MoE training at different scales of nodes, thus we further deploy BirdMoE upon the MoE training environment containing various numbers of 4, 8, 12, and 16 GPU nodes. Then we validate our scalability superiority by comparing the average time breakdown and MoE training speedup performance with different compression methods.

Figure 7 illustrates the average time breakdown per iteration of the runtimes of forward and backward propagation (FF&BP), communication, and compression. The red values are acceleration rate (the runtime of the original no-compression MoE training divided by the runtime of MoE training with compression). Experimental results suggest that (*i*) the moderate communication compression ratio and significant compression overheads of PWLQ and Shapeshifter limit their training speedup performance and even prolong the training time to several tasks. Additionally, their inability to address the imbalanced communication load also limits the training speedup performance. (*ii*) BirdMoE$_{FP}$ achieves comparable compression ratio on different tasks at low computational costs, obtaining higher MoE training speedup of 1.13×-3.67×. (*iii*) BirdMoE$_{MP}$ further reduces the communication time by balancing the communication traffic among GPUs, obtaining the highest 1.18×-5.27× MoE training speedup. (*iv*) BirdMoE's training speedup performance improves with the increase of nodes. That is because increasing the number of nodes will increase the compressed communication volume without increasing the compression overhead due to the BirdMoE is deployed on different nodes and executed in parallel. Generally, the above results demonstrate that BirdMoE efficiently reduces communication costs and improves the efficiency of MoE training with different numbers of nodes. Therefore, the good scalability of BirdMoE not only advances research on MoE for resource-constrained platforms but also scales effectively to large-scale MoE training environments.

V. Conclusion and Future Work

In this paper, we propose a lightweight load-aware communication compression technique BirdMoE, which utilizes random quantization and mixed precision to compress the imbalanced and intensive communication of MoE training. Compared with the state-of-the-art approaches, experiments on mainstream MoE training tasks demonstrate that BirdMoE achieves higher 4.06×-10.44× communication compression ratio and 1.18×-5.27× training speedup while maintaining the MoE training quality. In the future, we aim to optimize the efficiency of BirdMoE with the CUDA and implement it into the all-to-all collaborative communication library.

Acknowledgment

This work was supported in part by the Guangdong S&T Program under Grant 2024B0101010002, in part by the National Natural Science Foundation of China (Nos. U2436208, 62372129, 62472127, 62032023, and T2125013), in part by the Project of Guangdong Key Laboratory of Industrial Control System Security under Grant 2024B1212020010, in part by the Innovation Funding of ICT, CAS under Grant No. E461050. (Corresponding author: Zhihong Tian; Wen Xia).

REFERENCES

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, and et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. ICLR'21*, 2021.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT'19*, 2019, pp. 4171–4186.

[3] D. Wu, W. Yang, H. Jin, and et al., "Fedcomp: A federated learning compression framework for resource-constrained edge computing devices," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst*, vol. 43, no. 1, pp. 230–243, 2024.

[4] R. Zhang, M. Javaheripi, Z. Ghodsi, and et al., "Adagl: Adaptive learning for agile distributed training of gigantic gnns," in *proc. DAC'23*, 2023.

[5] D. Wu, X. Zou, S. Zhang, and et al, "Smartidx: Reducing communication cost in federated learning by exploiting the cnns structures," in *Proc. AAAI'22*, 2022, pp. 4254–4262.

[6] D. Wu, W. Yang, X. Zou, and et al., "Smart-dnn+: a memory-efficient neural networks compression framework for the model inference," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 4, pp. 1–24, 2023.

[7] N. Shazeer, A. Mirhoseini, K. Maziarz, and et al., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *Proc. ICLR 2017*, 2017.

[8] J. He, J. Zhai, T. Antunes, and et al, "Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models," in *Proc. PPoPP '22*, 2022.

[9] J. Li, Y. Jiang, Y. Zhu, and et al., "Accelerating distributed moe training and inference with lina," in *Proc. USENIX ATC 2023*, 2023.

[10] M. Zhai, J. He, Z. Ma, and et al., "Smartmoe: Efficiently training sparsely-activated models through combining offline and online parallelization," in *Proc. USENIX ATC 2023*, 2023.

[11] D. Lepikhin, H. Lee, Y. Xu, and et al., "Gshard: Scaling giant models with conditional computation and automatic sharding," in *Proc. ICLR 2021*, 2021.

[12] F. Xue, Z. Shi, F. Wei, and et al., "Go wider instead of deeper," in *proc. AAAI'22*, 2022, pp. 8779–8787.

[13] D. Wu, W. Yang, X. Zou, and et al., "Bird+: Design of a lightweight communication compressor for resource-constrained distribution learning platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 11, pp. 2193–2207, 2024.

[14] D. Alistarh, D. Grubic, J. Li, and et al., "QSGD: communication-efficient SGD via gradient quantization and encoding," in *Proc. NIPS'17*, 2017.

[15] D. Wu, W. Yang, C. Deng, and et al., "Bird: A lightweight and adaptive compressor for communication-efficient distributed learning using tensor-wise bi-random sampling," in *proc. ICCD'23*. IEEE, 2023.

[16] Y. Lin, S. Han, H. Mao, and et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. ICLR'18*, 2018.

[17] T. Vogels, S. P. Karimireddy, and M. Jaggi, "Powersgd: Practical low-rank gradient compression for distributed optimization," in *Proc. NIPS'19*, 2019, pp. 14 236–14 245.

[18] Y. He, X. Huang, and K. Yuan, "Unbiased compression saves communication in distributed optimization: When and how much?" in *Proc. NIPS'23*, 2023.

[19] L. Condat, K. Yi, and P. Richtárik, "EF-BV: A unified theory of error feedback and variance reduction mechanisms for biased and unbiased compression in distributed optimization," in *Proc. NIPS'22*, 2022.

[20] H. Xu, K. Kostopoulou, A. Dutta, and et al., "Deepreduce: A sparse-tensor communication framework for federated deep learning," *in Proc. NIPS'21*, pp. 21 150–21 163, 2021.

[21] A. Vaswani, N. Shazeer, N. Parmar, and et al., "Attention is all you need," in *Proc. NIPS'17*, 2017, pp. 5998–6008.

[22] T. B. Brown, B. Mann, N. Ryder, and et al., "Language models are few-shot learners," in *Proc. NIPS'20*, 2020.

[23] C. Riquelme, J. Puigcerver, B. Mustafa, and et al., "Scaling vision with sparse mixture of experts," in *Proc. NIPS'21*, 2021.

[24] S. Shi, X. Zhou, S. Song, and et al., "Towards scalable distributed training of deep learning on public cloud clusters," in *Proc. MLSys'21*, 2021.

[25] A. Mandal, H. Jiang, A. Shrivastava, and V. Sarkar, "Topkapi: Parallel and fast sketches for finding top-k frequent elements," in *Proc. NIPS'18*, 2018, pp. 10 921–10 931.

[26] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *Proc. ICLR'17*, 2017.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR'15*, 2015.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NIPS'12*, pp. 1106–1114.

[30] J. Fang, A. Shafiee, H. Abdel-Aziz, and et al., "Post-training piecewise linear quantization for deep neural networks," in *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 12347. Springer, 2020, pp. 69–86.

[31] A. D. Lascorz, S. Sharify, I. E. Vivancos, and et al., "Shapeshifter: Enabling fine-grain data width adaptation in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 2019, pp. 28–41.

[32] A. Paszke, S. Gross, F. Massa, and et al., "Pytorch: An imperative style, high-performance deep learning library," in *Proc. NIPS'19*, 2019.

ACRS: Adjacent Computation Resource Sharing among Partitioned GPU Sub-Cores

Penghao Song[1,2], Chongxi Wang[1,2], Chenji Han[1,2], Haoyu Zhao[3], Tingting Zhang[3], Tianyi Liu[4,*], Jian Wang[1,2,*]
[1]State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]Loongson Technology Co. Ltd, Beijing, China
[4]University of Texas at San Antonio, United States
{songpenghao16,wangzhongxi15,hanchenji16}@mails.ucas.ac.cn,
{zhaohaoyu,zhangtingting}@loongson.cn, tianyi.liu@utsa.edu, jw@ict.ac.cn

***Abstract*—Modern GPUs typically segment Streaming Multiprocessors (SMs) into sub-cores (e.g. 4 sub-cores) to reduce power consumption and chip area. However, this partitioned design prevents potential task distributions across sub-cores, impairing overall execution efficiency. In this paper, we explore the performance benefit of sharing hardware resources among sub-cores and identify functional units (FUs) as critical components for compute-intensive applications. Moreover, our observations reveal that instructions residing in operand collectors can be obstructed by back-end FUs, but there is a high probability that unoccupied FUs are available in adjacent sub-cores during such blockages.**

In response, we introduce the adjacent computation resource sharing (ACRS) framework to efficiently utilize these unoccupied units among sub-cores. ACRS has two key modules: Shared FU Issue (SF_ISSUE) and Shared FU Write Back (SF_WriteBack). SF_ISSUE monitors the status of operand collectors and functional units, and offloads instructions from blocked sub-cores to unoccupied resources. Meanwhile, SF_WriteBack routes results back to the original sub-core.To minimize wiring overhead, each sub-core is assigned a fixed target core for sharing. We design a series of matching policies and finally filter out the most effective sequential method. Evaluation results show that ACRS improves performance by up to 46.4%, with an average of 14.1% over the traditional partitioned architecture, while reducing energy consumption by 8.3%. Besides, ACRS achieves an additional 12.3% performance improvement compared with the SOTA method.

I. Introduction

Modern GPUs have grown in complexity to meet the expansion of high-concurrency applications such as artificial intelligence and high-performance computing. To address the increases in power consumption and die size, GPU architectures now typically divide the main processing unit into homogeneous sub-cores [2]–[10], each equipped with its own control logic and computation resources.

However, partitioned architecture can significantly affect execution efficiency at runtime. Barnes et al. [11] revealed that partitioned architectures in streaming multiprocessors (SMs) cause performance drops when computing tasks are distributed among sub-cores with unbalanced workloads, as well as the increased opportunity of register bank conflicts. Our preliminary study further shows that there exists a large optimization space when integrating functional units (FUs), with an ideal improvement of 28.1%. Based on these works, we aim to explore a practical FU sharing strategy among sub-cores.

According to our observations, it is greatly effective to share FUs during the blockage of operand collectors (OCs). First, OCs often block the pipeline when functional units are unavailable. This is especially true for FUs with limited execution lanes and high occupancy rates. Second, there is a high probability of finding unoccupied FUs in neighboring sub-cores during these blockages. Our analysis shows that in 68.4% of blockage cases, at least one sub-core in the same SM has available FUs of the same type, indicating a significant potential for resource sharing.

However, the resource-sharing approach encounters challenges, particularly with global sharing within an SM. The challenge arises because resource isolation aims to reduce area and energy consumption. Introducing a crossbar-like network for data exchange among the sub-cores could lead to high overhead. Furthermore, integrating additional write-back paths into the register files requires a significant revision of the write-back logic. This integration could increase pipeline blockages, as data from multiple sub-cores would compete for the same write port. Additionally, adding more write ports would further complicate the issue by expanding the register files' size.

To address these challenges, we present adjacent computation resource sharing **(ACRS)** framework to enhance the utilization of separated functional units and effectively accelerate program execution. This design comprises two hardware components and one pairing strategy: an **SF_ISSUE** module to reroute blocked OC instructions to unoccupied FUs; an **SF_WriteBack** module at the FU exit that returns results to the correct register files via the sub-core identifier; and a **Matching Policy** to determine the sharing target sub-core. Specifically, to simplify control and transmission logic, we have shifted from a global sharing model to a more efficient one-to-one sub-core sharing scheme, significantly reducing the crossbar overhead. Moreover, we have implemented a four-entry write buffer at the register file's entrance to temporarily store write data from other sub-cores, thus avoiding the need for extra write ports and mitigating direct pipeline blockages.

The contributions of this paper include,

*Corresponding author.

- A performance analysis of the impact of sharing different resource types and demonstrate that integrating functional units provides the greatest performance boost.
- An ACRS framework that enables adjacent sub-cores to share unoccupied FUs, allocating available resources to busy operand collectors, thus improving load balancing.
- A comprehensive evaluation, which achieves performance improvement of up to 46. 4%, with an average of 14.1% over the traditional partitioned architectures, while reducing energy consumption by 8.3%.

II. Background

A. Partitioned Architecture of the SM

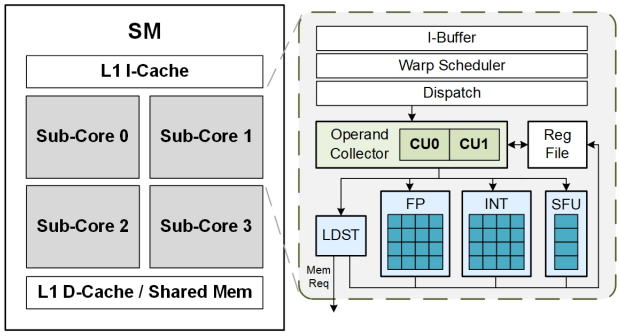


Fig. 1: Baseline SM architecture.

In contemporary GPU architectures, a hierarchical and homogeneous design paradigm is commonly adopted to streamline scheduling and control logic. The design principle is evident at the foundational hardware level, where each basic hardware unit is uniformly divided into four identical segments. For instance, NVIDIA divides a single Streaming Multiprocessor into four sub-cores [2]–[6], while AMD structures each Compute Unit with four SIMD units [7]–[10]. As illustrated in Figure 1, each quadrant of a divided SM possesses its own control and computation resources yet shares a unified L1 data cache and shared memory. Threads are organized into warps that operate synchronously. Once dispatched to a specific sub-core, a warp becomes one of several candidates, with its instructions stored in the I-buffer (instruction buffer) after the fetch and decode stages. Subsequently, an internal warp scheduler orchestrates the distribution of instructions to the operand collector, where they await the retrieval of operands from the register file. Upon readiness of the operands, the instruction is transmitted to either the functional units or the load/store units for further execution.

B. Blocking issues in FUs and OCs

Within each sub-core, multiple functional units process different data types and tasks, including floating point units (FP) for single-precision data, integer units (INT) for integer operations, and special function units (SFU) for complex functions like trigonometry. However, due to hardware limits, a sub-core often cannot process an entire warp simultaneously. For example, each sub-core in the RTX 2060 features 16 FPs and 4 SFUs, yet a warp consists of 32 threads [3]. Consequently, FP instructions require 2 consecutive cycles to be issued to the function units, while SFU instructions take at least 8 cycles.

TABLE I: Baseline simulation parameters.

SM Config	34 SMs, 1600 MHz
Per SM	4 sub-cores, Round-Robin sub-core scheduling
Per sub-core	8 warps, 2 CUs, Greedy-Then-Oldest warp scheduling, 64KB register files, 16 INTs, 16 FPs, 4 SFUs
Cache	128KB L1 I-Cache, 64KB L1 D-Cache/shared memory, 64KB Constant Cache, 4MB L2 Cache

TABLE II: Evaluated applications. The labels **C** and **M** in the Type column stand for compute-intensive and memory-intensive cases, respectively.

Suite	Abbr.	Type	Name
Rodinia [13]	LMD	C	Lava Molecular Dynamics
	KMS	M	K-means Clustering
Dragon-li [1]	JON	C	Dynamic Parallelism Join
Ispass [27]	RAY	C	Ray Tracing
Proxy [15], [16], [28]	XSB	C	XSBench
	RSB	C	RSBench
	LLS	M	Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics
Parboil [14]	SPM	M	Sparse-Matrix Dense-Vector Product

This behavior, where a single instruction occupies multiple pipeline stages, can ultimately cause instruction blocking.

The core structure of the sub-core data flow is the operand collector [17], [18], shared by all functional units. The operand collector has multiple collector units (CUs) assigned by the dispatch logic. These units temporarily store instructions fetching operands and are designed to mitigate register file bank conflicts by processing multiple warp instructions concurrently. However, execution around the OCs can stall due to various issues. If no CU is available, the dispatch phase stalls. Additionally, if an instruction occupies multiple execution pipeline stages, especially SFU instructions, subsequent instructions requiring the same unit may stall at the OC exit, delaying execution and reducing efficiency.

III. Experimental Environment

We employ GPGPU-Sim 4.2.0 [12] to simulate a Turing-like architecture, modified from the RTX2060-S configuration provided. Table I presents the baseline parameters. We utilize four sub-cores per SM, mirroring the common 4-core architecture of mainstream GPUs. The methods described in this paper are applicable to architectures with more sub-cores. Additionally, we incorporate two collector units in each sub-core, as this configuration most accurately reflects the actual architecture [11]. This base architecture adeptly models the subdivision of a single SM into sub-cores, facilitating a detailed analysis of the enhancements proposed in this study. Table II presents the representative compute-intensive and memory-intensive applications we select from multiple benchmark suites, including Rodnia [13], Dragon-li [1], Ispass [27], Proxy-apps-doe [15], [16], [28], and Parboil [14]. The selection is based on the applications' characteristics and insights from prior works [21], [29],

trying to ensure a balanced representation of both categories. Previous research [19] indicates that the experimental results tend to converge after approximately 2 billion instructions are executed. We use the largest input sizes for each application and aim to execute each application either until completion or for a maximum of 4 billion instructions.

IV. Analysis

A. Impact of Multiple Partitioned Resources

Each sub-core exclusively possesses four key types of resources in the partitioned architecture: warp scheduler, register files, operand collector, and functional units. Unlike fully connected architectures, these monopolized resources are inaccessible to other sub-cores, even when they are underutilized or left idle. To evaluate the performance impact of individual resources, we share each type of resource within the SM modified from a partitioned structure and then measure the improvements in instructions per cycle (IPC) using the geometric mean, as shown in Figure 2. The following observations are made:

Observation 1: Among the four shared resources, the sharing of functional units and operand collectors both significantly boost performance, with functional units showing the most notable benefit in compute-intensive cases, resulting in an average IPC improvement of 28.1%.

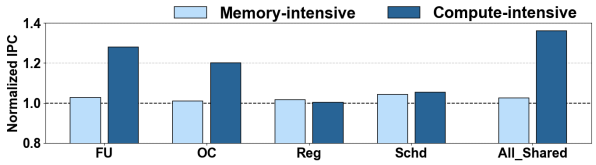


Fig. 2: IPC improvement from shared resources among sub-cores: warp schedulers(Schd), register files(Reg), operand collectors(OC), functional units(FU), and fully connected cores (All_Shared). The results are normalized to the 4-core baseline.

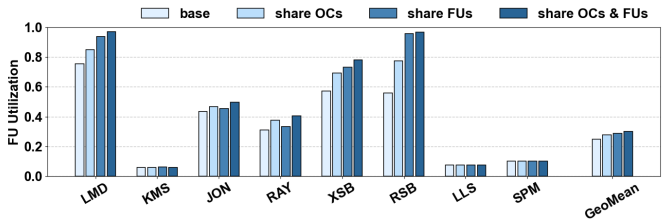


Fig. 3: Comparison of FU utilization for shared structures: shared operand collectors, shared functional units, and their combination, against the 4-core baseline.

Due to the close relationship between operand collectors and functional units, we further investigate whether sharing these two components offers similar benefits. We simultaneously share both operand collectors and functional units, demonstrating their impact on FU utilization as illustrated in Figure 3. FU utilization is defined as the proportion of cycles in which FU is active out of the total number of running cycles. From this figure, we make two main observations:

Observation 2: In memory-intensive cases (KMS, LLS, SPM), FU utilization remains below 20%, and sharing OCs and FUs offers virtually no benefit. This inefficiency stems from the substantial cache misses in these scenarios, which cause prolonged memory access delays and greatly impede the processing of subsequent instructions.

Observation 3: Simultaneously sharing OCs and FUs does not show a significant difference compared to integrating them separately. However, sharing FUs still outperforms sharing OCs. This is especially true in the cases of LMD, XSB, and RSB, where utilization is the highest among all applications.

Insights: The integration of functional units and operand collectors both significantly influences execution efficiency. Compared to sharing both, focusing solely on FUs achieves a similar impact. Additionally, it avoids the extra overhead associated with sharing OCs.

B. The Sharing Opportunity of Blocking Events

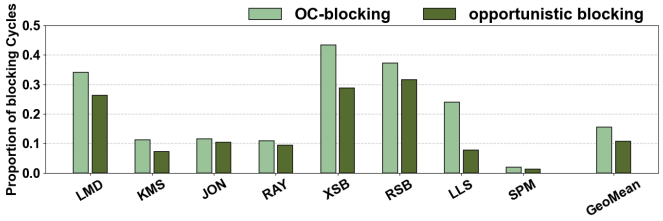


Fig. 4: The proportion of cycles for OC-blocking and opportunistic blocking relative to the total execution cycles.

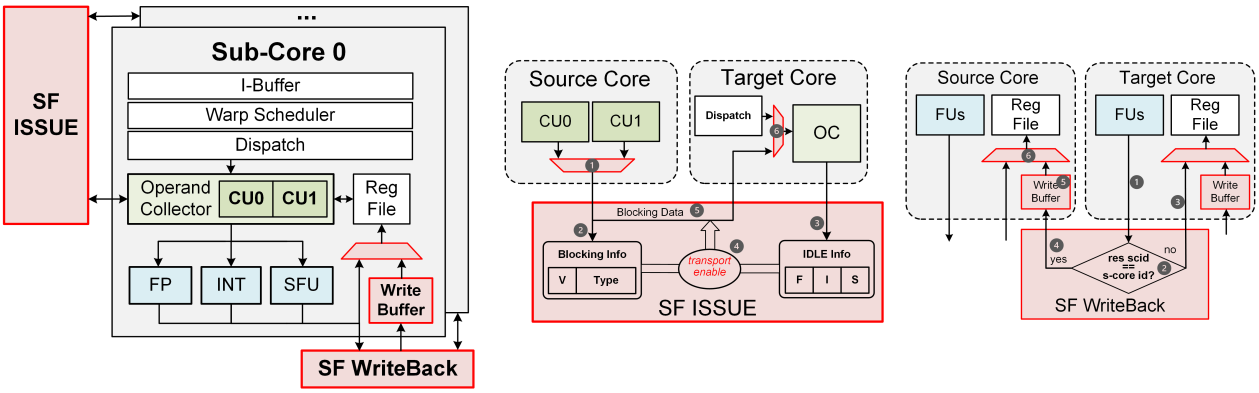
Pipeline blocking, as described in Section II-B, poses a significant issue in the partitioned architecture. We refer to situations where at least one CU in the OC is blocked as *OC-blocking*. OC-blocking not only diminishes the execution efficiency of the current instruction but also consumes OC resources, potentially hindering the dispatch of subsequent instructions and posing a broader threat to system performance. Figure 4 classifies OC-blocking into two categories: opportunistic and non-opportunistic, and provides statistics on their duration throughout the execution process. *Opportunistic blocking* refers to situations where other sub-cores have similar-type functional units available when the current sub-core experiences OC-blocking. To further understand the sharing potential during OC-blocking, we calculate the conditional probability of opportunistic blocking based on the following equation:

$$P(\text{opportunistic} \mid \text{OC-blocking}) = \frac{F(\text{opportunistic})}{F(\text{OC-blocking})} \quad (1)$$

where $F(*)$ represents the proportion of cycles observed in the total number of execution cycles. We make two key observations from the figure:

Observation 4: OC-blocking is quite prevalent, especially in compute-intensive cases. On average, the execution flow in the programs tested is disrupted by OC-blocking 15.5% of the time, with some tests (LMD, XSB, RSB) experiencing rates as high as approximately 40%.

Observation 5: In all applications tested, the average conditional probability of opportunistic blocking is 68.4%, reaching a peak of 90.5% in the RSB case. This suggests a high probability of reducing the frequency of OC-blocking by utilizing unoccupied units from other sub-cores.



(a) Overall structure (b) SF_ISSUE module (c) SF_WriteBack module

Fig. 5: The proposed ACRS framework.

Insights: OC-blocking is a common issue that impacts execution efficiency. However, there is substantial potential to mitigate these blockages by sharing computation resources. Specifically, if a sub-core can identify and utilize unoccupied execution units from other cores during OC-blocking episodes, overall performance can be greatly enhanced.

V. Design

A. ACRS Architecture

We propose the implementation of ACRS (adjacent computation resource sharing), which is designed to share computation resources among specific sub-core combinations within the SM, alleviating pipeline blocking and enhancing execution efficiency. Figure 5(a) illustrates the overall framework of ACRS where the red parts indicate modifications from the baseline architecture. To facilitate resource sharing, multiple additional data flows are introduced among the four sub-cores. The structure is divided into two main components: SF_ISSUE (Shared-FU Issue) and SF_WriteBack (Shared-FU Write Back), which respectively handle cross-sub-core data exchange before and after task execution. Additionally, a small write buffer is added at the write entrance to the register files to temporarily store results from other sub-cores.

Considering routing overhead, not all resources are made accessible to each core. For a specific sub-core, known as the Source Core (S-Core), when a blocking instruction that can be shared is encountered, the designated Target Core (T-Core) evaluates its ability to accept and process this instruction. The sharing relationship is fixed as one-to-one among the four cores to prevent the formation of an expensive crossbar. We explore various matching policies between S-Cores and T-Cores, which clearly affect the ultimate effectiveness of sharing.

SF_ISSUE. Figure 5(b) depicts the issue logic for a set of S-Core and T-Core. When S-Core experiences OC-blocking, a blocking instruction is first selected from two CUs using the loose round-robin strategy (❶), and its details are then sent to the SF_ISSUE module for arbitration. The transferred information consists of two components: the Blocking Info (❷) and the Blocking Data (❺). Specifically, the Blocking Info identifies the type and validity of each instruction. Instruction types are classified into F(FP), I(INT), and S(SFU) based on the back-end functional units. At the T-Core side, the operand collector consistently monitors the idle status of functional units and sends the information to the SF_ISSUE module. The IDLE Info (❸) consists of 3 bits corresponding to the three different function units. When no CU is idle, all three bits are 0. If there is an idle CU and the corresponding FU is also available, the respective bit is set to 1. Thus, the high status of IDLE Info bits indicates that the corresponding type of instruction can be promptly sent to the T-Core without further delay.

Once in possession of the Blocking Info from the S-Core and the IDLE Info from the T-Core, SF_ISSUE is able to employ comparison logic (❹) to determine their compatibility. The comparison leverages the type from the Blocking Info to examine the three bits of the IDLE Info. If the corresponding bit is set to 1 and the instruction is valid, the Blocking Data will be issued to the T-Core for execution. Data from the S-Core is given higher priority (❻) over data dispatched inside the T-Core and is directly accepted by the operand collector. The design rationale is that the transport enable signal is set high only when the execution flow of the S-Core is busy, while the execution flow of the T-Core is comparatively idle, allowing the T-Core to defer processing its own requests.

SF_WriteBack. When the S-Core instruction completes execution in the T-Core, the computation outputs need to be sent back via SF_WriteBack. As illustrated in Figure 5(c), the results (❶) from the functional units are directly forwarded to SF_WriteBack. The returned data includes the core ID, which identifies the type of the issuing core. If SF_WriteBack determines the T-Core as the issuing core (❸), it confirms local execution and allows direct write-back. Conversely, if the issuing core is the S-Core (❹), the results must be routed to the corresponding S-Core. By accurately routing the computation data, the integrity of the SF results is maintained.

The design augments the register write path with a route enabling data return from other cores. To avoid the substantial overhead associated with adding additional ports to the register file, SF_WriteBack incorporates a small WriteBuffer at the entrance to the register file (❺) to temporarily store computation results from other cores. The local write-back path and the WriteBuffer share a common write port. During arbitration (❻), the data with the earliest dispatch time is selected for

return to the register file. According to our experiments, a write buffer with a capacity of four entries is sufficient for collecting externally returned data.

Hardware Overheads. Compared to a crossbar used for global sharing, the structure proposed in this paper achieves a significant reduction in layout and routing overhead. The area complexity of a crossbar scales as $O((pw)^2)$ [42], while its scheduler incurs an additional overhead of $O(p^4)$ [43], where p denotes the number of ports and w represents the width of each port. By employing a fixed matching policy, the proposed approach reduces the overhead to $O(pw)$, effectively avoiding the exponential growth in area complexity. Apart from the wiring logic, the primary hardware overhead is the write buffer at the register file interface. Each buffer consists of four entries, with each entry capable of storing a 32-bit result for one warp, totaling 128 bytes. For 64-bit operations, each instruction occupies two buffer entries Consequently, the total size of the write buffer for each sub-core is 512 bytes, which is negligible compared to the 64KB register files.

B. Matching Policies

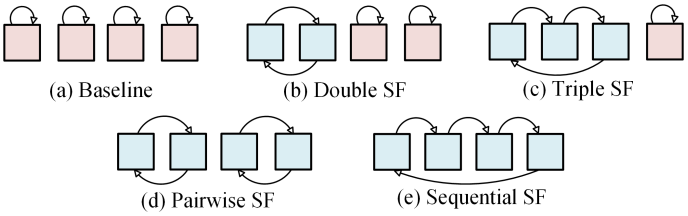


Fig. 6: Matching policies among four sub-cores. Each connection line represents a sharing relationship (from S-Core to T-Core). Blue squares are sub-cores participating in sharing, while red squares are not. We focus on the latter four policies.

Based on the structure proposed in section V-A, each sub-core is required to serve as both an S-Core and a T-Core. With four sub-cores in the SM, four sets of S-Cores and T-Cores are needed. Treating all four sub-cores as identical, we enumerate all possible matching policies for the shared functional units (SF) architecture based on topological principles, as illustrated in Figure 6. In the diagram, each square symbolizes a sub-core, and the connecting lines demonstrate the pairings from S-Cores to T-Cores. Figure 6(a) illustrates the baseline scenario with no sharing, where each core serves as its own S-Core and T-Core. Figure 6(b) shows the *Double SF* configuration, in which two cores share mutually, each acting as the T-Core for the other, while the other two cores remain unshared. Figure 6(c) depicts *Triple SF*, which divides the sub-cores into groups of three and one, with the trio employing a sequential sharing method and the single remaining core not participating in sharing. Figure 6(d), illustrating *Pairwise SF*, divides the four sub-cores into two pairs, with each pair of sub-cores serving reciprocally as T-Cores. Finally, as shown in Figure 6(e), *Sequential SF* policy organizes the sub-cores into a ring, where each subsequent sub-core serves as the T-Core for the previous one.

Notably, we treat all four cores as equivalent, therefore the division of match groups depends only on the number of cores, not on specific cores. For instance, given core IDs from 0 to 3 with parentheses indicating the division into sharing groups, the matching policy for the group (0 1) and (2 3) is considered the same as for the group (0 2) and (1 3), both classified as Pairwise SF. Consequently, the above four matching policies primarily differ in their scope of sharing. Within each group, cores are sequentially assigned a unique T-Core. Compared to the FU sharing method discussed in Section IV-A, referred to as *Ideal SF*, these one-to-one and fixed sharing relationships significantly reduce hardware layout and routing pressure.

VI. Evaluation

In this section, we evaluate the proposed optimization methods using benchmarks from Table II and simulator configurations from Table I as our baseline.

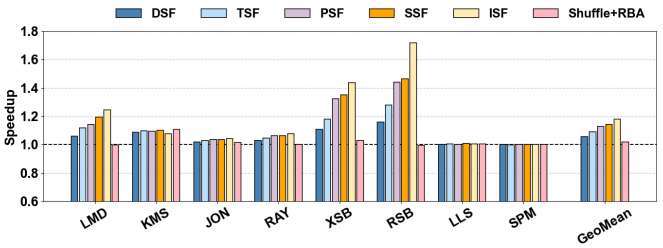


Fig. 7: IPC improvement over the baseline of Shuffle+RBA (SOTA) and our ACRS method with four candidate matching policies: DSF (Double SF), TSF (Triple SF), PSF (Pairwise SF), and SSF (Sequential SF), and the ideal scenario: ISF (Ideal SF).

Performance. Figure 7 illustrates the IPC improvement across all applications under the matching policies described in Section V-B. First, all four matching policies exhibit noticeable improvements in IPC, with larger scopes of sharing resulting in greater enhancements. The sequential matching policy (SSF) shows the largest increase, achieving an average IPC improvement of 14.1%, closely approaching the results of the ISF policy. Second, when all four sub-cores participate in sharing, significant performance improvements are observed across all sharing strategies, with minimal differences among them. The PSF strategy, for instance, achieves an average speedup only 1.3% lower than the best-performing SSF strategy. Third, compute-intensive applications benefit markedly from the shared structure, with the SSF strategy exhibiting up to 46.4% and an average improvement of 19.1%. As for memory-intensive applications, although the enhancements are modest, there are no adverse effects.

Compared with the Shuffle sub-core scheduler and RBA warp scheduler [11] that has a modest 1.8% average performance improvement, our ACRS with SSF matching policy achieves an additional 12.3% improvement, which demonstrates the effectiveness of our approach. The main reason lies in that Shuffle+RBA only focuses on front-end optimization, leaving OC-to-FU stalls unaddressed. The Shuffle scheduler mitigates inter-warp divergence by employing randomized sub-core scheduling patterns, while the RBA scheduler reduces register bank conflicts by sharing queue information of register file banks with the warp scheduler. However, both strategies are unable to resolve the FU separation challenges. In contrast, the ACRS architecture alleviates OC-blocking through FU sharing,

significantly improving pipeline efficiency by addressing back-end bottlenecks.

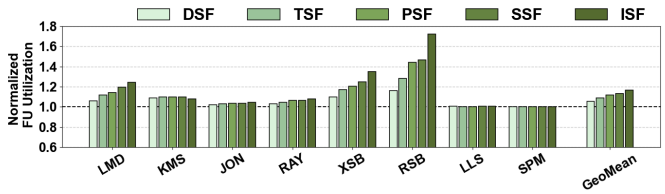


Fig. 8: FU utilization normalized to the baseline.

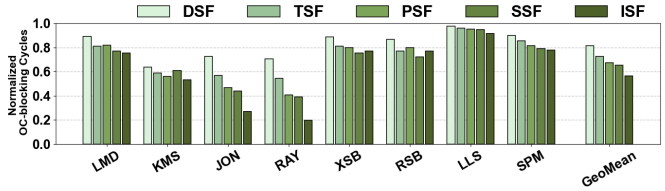


Fig. 9: OC-blocking cycles normalized to the baseline.

We further explore the root causes for the performance improvement and confirm it mainly comes from increased FU utilization and reduction of OC-blocking. Comparing Figure 7, 8, and 9 together, tests showing notable performance increases also exhibit a significant rise in FU utilization and a marked decrease in OC-blocking time. For instance, in the most effective RSB case, the SSF strategy increases FU utilization by 46.5% and reduces the OC-blocking cycles by 27.7%. This observation consistently holds for the top three effective compute-intensive cases (LMD, XSB, RSN). Interestingly, in cases like JON and RAY, although there is a notable reduction in OC-blocking time, the performance improvements are not substantial. This is because OC-blocking is relatively infrequent in these tests and does not represent a bottleneck in the compute pipeline. Therefore, reducing the frequency of OC-blocking has minimal impact on performance in these cases. As for memory-intensive cases (LLS, SPM), their performance is limited by long-latency memory accesses, with memory stalls dominating execution. Consequently, FU utilization remains largely unchanged, and while OC-blocking may be alleviated to some extent, it does not substantially affect overall efficiency due to its minor role in these cases.

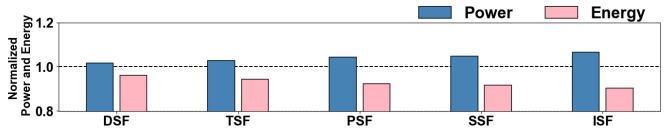


Fig. 10: Normalized power and energy consumption of different matching policies compared with partitioned baseline.

Power. We conduct a power consumption analysis using the AccelWattch platform [20]. The results are normalized and depicted in Figure 10. Compared to the partitioned baseline, all matching policies exhibit an increase in power consumption because they complete the same workload in a shorter period. However, higher power consumption does not necessarily lead to a rise in heat generation. The acceleration of program execution reduces circuit operating time, which in turn decreases static energy consumption and heat produced by control logic circuits. Notably, under the highly effective SSF strategy, while power consumption increases by 4.7%, energy consumption experiences a more significant reduction, decreasing by 8.3%.

VII. Related Work

Resource Sharing. Various internal GPU sharing strategies at different granularities have been proposed to enhance resource utilization and improve overall performance. [21], [22] focused on optimizing the memory subsystem by enabling the private L1D cache of one SM to be accessible by other SMs. This approach not only reduced data redundancy but also significantly increased memory access efficiency, addressing the bottlenecks commonly associated with memory-bound workloads. [23] introduced BigSM, which meets the computational demands of different kernel types by sharing all resources among adjacent SMs, demonstrating the potential of resource pooling in achieving greater flexibility and scalability in GPU. Tao Zhang et al. [25] shared the pipeline front-end across different SMs, effectively reducing power consumption. Other research [24] targeted sharing at the warp level within an SM by employing a combination of hardware and software techniques to enable register file sharing.

FU Utilization. In addition to sharing methods, previous research has explored different approaches to improve the FU utilization in GPUs. [30]–[34] concentrated on mitigating the impact of branch divergence. By increasing the number of active threads per warp, these approaches effectively reduced the number of idle compute units, ensuring a more balanced workload distribution and enhancing overall throughput in branch-heavy applications. [35]–[37] proposed modifications to warp scheduling strategies aimed at maximizing thread-level parallelism. By carefully prioritizing warps, their method ensured a steadier supply of instructions to the FUs. Meanwhile, [38]–[41] introduced out-of-order execution structures into the SM, utilizing instruction-level parallelism to further reduce pipeline stalls before the execution stage.

Compared to prior works, our approach is the first to explore sharing at the granularity of different SM sub-cores and to primarily focus on functional units as the central sharing objects. Additionally, our approach can complement previous sharing strategies, further improving overall utilization.

VIII. Conclusion

In partitioned SM architectures, the separation of sub-core resources can significantly impact execution efficiency. This paper quantifies the performance impact of resource integration, identifying functional units as critical in compute-intensive applications. To address this issue, we propose ACRS, an architecture enabling functional unit sharing between adjacent sub-cores when operand collectors are blocked. We also introduce several sub-core matching policies, with experimental results showing that the sequential policy yields the greatest performance gains.

Acknowledgment

We appreciate the constructive comments and suggestions from all the reviewers. This work is supported by the ICT Innovation Grant E461100.

REFERENCES

[1] "dragon_li." [Online]. Available: https://github.com/gtcasl/dragon_li

[2] NVIDIA, "NVIDIA GeForce GTX 750 Ti," 2014. [Online]. Available: https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf

[3] NVIDIA, "NVIDIA Turing GPU Architecture," 2018. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[4] NVIDIA, "NVIDIA Ampere GA102 GPU Architecture," 2021. [Online]. Available: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf

[5] NVIDIA, "NVIDIA Ada GPU Architecture," 2023. [Online]. Available: https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf

[6] J. Choquette, O. Giroux, and D. Foley, "Volta: Performance and Programmability," IEEE Micro, vol. 38, no. 2, pp. 42–52, Mar. 2018.

[7] AMD, "Introducing RDNA Architecture," 2019. [Online]. Available: https://www.amd.com/system/files/documents/rdna-whitepaper.pdf

[8] AMD, "Introducing AMD CDNA Architecture," 2020. [Online]. Available: https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf

[9] AMD, "AMD CDNA 2 Architecture," 2021. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf

[10] AMD, "AMD CDNA 3 Architecture," 2023. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf

[11] A. Barnes, F. Shen, and T. G. Rogers, "Mitigating GPU Core Partitioning Performance Effects," in 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Montreal, QC, Canada: IEEE, Feb. 2023, pp. 530–542.

[12] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain: IEEE, May 2020, pp. 473–486.

[13] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in 2009 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA: IEEE, Oct. 2009, pp. 44–54.

[14] J. A. Stratton et al., "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," 2012.

[15] J. R. Tramm, A. R. Siegel, T. Z. Islam, and M. Schulz, "Xsbench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis," 2014.

[16] J. R. Tramm, A. R. Siegel, B. Forget, and C. Josey, "Performance Analysis of a Reduced Data Movement Algorithm for Neutron Cross Section Data in Monte Carlo Simulations," in EASC 2014 - Solving Software Challenges for Exascale, Stockholm, 2014.

[17] S. Liu, J. E. Lindholm, M. Y. Siu, B. W. Coon, and S. F. Oberman, "Operand collector architecture," 7834881, Nov. 16, 2010

[18] J. H. Choquette, M. O. Gautho, and J. E. Lindholm, "Methods and apparatus for source operand collector caching," 20130159628, Jun. 20, 2013

[19] Z. Lin, M. Mantor, and H. Zhou, "GPU Performance vs. Thread-Level Parallelism: Scalability Analysis and a Novel Way to Improve TLP," ACM Trans. Archit. Code Optim., vol. 15, no. 1, Mar. 2018.

[20] V. Kandiah et al., "AccelWattch: A Power Modeling Framework for Modern GPUs," in MICRO-54, : 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event Greece: ACM, Oct. 2021, pp. 738–753.

[21] H. Falahati et al., "Cross-Core Data Sharing for Energy-Efficient GPUs," ACM Trans. Archit. Code Optim., p. 3653019, Mar. 2024.

[22] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and Leveraging Decoupled L1 Caches in GPUs," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Korea (South): IEEE, Feb. 2021, pp. 467–478.

[23] A. Dhar and D. Chen, "Efficient GPGPU Computing with Cross-Core Resource Sharing and Core Reconfiguration," in 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA: IEEE, Apr. 2017, pp. 48–55.

[24] F. Khorasani, H. Asghari Esfeden, A. Farmahini-Farahani, N. Jayasena, and V. Sarkar, "RegMutex: Inter-Warp GPU Register Time-Sharing," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA: IEEE, Jun. 2018, pp. 816–828.

[25] T. Zhang and X. Liang, "Dynamic front-end sharing in graphics processing units," in 2014 IEEE 32nd International Conference on Computer Design (ICCD), Seoul, South Korea: IEEE, Oct. 2014, pp. 286–291.

[26] A. Jog et al., "Exploiting Core Criticality for Enhanced GPU Performance," in Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science, Antibes Juan-les-Pins France: ACM, Jun. 2016, pp. 351–363.

[27] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, USA: IEEE, Apr. 2009, pp. 163–174.

[28] I. Karlin, "LULESH Programming Model and Performance Ports Overview," LLNL-TR-608824, 1059462, Dec. 2012.

[29] A. Sethia, D. A. Jamshidi, and S. Mahlke, "Mascar: Speeding up GPU warps by reducing memory pitstops," in 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, USA: IEEE, Feb. 2015, pp. 174–185.

[30] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow," in Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, in MICRO 40. USA: IEEE Computer Society, 2007, pp. 407–420.

[31] J. Meng, D. Tarjan, and K. Skadron, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," in Proceedings of the 37th annual international symposium on Computer architecture, Saint-Malo France: ACM, Jun. 2010, pp. 235–246.

[32] W. W. L. Fung and T. M. Aamodt, "Thread block compaction for efficient SIMT control flow," in 2011 IEEE 17th International Symposium on High Performance Computer Architecture, San Antonio, TX, USA: IEEE, Feb. 2011, pp. 25–36.

[33] Minsoo Rhu and M. Erez, "The dual-path execution model for efficient GPU control flow," in 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), Shenzhen: IEEE, Feb. 2013, pp. 591–602.

[34] A. ElTantawy, J. W. Ma, M. O'Connor, and T. M. Aamodt, "A scalable multi-path microarchitecture for efficient GPU control flow," in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, USA: IEEE, Feb. 2014, pp. 248–259.

[35] S.-Y. Lee and C.-J. Wu, "CAWS: criticality-aware warp scheduling for GPGPU workloads," in Proceedings of the 23rd international conference on Parallel architectures and compilation, Edmonton AB Canada: ACM, Aug. 2014, pp. 175–186.

[36] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, Porto Alegre Brazil: ACM, Dec. 2011, pp. 308–317.

[37] A. Jog et al., "OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance," in Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems, Houston Texas USA: ACM, Mar. 2013, pp. 395–406.

[38] K. Kim, S. Lee, M. K. Yoon, G. Koo, W. W. Ro, and M. Annavaram, "Warped-preexecution: A GPU pre-execution approach for improving latency hiding," in 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), Barcelona: IEEE, Mar. 2016, pp. 163–175.

[39] X. Gong, X. Gong, L. Yu, and D. Kaeli, "HAWS: Accelerating GPU Wavefront Execution through Selective Out-of-order Execution," ACM Trans. Archit. Code Optim., vol. 16, no. 2, pp. 1–22, Jun. 2019.

[40] K. Iliakis, S. Xydis, and D. Soudris, "Repurposing GPU Microarchitectures with Light-Weight Out-Of-Order Execution," IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 2, pp. 388–402, Feb. 2022.

[41] R. Huerta, J.-M. Arnau, and A. Gonzalez, "Simple Out of Order Core for GPGPUs," in Proceedings of the 15th Workshop on General Purpose Processing Using GPU, Montreal Canada: ACM, Feb. 2023, pp. 21–26.

[42] N. E. Jerger, T. Krishna, and L.-S. Peh, On-Chip Networks. in Synthesis Lectures on Computer Architecture. Cham: Springer International Publishing, 2017.

[43] G. Passas, M. Katevenis, and D. Pnevmatikatos, "VLSI micro-architectures for high-radix crossbar schedulers," in Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip, Pittsburgh Pennsylvania: ACM, May 2011, pp. 217–224.

HoBBy: Hardening Unbalanced Branches against Control Flow Attacks on Intel SGX and AMD SEV

Chang Liu[†*], Shuaihu Feng[‡*], Yuan Li[‡¶], Dongsheng Wang[†‡], Trevor E. Carlson[§]
[†]Tsinghua University, Beijing, China
[‡]Zhongguancun Laboratory, Beijing, China
[§]National University of Singapore, Singapore

***Abstract*—This paper introduces HoBBy, a compiler-based tool that hardens unbalanced branches at the instruction level, making parallel control flows indistinguishable to state-of-the-art attacks that bypass the source-code level balancing. To achieve this, we propose a single-step analysis method to identify unbalanced instructions in secret-dependent branches, and implement instruction shadowing, cogging, and spiraling techniques to protect them. We evaluate HoBBy by hardening secret-dependent branches in four real-world applications, validating its resilience against three state-of-the-art attacks targeting Intel SGX and AMD SEV. HoBBy achieves a runtime overhead of 2.8% for cryptographic libraries and a binary size overhead of 0.6%.**

I. INTRODUCTION

Modern CPUs incorporate Trusted Execution Environments (TEEs), such as Intel Software Guard Extensions (SGX) [16] and AMD Secure Encrypted Virtualization (SEV) [3], to safeguard the instructions and data of applications, preventing data leakage from malicious programs, including the operating system. However, TEE designers do not comprehensively address side-channel threats [27], which has led to various control flow attacks [5], [9], [26] that expose control flow information, such as the target of a secret-dependent branch, posing significant risks to applications like cryptographic libraries used for RSA [23], [39] and ECDSA [22] algorithms.

To defend against these attacks, researchers propose constant-time programming, which maintains constant execution time regardless of the specific data being processed. Two primary methods are available for implementing constant-time programming. The first approach is to avoid making control flow directly dependent on secrets [7], [15], thereby preventing control-flow attacks by eliminating associations between sensitive data and control flow. However, this technique is often limited to specific cryptographic algorithms [9] and may introduce significant performance overhead [7], [26]. The second method is branch balancing, also known as code obfuscation [1], [36], which ensures equal execution time across branch paths, making it difficult for attackers to distinguish between paths using CPU side channels. Branch balancing is algorithm-agnostic and preserves program semantics, enhancing its deployability.

However, existing branch balancing implementations still expose vulnerabilities and have been bypassed by recent attacks [5], [25], [26], primarily due to the following three reasons. First, current branch balancing techniques are designed mainly to counter traditional power, frequency and timing side-channel attacks [35], [41], ensuring only that the overall execution time of two branch paths remains consistent. They overlook that attackers with root privileges have access to advanced techniques, such as single-step execution [30], [31], [37], which enable detection of information at the instruction level. Second, current branch balancing is typically implemented at the source-code level [22], [23] and does not account for imbalances introduced during compilation, which may introduce new attack surfaces [25], [32]. Third, existing branch balancing does not fully consider the influence of the Program Counter (PC) [36]. For example, even if two branches contain identical instructions, differing PCs can still impact CPU behavior in ways exploitable by attackers [5], [26]. Currently, no approach has been proposed to harden secret-dependent branches at the instruction level.

In this paper, we propose HoBBy, an algorithm-agnostic, compile-based software defense that Hardens unBalanced Branches at the instruction level. By ensuring consistency in instructions (i.e., undistinguishable instruction types, counts, data flow, and PC targets) on both branch paths during compilation, HoBBy eliminates side-channel information exploitable by attackers, effectively defending against such threats. To achieve this, HoBBy first performs single-step analysis through instrumentation to evaluate the balance of secret-dependent branches within applications, identifying unbalanced instruction sequences that may be vulnerable to side-channel leakage. Next, HoBBy employs instruction and data shadowing, replicating instructions and registers without altering program semantics, to maintain balance in instruction type, count, and data flow across both branch paths. Finally, HoBBy adjusts the layout of original and shadow instructions within memory pages, merging store and load instructions to maintain the PC targets balance.

We implement HoBBy with LLVM [21] and apply it to harden the unbalanced branches in three cryptographic [22], [23], [39] and one non-cryptographic applications [19]. To assess HoBBy's attack resilience, we show that HoBBy can drop the accuracy of three state-of-the-art attacks, two on Intel SGX [5], [28] and one on AMD SEV [9], to the equivalent of random guessing (i.e., approximately 50%). Finally, we evaluate HoBBy's performance overhead, showing that HoBBy introduces an average runtime overhead of 2.8% for

* These authors contributed equally to this work.
¶ Corresponding author.

the cryptographic libraries, with a binary size overhead of 0.6%, which is much lower compared to other mitigations [7].

In summary, the contributions of this paper are as follows:

- We introduce HoBBy, a tool that hardens unbalanced branches at the instruction level in real-world applications, and defends against state-of-the-art control flow attacks on these applications.
- We propose a method to evaluate the security of secret-dependent branches, enabling automated analysis of unbalanced instructions along branch paths and identifying potential side-channel risks.
- We develop various methods to balancing branch in terms of instruction type, count, data address and PC target.
- We evaluate HoBBy's effectiveness and performance overhead in 4 real-world applications, demonstrating that HoBBy defends against state-of-the-art attacks on Intel SGX and AMD SEV with an average runtime overhead of approximately 2.8% for the cryptographic libraries, and a binary size overhead of approximately 0.6%.

II. BACKGROUND ON CONTROL FLOW ATTACKS

Control flow side-channel attacks infer which code of an application running in the TEE is executed by detecting changes in architecture-level or microarchitecture-level states, thereby leaking secrets related to control flow. Secret-dependent branches [5], [14] are one of the primary targets of such attacks, as the execution of these branches depends on a secret determining whether the branch is taken or not. Secret-dependent branches are commonly found in cryptographic applications, such as RSA and ECDSA [22], [23], [39], and are also present in non-cryptographic applications [18], [19].

Current control flow attacks targeting secret-dependent branches can be divided into two categories. The first category is branch-oriented attacks [14], [17], [42], [43], which exploit the effects of branch instructions on the branch predictor to leak sensitive data. The second category is non-branch-oriented attacks [5], [9], [25], [26], [28], [40], which infer the branch direction by exploiting side channels of other instructions along the branch paths.

For branch-oriented attacks, the leakage source is known, and defenses can be implemented by flushing the branch predictor during context switches [42], [43] or confounding the control flow through compiler [12], [17]. Therefore, such attacks are not considered in this paper. For non-branch-oriented attacks, imbalances between the two branch paths, such as differences in instruction count [25], instruction types [2], [9], data addresses [12], [28], or instruction addresses (PC) [5], [26], are exploited to infer branch paths, as shown in Table I. This type of attack leverages various complex microarchitectural side channels and diverse code patterns, posing significant challenges to effective defenses.

TABLE I
NON-BRANCH-ORIENTED CONTROL FLOW ATTACKS ON SECRET-DEPENDENT BRANCHES IN INTEL SGX AND AMD SEV. PAGE MEANS PAGE ATTACKS OR CONTROLLED CHANNEL ATTACKS, WHILE CACHE REFERS TO CACHE SIDE-CHANNEL ATTACKS.

Attack Type / Name	Source	Imbalance Requirement
Page [12], [33], [34], [40]	Interrupt	Data Flow & PC (4 KiB)
Nemisis [32]	Interrupt	Type
Copycat [25]	Interrupt	Count
Frontal Attack [26]	Frontend	PC (Store)
TLBleed [11]	TLB	Data Flow & PC (4 KiB)
PortSmath [2]	Port	Type & Count
Cache [4], [13], [24], [28]	Cache	Data Flow & PC (64 B)
AfterImage [5]	Prefetcher	PC (Load)
SQUIP [9]	Scheduler	Type & Count

III. OVERVIEW OF HOBBY

A. Motivation

Branch balancing prevents various side-channel attacks by ensuring identical CPU behavior on both sides of a branch. This approach does not modify program algorithms and has therefore already been adopted by some developers [22], [23], [39]. However, current branch balancing is primarily implemented at the source-code level, without addressing imbalances introduced during compilation [25] or leakage from instruction PCs due to specific CPU characteristics [5], [26]. As a result, branch balancing is not always effective against control flow attacks. In this work, we look at the fundamental root causes of branch balancing issues and propose new compiler-based techniques to eliminate branch-balancing issues. Specifically, we aim to strengthen balanced branches at the instruction level to defend against state-of-the-art non-branch-oriented attacks, such as those listed in Table I.

B. Assumptions and Threat Model

We assume an attacker with root privileges (e.g., a malicious OS) aims to leak the direction of a secret-dependent branch within the TEE application using control flow attacks. The attacker can manage page tables and handle exceptions to execute victim code in single-step mode (e.g., using SGX-Step [31] or SEV-Step [37]) and perform side-channel attacks. The victim is a user application running in the TEE, and seeks to harden its code against control flow attacks through compiler-based defenses. Both the victim and attacker can profile the binary file and identify secret-dependent branches. The victim can compile, modify, and execute the binary, while the attacker can only analyze the binary.

C. Workflow of HoBBy

The workflow of HoBBy is illustrated in Fig. 1. The input is the source code containing secret-dependent branches that require hardening. First, HoBBy automatically identifies these branches and their paths. Next, HoBBy instruments each instruction along the branch paths to analyze the context of each instruction. After the analysis, HoBBy balances the type, count, data flow, and PC of instructions along the branch paths to defend against control flow attacks. The output is a hardened application, preventing privileged attackers from using side channels to distinguish between different branch paths.

In Sections IV and V, we introduce the design of secret-dependent branch analysis (❶) and branch balancing (❷),

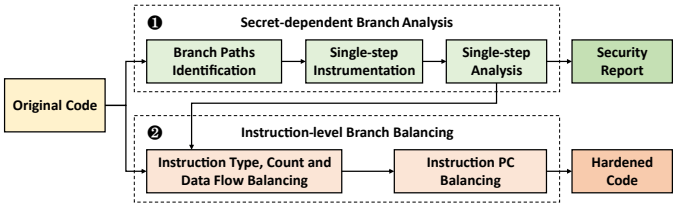


Fig. 1. Workflow of HoBBy. HoBBy first identify and evaluate the security of secret-dependent branches (❶), and then balancing the instruction type, count, data flow and PC target (❷).

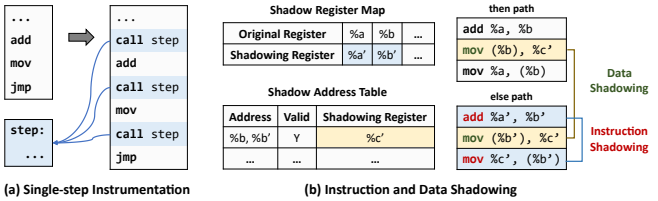


Fig. 2. Methods to perform single-step analysis at the instruction level (a), and to implement instruction and data shadowing (b) so that instruction types and memory accesses are undistinguishable on the two branch paths.

respectively. In Section VI, we present the effectiveness and performance evaluation of HoBBy.

IV. SECRET-DEPENDENT BRANCHES ANALYSIS

In this section, we introduce HoBBy's single-step analysis of secret-dependent branches. This step identifies unbalanced and potentially vulnerable instruction sequences, which will be hardened in subsequent steps.

A. Branch Paths Identification

Similar to previous research [8], we use DFSan [20] to perform taint analysis for identifying secret-dependent branches. Next, we analyze the program's Control Flow Graph (CFG) to determine the control flow in both directions of each branch. For most secret-dependent branches, only a small portion of the secret (e.g., a single bit) is processed per execution, causing these branches to be executed multiple times. Consequently, there is often a common successor across different paths for the majority of secret-dependent branches. Based on this observation, we first use Breadth-First Search (BFS) to identify the nearest common successor among different branch paths.

Subsequently, we use Depth-First Search (DFS) to trace all paths from the basic block containing the secret-dependent branch to the nearest common successor. During DFS, other branches may be encountered along these paths (e.g., branches for loops or exception handlers [23], [39]), adding complexity to the control flow. To handle loops, we perform a static analysis to partition the loop into the loop head and loop body, ensuring that each loop body is analyzed only once. For branches handling exceptions, we apply a heuristic approach: if the number of predecessors for the branch target exceeds a specified threshold, and the basic block's successor is unreachable from the nearest common successor on the branch path, we skip the analysis of that basic block and its successors.

B. Single-step Instrumentation

According to the threat model described in Section III-B, the attacker can interrupt the victim's code [45] at the instruction level [30] and perform side-channel attacks. To analyze the vulnerability of branches, we conduct a dynamic single-step analysis on the instructions along the branch paths after identifying them. To achieve this, we insert a `call` instruction targeting the function `step` between each instruction, as shown in Fig. 2(a). In each `step`, we save the current context by sequentially storing the values of the `EFLAGS` register and general-purpose registers, followed by values in the call stack. Additionally, for store and load instructions, we capture their virtual addresses for later offline analysis.

To ensure that `step` does not alter the program's context, we save the current stack frame register `RSP` before executing `step`, then adjust it to a new stack space to store the execution context. Furthermore, to retrieve information about the executing instruction, we additionally save the PC stored in `RIP`. Given that the length of the instrumented instructions is known, we can dynamically calculate the PC of each analyzed instruction and access the corresponding instruction page to retrieve its machine code. Finally, after `step` completes, we restore the context and adjust the stack base to ensure the program continues to execute in single-step mode.

C. Analysis on Unbalanced Instructions

After collecting the context of each instruction along the branch paths, we perform an offline analysis to determine whether the instructions are balanced and, if not, identify which instructions might be vulnerable to side-channel attacks. First, we disassemble the machine code to obtain the opcodes and then count the number of each opcode type on each path. If the instruction counts differ, or if there are different types of instructions, the branch may be vulnerable to port-based [2], [9] or interrupt-based attacks [25], [32]. Next, we analyze the presence of load and store instructions within the branch paths. If such instructions exist, the branch could be vulnerable to the attacks targeting the PC of these instructions [5], [26]. Finally, we analyze the operands of these instructions to determine whether the addresses for loads and stores are in different pages or cache lines, where hardening is required because distinct data flows may be vulnerable to side-channel attacks related to page [40], TLB [11], and cache [10], [28].

V. HARDENING UNBALANCED BRANCHES

In this section, we introduce HoBBy's methods for hardening unbalanced instruction types, instruction counts, data flow, and the PC targets. Even if an attacker can execute the victim code in single-step mode, identical instructions from the victim program are executed, and the same data addresses are accessed, which effectively prevents these control flow attacks.

A. Instruction Balancing with Instruction Shadowing

HoBBy first balances instruction types and counts through *instruction shadowing*. As shown in Fig. 2(b), we start by creating a shadow register mapping table that maps all register operands in unbalanced instructions to shadow registers.

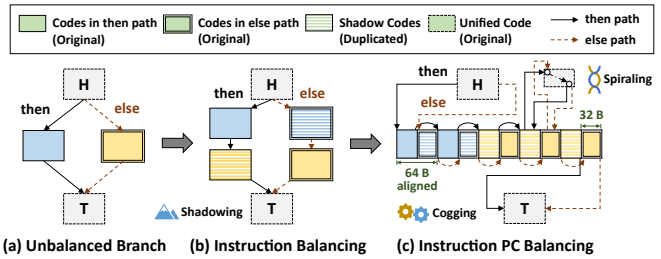


Fig. 3. Workflow of hardening the unbalanced branches. First, the instruction type, count and data flow are balanced through instruction and data shadowing (b). Next, the instruction PC target is balanced through instruction cogging and instruction spiraling (c).

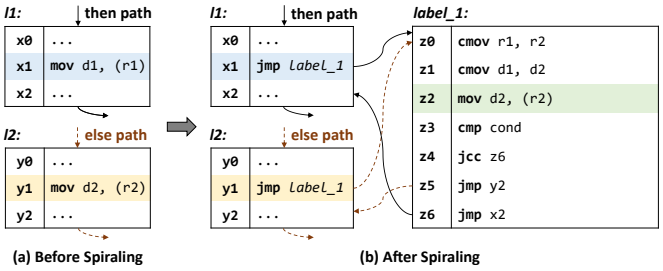


Fig. 4. Method to implement instruction spiraling. The load or store instructions on two branch paths can be merged to hide the side-channel effects caused by these instructions with different PCs.

Shadow registers are registers outside the program's original semantics and are not used as operands in the original instructions, and thus do not affect the program's behavior.

As shown in Fig. 3(b), if there are imbalances in instruction types or counts between the two branch paths, we replicate the basic blocks. We then insert new shadow instructions into the replicated basic blocks, adjusting the instruction opcodes to ensure that both branch paths contain the same number and types of specific instructions. When adding these instructions, we maintain the same relative position from the entry of each basic block so that, during the single-step execution, identical instruction types are executed on both branch paths. For shadow instructions, we ensure their target registers are always shadow registers, preventing the execution of these instructions from impacting the program's original context. For example, as shown in Fig. 2(b), if an `add` instruction exists on one branch path but not on the other, we replicate the basic block on the other path and add a shadow `add` instruction with operands drawn from the shadow register map.

B. Data Flow Balancing with Data Shadowing

Since store and load instructions trigger data accesses and impact CPU backend units such as the cache, we need to balance their memory addresses to avoid data flow imbalance through *data shadowing*. For load instructions, we access the same address in the shadow instructions. For store instructions, to prevent shadow instructions from modifying memory, we insert a shadow load before the shadow store to retrieve the original data from that address, as shown in Fig. 2(b). To balance the data flow, we also insert a load on the original path that accesses the same address but uses a shadow register as the destination, ensuring program correctness is maintained.

C. PC Target Balancing with Instruction Cogging

In modern CPUs, instructions are fetched from memory, so even if the types and counts of instructions are identical on both branch paths, instruction fetching can still affect the cache and TLB, potentially creating side-channel leakage risks [33]. Therefore, it is necessary to balance the instruction PC target to ensure that both original and shadow instructions reside on the same page and access the same instruction cache line.

To achieve this, we introduce the *instruction cogging* technique. As shown in Fig. 3(c), after completing instruction balancing, we divide the instructions on both branch paths into 32-byte blocks and alternately place these blocks within a 64-byte aligned address space, connecting the two paths with `jmp` instructions to maintain semantic correctness. This ensures that, even if an attacker executes instructions in single-step mode, balanced instructions from both paths appear simultaneously within the same instruction page and cache line, preventing related side-channel attacks.

D. PC Target Balancing with Instruction Spiraling

Recent attacks exploit the different effects of stores or loads with different PCs on CPU microarchitecture to infer which stores or loads are executed on different paths. For example, the Frontal Attack [26] leverages variations in single-step execution time for a store based on its distance from the conditional branch. AfterImage [5] takes advantage of loads with different 8 least significant bits (LSBs) in their PCs, which update distinct entries in the IP-stride prefetcher.

To defend against such attacks, only balancing the instruction and data flow is insufficient. Therefore, we propose the *instruction spiraling* technique, which merges the store and load instructions from both paths, as shown in Fig. 4. In the instruction balancing phase, we have ensured balanced store and load instructions and data flow on both branch paths using instruction and data shadowing. Now for each pair of balanced stores or loads, we create a new basic block in the CFG and replace the original stores or loads on both paths with `jmp` instructions targeting this new basic block.

In this new basic block, we insert a `cmov` instruction that, based on the branch condition, determines the source and destination registers for the stores or loads. This setup ensures that the same store or load with a fixed PC is executed on both paths each time, producing consistent effects on CPU microarchitecture. To maintain correct control flow, the program jumps back to one of the original paths based on the branch condition after executing the store or load in the new basic block, ensuring that execution proceeds as intended.

VI. EVALUATION ON HOBBY

We implement HoBBy in the LLVM backend [21] for the x86-64 instruction set. Specifically, we include single-step analysis, instruction shadowing, data shadowing, and instruction spiraling for stores and loads. Our implementation does not cover 32-bit instructions, extended instruction sets, or

TABLE II

SINGLE-STEP ANALYSIS ON THE UNBALANCED INSTRUCTIONS OF 8 SECRET-DEPENDENT BRANCHES IN 4 REAL-WORLD APPLICATIONS BEFORE HARDENING AND AFTER HARDENING. THE LAST 2 COLUMNS SHOW THE AVERAGE RUNTIME OVERHEAD OF THE HARDENING FOR THE FUNCTIONS THAT CONTAIN THE SECRET-DEPENDENT BRANCHES, AND THE WHOLE APPLICATIONS.

Application	Secret-dependent Branch Location	Imbalance Before Hardening				Imbalance After Hardening	Performance Overhead	
		Count	Type	Data Flow	PC		Function	Program
MbedTLS v2.6.1 [22]	bignum.c, line 1013	1	2	0	14	0	14.3%	2.8%
	bignum.c, line 1044	1	2	0	14			
	bignum.c, line 1581	0	0	0	10			
MbedTLS v3.6.1 [23]	bignum.c, line 1823	1	0	0	10	0	5.2%	0.5%
	bignum.c, line 1959	0	0	0	18			
WolfSSL v5.7.2 [39]	integer.c, line 1329	0	0	0	18	0	0.4%	0.2%
Libjpeg v9f [19]	jidctint.c, line 208	69	7	42	27	0	66.4%	19.3%
	jidctint.c, line 325	71	5	43	33			

system instructions, as these are not present in the programs we analyzed. We leave these aspects for future work. In this section, we selected Libjpeg [19], WolfSSL [39], and MbedTLS [22], [23] for analysis, because recent research has identified secret-dependent branches in these applications as attack targets. We use single-step analysis before and after hardening to demonstrate HoBBy's effectiveness. We then show that HoBBy can defend against three state-of-the-art attacks targeting Intel SGX and AMD SEV. Finally, we present the performance overhead of HoBBy.

A. Effectiveness

In Table II, we select a few secret-dependent branches from each program and report the number of unbalanced instructions before and after hardening. The developers of WolfSSL and MbedTLS have considered side-channel attacks and implemented source-code level balancing (i.e., apart from function parameters, the code on both sides of the branches is identical). However, these branches still exhibit risks due to unbalanced instructions [5], [25], [26]. Through instruction balancing and PC target balancing, HoBBy eliminates the imbalance and mitigates potential side-channel risks.

As an example from a non-cryptographic library, Libjpeg does not consider branch balancing during development. Instead, Libjpeg's IDCT algorithm uses branching to skip complex computations when the image data is simple (e.g., a pixel block consists of a single color), and HoBBy efficienctly hardens these unbalanced instructions.

B. Case Studies

In this section, we select three state-of-the-art control-flow attacks to demonstrate the effectiveness of HoBBy. These attacks exploit imbalances in data flow, instruction count, and PC, respectively. For the victim, we use the secret-dependent branches in the modular inverse function of MbedTLS [23]. We do not implement a full end-to-end attack; instead, we achieve synchronization by actively switching between attacker and victim. This approach simplifies the attack process but is sufficient to demonstrate HoBBy's effectiveness.

Cache Attacks [4], [28] exploit the imbalance in the data flow. If the victim's load instructions on different branch paths access different data addresses, an attacker can use cache side channels to determine which load is executed. As shown in Fig. 5(a), if two loads access distinct cache sets, the attacker can distinguish which cache set the victim accessed after each branch execution using Prime+Probe [28], achieving a 99.75% success rate. HoBBy applies data shadowing to ensure that loads on both paths access the same cache sets, so the attacker always observes both cache sets as accessed, rendering the attack ineffective (i.e., equivalent to random guessing).

SQUIP [9] leverages the difference in the number of arithmetic instructions (e.g., multiplications) on different branch paths to create contention in the scheduler of AMD CPU execution units, thereby revealing which path the victim's code take. As shown in Fig. 5(b), the attacker's code exhibits different execution times for victim paths with varying numbers of multiplications, resulting in a 92.77% attack success rate. After hardening, the instruction counts on both paths are identical, and the execution time of the attacker's code becomes the same, reducing the attack success rate to around 50%.

AfterImage [5] exploits the differing effects of loads with distinct PCs on the prefetcher to infer which load is executed. For example, if a load on a particular path is executed, the prefetcher entry associated with that load's PC is updated; otherwise, the prefetcher entry remains unchanged. By repeatedly executing the victim code and using a cache side channel to detect the prefetcher update rate, the attacker can infer whether the load is executed. As shown in Fig.5(c), when the update rate is zero, the load is not executed, indicating that the branch does not follow that path, with a success rate of 98.89%. HoBBy effectively mitigates AfterImage by merging loads on both paths through instruction spiraling, so the attacker always observes a prefetcher update, making the attack ineffective.

C. Performance Overhead

We analyze the performance overhead of HoBBy on both the functions containing the secret-dependent branches and the

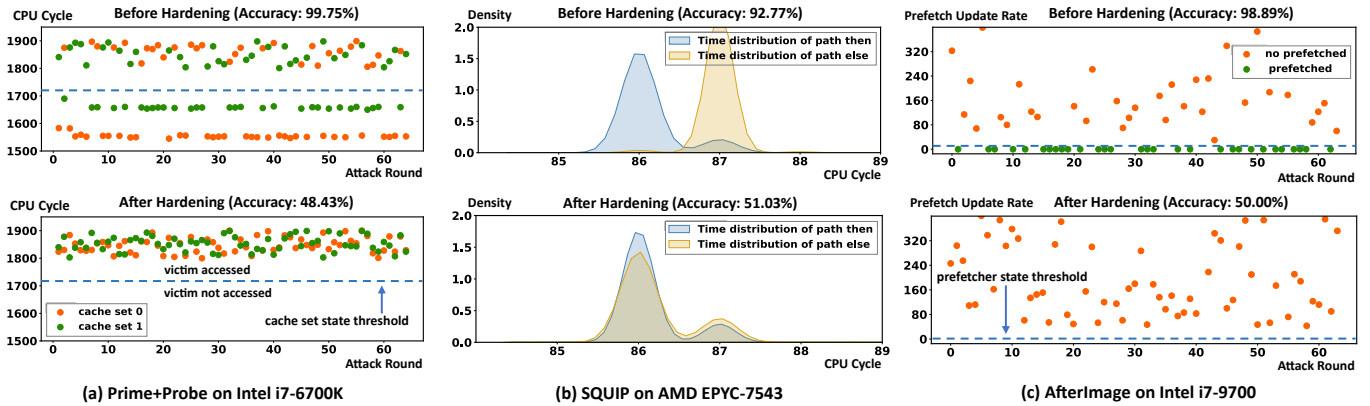


Fig. 5. Case studies on three state-of-the-art control flow attacks on Intel SGX and AMD SEV. HoBBy effectively defend these attacks by balancing the data flow [28], count of `imul` instructions [9] and load PC [5], making different paths of the secret-dependent branches not distinguishable.

overall program, as shown in the last two columns of Table II. For cryptographic libraries, we focus on the RSA algorithms as they are the most common attack targets in recent studies [5], [9], [28]. For Libjpeg, we focus on the IDCT function.

While hardening introduces runtime overhead, the overall performance impact remains acceptable given the security benefits. For instance, Libjpeg's IDCT function sees a significant 66.4% overhead due to disrupted optimizations, but the total overhead for Libjpeg is only 19.3%, which is acceptable in many cases. In cryptographic libraries, if the hardened function isn't on the critical execution path, the performance impact is minimal. For example, hardening the modular inverse function in MbedTLS results in just a 2.8% overall runtime overhead, indicating a negligible performance penalty.

We also evaluate the binary size overhead introduced by HoBBy. The additional instructions primarily stem from shadow instructions introduced by instruction and data shadowing and `jmp` instructions added by instruction spiraling. For branches that are not balanced at the source-code level (e.g., Libjpeg's IDCT), the number of added instructions is significantly higher than in other programs. However, in terms of overall binary size, these additional instructions do not have notable effects. For example, the size increase in WolfSSL and Libjpeg due to HoBBy does not exceed 0.6%.

VII. Related Work

Currently, various defenses have been proposed to mitigate control-flow attacks on Intel SGX and AMD SEV. First, as suggested by CPU vendors [15], preventing control flow from depending on secrets can fundamentally block all control-flow attacks [7]. However, this approach is difficult to extend to complex programs [9] and incurs significant overhead [7]. Second, code-oblivious techniques hide timing differences across different control flows. SGX-Shield [29] and Klotski [44] achieve branch control-flow balancing at the page level but cannot defend against finer-grained control-flow attacks. Compiler-Assisted Hardening [38] and Obfuscuro [1] offer finer-grained instruction balancing but do not address leaks due to PC imbalances. Obelix [36] is a state-of-the-art tool that protects both code and data within the TEE, defending against single-step control-flow attacks and ciphertext side channels [8] while also addressing PC imbalances [26]. However, Obelix primarily focuses on ensuring constant execution time for instructions during single-step execution and does not consider the persistent impact of specific instruction PCs on other CPU components, as highlighted in AfterImage [5].

In addition, some defenses are designed to mitigate specific attacks. First, by flushing the state of leakage sources during context switches, attacks with known sources can be mitigated, such as the BPU [42], [43] and Prefetcher [5] attacks. This approach is efficient and does not require modifications to the program, but is limited to specific attacks. Second, AEX-Notify [6] detects single-step attacks by monitoring the frequency of interrupts. However, it requires additional CPU support (e.g., microcode updates), which makes it unavailable for immediate deployment [36]. Besides, other TEEs (e.g., AMD SEV), currently lack support for similar techniques.

VIII. Conclusion

We present HoBBy, a tool that hardens unbalanced secret-dependent branches at the instruction level in applications running within a TEE. HoBBy balances instruction types, counts, data flow, and PC targets by employing single-step analysis, effectively defending against control-flow attacks on Intel SGX and AMD SEV. We demonstrate its effectiveness across four real-world applications, showcasing its ability to counter state-of-the-art control-flow attacks targeting these platforms. HoBBy incurs runtime and binary storage overheads of 2.8% and 0.6% for the RSA algorithm in cryptographic libraries, respectively. Overall, HoBBy provides strong security guarantees for secret-dependent branches within the TEE with acceptable overhead.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China (Grant No. 62372258) and the Singapore Ministry of Education (Grant No. T2EP20222-0026). Chang Liu was supported by China Scholarship Council (Student No. 202406210249).

References

[1] A. Ahmad, B. Joe *et al.*, "OBFUSCURO: A Commodity Obfuscation Engine on Intel SGX," in *Network and Distributed System Security Symposium (NDSS)*, 2019.

[2] A. C. Aldaya, B. B. Brumley *et al.*, "Port Contention for Fun and Profit," in *Symposium on Security and Privacy (SP)*, 2019, pp. 870–887.

[3] AMD, "AMD64 Architecture Programmer's Manual Volume 2: System Programming," https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24593.pdf, 2024.

[4] F. Brasser, U. Müller *et al.*, "Software Grand Exposure: SGX Cache Attacks Are Practical," in *USENIX Workshop on Offensive Technologies (WOOT)*, 2017.

[5] Y. Chen, L. Pei, and T. E. Carlson, "AfterImage: Leaking Control Flow Data and Tracking Load Operations via the Hardware Prefetcher," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2*, 2023, pp. 16–32.

[6] S. Constable, J. Van Bulck *et al.*, "AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves," in *USENIX Security Symposium (USENIX Security)*, 2023, pp. 4051–4068.

[7] B. Coppens, I. Verbauwhede *et al.*, "Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors," in *Symposium on Security and Privacy (SP)*, 2009, pp. 45–60.

[8] S. Deng, M. Li *et al.*, "CipherH: Automated Detection of Ciphertext Side-channel Vulnerabilities in Cryptographic Implementations," in *USENIX Security Symposium (USENIX Security)*, 2023, pp. 6843–6860.

[9] S. Gast, J. Juffinger *et al.*, "SQUIP: Exploiting the Scheduler Queue Contention Side Channel," in *Symposium on Security and Privacy (SP)*, 2023, pp. 2256–2272.

[10] J. Götzfried, M. Eckert *et al.*, "Cache Attacks on Intel SGX," in *Proceedings of the European Workshop on Systems Security (EuroSec)*, 2017, pp. 1–6.

[11] B. Gras, K. Razavi *et al.*, "Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks," in *USENIX Security Symposium (USENIX Security)*, 2018, pp. 955–972.

[12] J. Gyselinck, J. Van Bulck *et al.*, "Off-Limits: Abusing Legacy x86 Memory Segmentation to Spy on Enclaved Execution," in *Engineering Secure Software and Systems (ESSoS)*, 2018, pp. 44–60.

[13] M. Hähnel, W. Cui, and M. Peinado, "High-Resolution Side Channels for Untrusted Operating Systems," in *USENIX Annual Technical Conference (USENIX ATC)*, 2017, pp. 299–312.

[14] A. Hajiabadi and T. E. Carlson, "Conjuring: Leaking Control Flow via Speculative Fetch Attacks," in *Design Automation Conference (DAC)*, 2024, pp. 1–6.

[15] Intel, "Guidelines for Mitigating Timing Side Channels Against Cryptographic Implementations," https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html.

[16] Intel, "Intel® 64 and IA-32 Architectures Software Developer Manuals," https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, 2024.

[17] S. Lee, M.-W. Shih *et al.*, "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing," in *USENIX Security Symposium (USENIX Security)*, 2017, pp. 557–574.

[18] M. Lexa and G. Valle, "PRIMEX: rapid identification of oligonucleotide matches in whole genomes," *Bioinformatics*, vol. 19, no. 18, pp. 2486–2488, 2003.

[19] libjpeg, "libjpeg version 9f," https://www.ijg.org/files/jpegsrc.v9f.tar.gz, 2024. [Online]. Available: https://www.ijg.org/files/jpegsrc.v9f.tar.gz

[20] llvm, "Dfsan," https://clang.llvm.org/docs/DataFlowSanitizer.html, 2020.

[21] llvm, "llvm-project," https://github.com/llvm/llvm-project, 2024.

[22] Mbedtls, "Mbedtls version 2.6.1," https://github.com/Mbed-TLS/mbedtls/tree/mbedtls-2.6.1, 2017. [Online]. Available: https://github.com/Mbed-TLS/mbedtls/tree/mbedtls-2.6.1

[23] Mbedtls, "Mbedtls version 3.6.1," https://github.com/Mbed-TLS/mbedtls/tree/mbedtls-3.6.1, 2024. [Online]. Available: https://github.com/Mbed-TLS/mbedtls/tree/mbedtls-3.6.1

[24] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX Amplifies the Power of Cache Attacks," in *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2017, pp. 69–90.

[25] D. Moghimi, J. Van Bulck *et al.*, "CopyCat: Controlled Instruction-Level Attacks on Enclaves," in *USENIX Security Symposium (USENIX Security)*, 2020, pp. 469–486.

[26] I. Puddu, M. Schneider *et al.*, "Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend," in *USENIX Security Symposium (USENIX Security)*, 2021, pp. 663–680.

[27] M. Schwarz and D. Gruss, "How Trusted Execution Environments Fuel Research on Microarchitectural Attacks," *IEEE Security & Privacy*, vol. 18, no. 5, pp. 18–27, 2020.

[28] M. Schwarz, S. Weiser *et al.*, "Malware Guard Extension: Using SGX to Conceal Cache Attacks," in *International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2017, pp. 3–24.

[29] J. Seo, B. Lee *et al.*, "SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs," in *Network and Distributed System Security Symposium (NDSS)*, 2017.

[30] D. Skarlatos, M. Yan *et al.*, "MicroScope: enabling microarchitectural replay attacks," in *Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 318–331.

[31] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control," in *Proceedings of the Workshop on System Software for Trusted Execution (SysTEX)*, 2017, pp. 1–6.

[32] J. Van Bulck, F. Piessens, and R. Strackx, "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic," in *Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 178–195.

[33] J. Van Bulck, N. Weichbrodt *et al.*, "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution," in *USENIX Security Symposium (USENIX Security)*, 2017, pp. 1041–1056.

[34] W. Wang, G. Chen *et al.*, "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX," in *Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 2421–2434.

[35] Y. Wang, R. Paccagnella *et al.*, "Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86," in *USENIX Security Symposium (USENIX Security)*, 2022, pp. 679–697.

[36] J. Wichelmann, A. Rabich *et al.*, "Obelix: Mitigating Side-Channels Through Dynamic Obfuscation," in *Symposium on Security and Privacy (SP)*, 2024, pp. 189–189.

[37] L. Wilke, J. Wichelmann *et al.*, "SEV-Step: A Single-Stepping Framework for AMD-SEV," *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2024.

[38] H. Winderix, J. T. Mühlberg, and F. Piessens, "Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks," in *European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 667–682.

[39] wolfSSL, "wolfssl version 5.7.2-stable," https://github.com/wolfSSL/wolfssl, 2024. [Online]. Available: https://github.com/wolfSSL/wolfssl

[40] Y. Xu, W. Cui, and M. Peinado, "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems," in *Symposium on Security and Privacy (SP)*. IEEE, 2015, pp. 640–656.

[41] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: a timing attack on OpenSSL constant-time RSA," *Journal of Cryptographic Engineering*, vol. 7, pp. 99–112, 2017.

[42] H. Yavarzadeh, A. Agarwal *et al.*, "Pathfinder: High-Resolution Control-Flow Attacks Exploiting the Conditional Branch Predictor," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3*, 2024, pp. 770–784.

[43] J. Yu, T. Jaeger, and C. W. Fletcher, "All Your PC Are Belong to Us: Exploiting Non-control-Transfer Instruction BTB Updates for Dynamic PC Extraction," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–14.

[44] P. Zhang, C. Song *et al.*, "Klotski: Efficient Obfuscated Execution against Controlled-Channel Attacks," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 1263–1276.

[45] X. Zhang, Z. Zhang *et al.*, "SegScope: Probing Fine-grained Interrupts via Architectural Footprints," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 424–438.

Delving into Topology Representation for Layout Pattern: A Novel Contrastive Learning Framework for Hotspot Detection

Silin Chen[1], Kangjian Di[1], Guohao Wang[4], Wenzheng Zhao[4], Li Du[2], Ningmu Zou[1,3]*,
[1]School of Integrated Circuits, Nanjing University, Suzhou, China
[2]School of Electronic Science and Engineering, Nanjing University, Nanjing, China
[3]Interdisciplinary Research Center for Future Intelligent Chips (Chip-X), Nanjing University, Suzhou, China
[4]ZetaTech Co.,Ltd.

***Abstract*—Recently, machine learning-based techniques have been applied for layout hotspot detection. However, existing methods encounter challenges in capturing the decision boundary across the entire dataset and ignore the geometric properties and topology of the polygons. In this paper, we introduce CLI-HD, a novel contrastive learning framework on layout sequences and images for hotspot detection. Our framework improves the ability to distinguish between hotspots and non-hotspots by similarity computations instead of a single decision boundary. To effectively incorporate geometric information into the model training process, we propose Layout2Seq, which encodes polygon shapes as vectors within sequences that are subsequently fed into the CLI-HD. Furthermore, to better represent topology information, we develop an absolute position embedding, replacing the standard position encoders used in Transformer architectures. Extensive evaluations on various benchmarks demonstrate that CLI-HD outperforms current state-of-the-art methods, with an accuracy improvement ranging from 0.82% to 4.77% and a reduction in false alarm rates by 4.9% to 23.18%.**

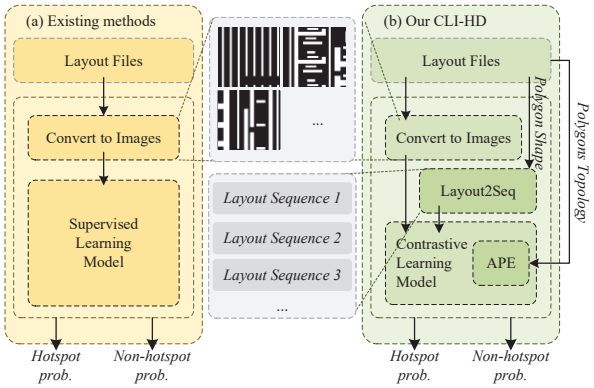


Fig. 1. Illustration of the hotspot detection flow. (a) The pipeline of existing supervised learning-based methods. (b) The pipeline of our proposed CLI-HD.

I. Introduction

With the rapid development of semiconductor techniques, the shrinking of transistor feature sizes and the increasing size of dies present significant challenges to Design for Manufacturability (DFM). Hotspot detection is introduced as a critical procedure in DFM to identify potential defect patterns early. Lithography simulation techniques can accurately detect hotspot patterns, but they consume a large amount of time for simulation and are not applicable to chip fabrication. Other works were based on pattern-matching methods to identify hotspots with similar patterns by retrieval or clustering. Such methods cannot cover hotspots outside the pattern or layout libraries [1], [2].

Recently, machine learning-based methods have been applied in DFM [3], [4] and achieved impressive progress in hotspot detection [5], [6]. As shown in Fig. 1(a), existing methods convert the layout files into binary images and employ supervised learning to classify HotSpot (HS) and Non-HotSpot (NHS). However, these methods utilizing convolutional neural networks (CNNs) to distinguish hotspots are limited by the receptive field in acquiring the global topology information of the layout images. To represent the global topology, others utilize polygons to construct a layout graph and perform pattern graph classification with graph neural networks (GNNs) for hotspot detection [7], [8]. Nevertheless, they decompose the polygon into multiple rectangular nodes, which introduces additional complexity and results in the shape information loss of the polygon. Generally, existing CNN-based and GNN-based methods rely on supervised learning, which tends to suffer from a high false-alarm rate when trained to construct a single decision boundary across the entire dataset with very few hotspot samples [6].

Beyond image and graph-based approaches, alternative layout representations have been explored in other DFM applications. A sequence pattern representation [9] is proposed for layout pattern generation that attempts to learn the shape and spatial information of patterns by the Transformer. However, there is a representation gap when converting coordinates and edge values to language token embedding. In addition, it ignores the position of polygonal patterns. Other methods represent a layout clip by using a topological matrix and two vectors [3], [4]. This representation is not flexible for existing deep neural networks [9]. To the best of our knowledge, the

*Corresponding Author: nzou@nju.edu.cn.
This work was supported by the National Natural Science Foundation of China (62341408).

shape (e.g., height, width, orientation) of individual polygons, along with the topological relationships (e.g., edge2edge, side2side) among multiple polygons, are critical factors for identifying hotspots [4]. Thus, the aforementioned supervised learning methods and layout representation struggle to construct robust statistical models for hotspot detection.

Considering the existing issues, we design a novel contrastive learning framework on layout sequences and images for hotpot detection, CLI-HD, which effectively distinguishes between hotspots and non-hotspots through similarity computations while incorporating geometric representations and topological information. The main contributions of this paper are summarized as follows:

- We propose a novel contrastive learning framework for hotspot detection, that leverages similarity computations, thereby eliminating the need to establish a unified decision boundary across the entire dataset.
- We present Layout2Seq, a novel sequence representation for layouts that efficiently converts polygons into vector formats, facilitating the extraction of geometric features. This approach can also provide the possibility for other machine learning-based DFM applications.
- We develop the Absolute Position Embedding (APE) method for Transformer-based sequence models, specifically designed to efficiently encode the topology relationships of multiple polygons based on their position context within the layout.
- Experimental results demonstrate that CLI-HD significantly improves the discrimination of hotspots through contrastive methods, achieving an accuracy improvement of 0.82% to 4.77% and a reduction in false alarm rates ranging from 4.9% to 23.18%. Furthermore, CLI-HD can be effectively fine-tuned with a single linear layer, yielding remarkable performance in hotspot detection.

II. Preliminaries

The lithographic process utilizes a mask to transfer a precisely designed layout pattern onto the wafer. Sensitive patterns are particularly susceptible to yield reduction caused by variations in the manufacturing process. These patterns are defined as hotspots within the layout [5]. In our paper, hotspot detection is considered as a binary classification task for the layout clips. Different from [6], [8], [10], we are dealing with layout clips where the hotspot locations are random and may include multiple hotspots, rather than just a single centered marker. It will test the model's ability to learn non-fixed hotspot patterns making it more relevant to real-world applications relevant to real-world application scenarios. The following definitions and metrics are used to evaluate the performance of a hotspot detector.

Definition 1 (Accuracy). The ratio of correctly identified hotspots to the total number of ground truth hotspots.

Definition 2 (False Alarm). The ratio of non-hotspot clips that are detected as hotspots by the classifier.

To match the evaluation metrics above, we formulate the hotspot detection problem as follows.

Problem 1 (Hotspot Detection). Given a collection of layout clips containing hotspot and non-hotspot patterns, our goal is to learn a feature vector from building image representations and sequence representations. Based on this vector we can use contrastive learning to distinguish hotspots or fine-tune a classifier for detecting hotspots.

III. Methodology

A. Overview

As shown in Fig. 1(b), we initially obtain the image representation of the layout, consistent with previous work. To construct a geometric representation of polygons, Layout2Seq is proposed to transform the shape of each polygon into a vector and combine multiple polygon vectors into a sequence corresponding to a layout clip. Leveraging images and layout sequences, a contrastive learning framework is designed to generate a unified representation aligning visual features with the geometric properties of polygons. In the contrastive network, we discard the wrong position encoder in the Transformer and develop the APE for introducing the centroid location of polygons.

B. Contrastive learning framework in CLI-HD

As illustrated in Fig. 2, our proposed framework consists mainly of two key components, specifically image encoder $\mathcal{I}(\cdot)$ and sequence encoder $\mathcal{S}(\cdot)$. Motivated by the [11], our objective is to jointly train $\mathcal{I}(\cdot)$ and $\mathcal{S}(\cdot)$ using pairs of binary layout images and corresponding sequence representations, ensuring that the learned image embedding $\mathbf{F}^I = [f_1^I, f_2^I ..., f_N^I] \in \mathbb{R}^{N \times D}$ are effectively aligned with the sequence embedding $\mathbf{F}^S \in \mathbb{R}^{N \times D}$, where N denotes the batch size and D represents the embedding dimension. Basically, a ViT-based model [12] is utilized to encode binary layout images, generating their corresponding image latent representations. In our proposed, the sequence encoder $\mathcal{S}(\cdot)$ is implemented using a transformer architecture, which generates the sequence embedding $\mathbf{F}^S$ from the output of Layout2Seq. Specifically, for the given image-sequence pairs $(\mathbf{X}_i^I, \mathbf{X}_i^S)$, the embedding f_i^I and f_i^S can be computed as follows:

$$f_i^I, f_i^S = \mathrm{Proj}(\mathcal{I}(\mathbf{X}_i^I)), \mathrm{Proj}(\mathcal{S}(\mathbf{X}_i^S)) \tag{1}$$

where $\mathrm{Proj}(\cdot)$ denotes the projection layer. During training process, we compute the loss function of image-to-text $L_{I \to S}$ as:

$$L_{I \to S} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp(\mathrm{sim}(f_i^I, f_i^S)/\tau)}{\sum_{j=1}^{N} \exp(\mathrm{sim}(f_i^I, f_j^S)/\tau)} \tag{2}$$

and the text-to-image contrastive loss $L_{S \to I}$:

$$L_{S \to I} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp(\mathrm{sim}(f_i^S, f_i^I)/\tau)}{\sum_{j=1}^{N} \exp(\mathrm{sim}(f_i^S, f_j^I)/\tau)} \tag{3}$$

where the τ is the temperature parameter and $\mathrm{sim}(\cdot)$ denotes the similary function which is calculated as:

$$\mathrm{sim}(f_i^I, f_i^S) = \frac{f_i^I \cdot \left(f_i^S\right)^{\mathrm{T}}}{\left\|f_i^I\right\| \cdot \left\|f_i^S\right\|} \tag{4}$$

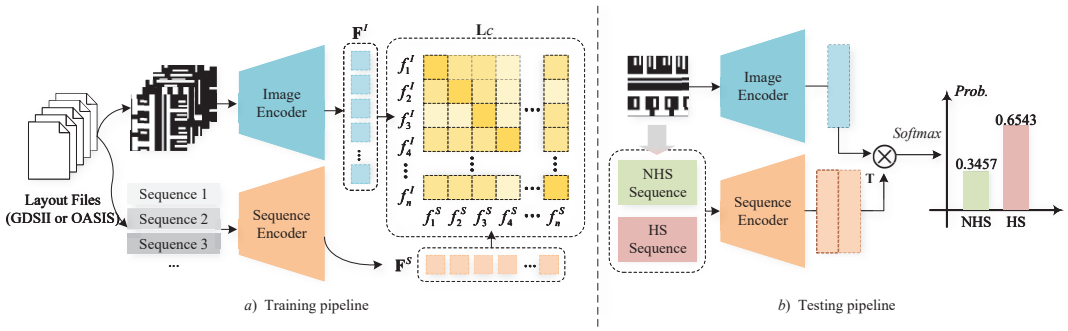


Fig. 2. The framework of contrastive learning. a) The training process of the constrastive learning for the hotspot detection. b) The simple testing process of our proposed framework.

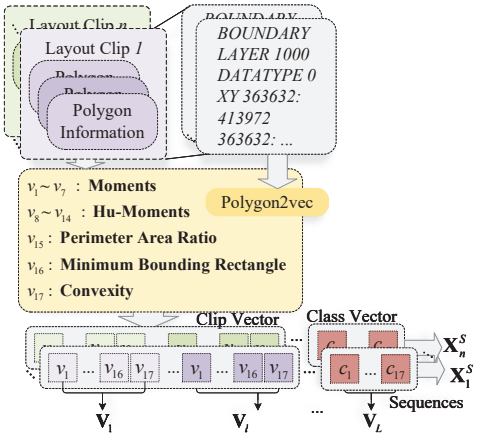


Fig. 3. Illustration of Layout2Seq.

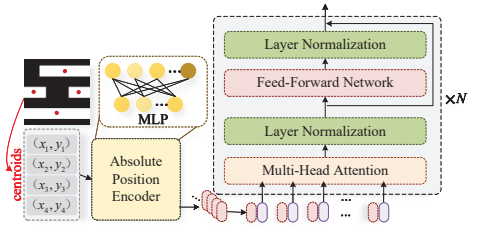


Fig. 4. Illustration of absolute position embedding in sequence encoder.

The total contrastive loss L_{con} in Fig 2(a) is calculated as:

$$L_c = \frac{L_{I \to S} + L_{S \to I}}{2} \tag{5}$$

This loss function is designed to optimize the model by increasing the similarity between matched pairs of representations, (f_i^I, f_i^S), while simultaneously decreasing the similarity between mismatched pairs, (f_i^I, f_j^S), for $i \neq j$.

As shown in Fig 2(b), given a layout image $\mathbf{X}^I$, the Layout2Seq is employed to extract both the hotspot sequence $\mathbf{X}_{hs}^S$ and the non-hotspot sequence $\mathbf{X}_{nhs}^S$ during the inference phase. These sequences, along with the image itself, are subsequently encoded via separate image and sequence encoders to generate their corresponding latent embedding, denoted as $(f^I, (f_{hs}^S, f_{nhs}^S))$. The similarity scores between the image embedding and each sequence embedding are then computed to facilitate category assignment.

Algorithm 1 Layout2Seq

Require: a set of polygons for the layout clip $\{P^k\}_{k=0}^{l}$, the polygon number of the layout clip l, the sequence context length L, the label of the layout $\hat{Y}$.
Ensure: the sequence for the i-th layout clip $\mathbf{X}_i^S$.

Initialize empty sequences $\mathbf{X}_i^S \leftarrow ()$.
for $k = 1$ to l **do**
  Initialize empty sequences $\mathbf{V}_i^k \leftarrow ()$;
  Rendering a binary image I^k by polygon P^k;
  Compute $\tilde{\mathbf{M}}_{pq}^k, \mathbf{Hu}_i^k$ by Equation(6)(7)(8)(9) and I^k;
  $\{v_1 \sim v_7\} = \{\tilde{\mathbf{M}}_{20}^k, \tilde{\mathbf{M}}_{11}^k, \tilde{\mathbf{M}}_{02}^k, \tilde{\mathbf{M}}_{30}^k, \tilde{\mathbf{M}}_{21}^k, \tilde{\mathbf{M}}_{12}^k, \tilde{\mathbf{M}}_{03}^k\}$;
  $\mathbf{V}_i^k \leftarrow \{v_1 \sim v_7\}$;
  $\{v_8 \sim v_{14}\} = \{\mathbf{Hu}_1, \mathbf{Hu}_2, \mathbf{Hu}_3, \mathbf{Hu}_4, \mathbf{Hu}_5, \mathbf{Hu}_6, \mathbf{Hu}_7\}$;
  $\mathbf{V}_i^k \leftarrow \{v_8 \sim v_{14}\}$;
  $a = \text{Area}(P^k), p = \text{Perimeter}(P^k)$;
  $w_{bbox}, h_{bbox} = \text{getBoundingRect}(P^k)$;
  $v_{15} = p/a, v_{16} = a/(w_{bbox} \times h_{bbox})$;
  $v_{17} = \text{isContourConvex}(P^k)$;
  $\mathbf{V}_i^k \leftarrow \{v_{15}, v_{16}, v_{17}\}$;
  $\mathbf{X}_i^S \leftarrow \mathbf{V}_i^k$;
end for
$l_{cur} = len(\mathbf{X}_i^S)$;
for $l = l_{cur}$ to L **do**
  if $\hat{Y} = HS$ **then**
    $\mathbf{X}_i^S \leftarrow \{1\}^{17}$;
  else if $\hat{Y} = NHS$ **then**
    $\mathbf{X}_i^S \leftarrow \{-1\}^{17}$;
  end if
end for
return $\mathbf{X}_i^S$.

C. Layout to Sequence

As shown in prior work [5], [6], converting layout files into their corresponding binary images is a straightforward task. However, to effectively capture geometric information from polygon, we propose a novel approach, called Layout2Seq.

This method transforms layout files into sequences, thereby enhancing the extraction and analysis of geometric features.

As shown in Fig. 3, we extract polygon coordinates from the original layout files and systematically parse each polygon using a vectorized polygon method. Image moments are extensively employed in computer vision and image processing to characterize object shapes [13]. Therefore, for each polygon P^k, where k is the index of the polygon, we generate a binary image I^k using the polygon's coordinates. Subsequently, the geometric moment $\mathbf{M}^k$ of the I^k is computed by:

$$\mathbf{M}_{pq}^k = \sum_{x=0}^{W} \sum_{y=0}^{H} x^p \cdot y^q \cdot I^k(x, y) \tag{6}$$

where W and H represent the width and height of I^k, p and q denote the order of the moment. We further compute the central moments $\mathbf{C^k}$ as:

$$\mathbf{C}_{pq}^k = \sum_{x=0}^{W} \sum_{y=0}^{H} (x - \frac{\mathbf{M}_{10}^k}{\mathbf{M}_{00}^k})^p \cdot (y - \frac{\mathbf{M}_{01}^k}{\mathbf{M}_{00}^k})^q \cdot I^k(x, y) \tag{7}$$

where $\mathbf{M}_{00}$, $\mathbf{M}_{10}$ and $\mathbf{M}_{01}$ can be used to compute the x-coordinate and y-coordinate of the centroid. For scale-invariant moments which are useful for comparing shapes regardless of their size, we compute the normalized moments $\tilde{\mathbf{M}}^k$ as follows:

$$\tilde{\mathbf{M}}_{pq}^k = \frac{\mathbf{C}_{pq}^k}{(\mathbf{M}_{00}^k)^{(p+q)/2+1}} \tag{8}$$

The $\{\tilde{\mathbf{M}}_{20}^k, \tilde{\mathbf{M}}_{11}^k, \tilde{\mathbf{M}}_{02}^k, \tilde{\mathbf{M}}_{30}^k, \tilde{\mathbf{M}}_{21}^k, \tilde{\mathbf{M}}_{12}^k, \tilde{\mathbf{M}}_{03}^k\}$ are incorporated into the polygon vector $\mathbf{V}^k$ as components $\{v_1 \sim v_7\}$.

It is well known that polygons in hotspot patterns exhibit geometric invariants, such as rotational invariance and other transformation properties. In our work with Layout2Seq, we utilize Hu's seven moments [13] to encapsulate the shape invariants of polygons, thereby enhancing robustness against geometric transformations. The Hu's moments can be calculated as:

$$\{\mathbf{Hu}_i\}_{i=1}^7 = \{\mathrm{f}_i(\tilde{\mathbf{M}}_{pq})\}_{i=1}^7 \tag{9}$$

where f_i is specific function or formula (non-linear combination) of the normalized moments $\tilde{\mathbf{M}}_{pq}^k$. The detailed calculations of the seven Hu moments can be found in [13]. Additionally, we have incorporated area-perimeter ratio v_{15}, minimum bounding rectangle v_{16}, and convexity v_{17} to more accurately characterize the geometric properties of polygons. Consequently, for each polygon, we derive a comprehensive vector of geometric information, denoted as $\mathbf{V}_i^k$, which includes a total of 17 values.

For the final sequence $\mathbf{X}^S \in \mathbb{R}^{L \times 17}$, where L denotes the length of the sequence, we insert an additional class vector to facilitate the similarity calculation in the contrastive learning. We assign a value of $+1$ to the category vectors for clips corresponding to the hotspot, while for all other cases, we assign a value of -1. Ultimately, l polygon vectors, along with their corresponding category vectors, collectively constitute the sequence $\mathbf{X}^S$. The complete Layout2Seq pipeline is detailed in Algorithm 1.

Algorithm 2 Training and testing for our proposed

Training process:

Require: Training pair set of layout representations $\{\mathbf{X}_i^I, \mathbf{X}_i^S\}_{i=0}^N$, where $\mathbf{X}_i^S$ is from Alogorithm 1.

Parameters: An image encoder $\mathcal{L}$, an sequence model $\mathcal{S}$.

Initialize $\mathcal{L}$, $\mathcal{S}$ randomly.
for $i = 1$ to N **do**
  $\mathbf{X}_i^{pos} \leftarrow$ Compute centroid by Equation (11);
  $\mathbf{X}_i^S = \mathbf{X}_i^{pos} + \mathbf{X}_i^S$, Absolute Position Embedding;
  Compute (f_i^I, f_i^S) by Equation (1);
  Optimize $\mathcal{L}, \mathcal{S}$ by Equation (5);
end for

Testing process:

Require: A layout clip image $\mathbf{X}^I$, $\mathcal{L}$ and $\mathcal{S}$, as returned by training process.

Ensure: the prediction result Y.

Assuming $\hat{Y} = HS$, compute the $\mathbf{X_{hs}^S}$ by Alogorithm 1;
Assuming $\hat{Y} = NHS$, compute the $\mathbf{X_{nhs}^S}$ by Alogorithm 1;
$f^I = \mathrm{Proj}(\mathcal{I}(\mathbf{X}^I))$;
$f_{hs}^S, f_{nhs}^S = \mathrm{Proj}(\mathcal{S}(\{\mathbf{X_{hs}^S}, \mathbf{X_{nhs}^S}\}))$;
$score_{hs}, score_{nhs} = \mathrm{Softmax}(\mathrm{sim}(f^I, (f_{hs}^S, f_{nhs}^S)))$;
$Y \leftarrow score_{hs}, score_{nhs}$;
return Y.

D. Absolute position embedding

The polygon-based topology is commonly associated with the definition of hotspots in design rules. We embedded the positions of the polygons into the learning process of the sequence model by the proposed APE. For a given polygon P^k, the centroid coordinates (x_k, y_k) are computed by central moments as follows:

$$x_k = \frac{\mathbf{C}_{10}^k}{\mathbf{C}_{00}^k}, y_k = \frac{\mathbf{C}_{01}^k}{\mathbf{C}_{00}^k} \tag{10}$$

As shown in Fig. 4, we exclude the position encoder typically employed in Transformer architectures, as the lack of continuous contextual relationships between polygons. Instead, we developed a straightforward multi-layer perceptron (MLP) to encode the coordinates for integration into the sequence model, yielding absolute position embedding $\mathbf{X}_i^{pos}$ as follows:

$$\mathbf{X}_i^{pos} = \{\sigma(\mathrm{FC}([\hat{x}_k, \hat{y}_k]))\}_{k=1}^l \tag{11}$$

Here, the coordinates need to be normalized as $\hat{x}_k = \frac{x_k}{W}$ and $\hat{y}_k = \frac{y_k}{H}$, W and H denotes the width and height of the input layout clip image, l denotes the number of polygon. Consistent with other approaches utilizing position embedding, we adopt summation to fuse position information into the sequence representation. Therefore, for the sequence model $\mathcal{S}(\cdot)$, the inputs are $\mathbf{X}_{i,input}^S = \mathbf{X}_i^{pos} + \mathbf{X}_i^S$.

The entire training and evaluation process for CLI-HD, which combines Layout2Seq and APE, is detailed in Alogorithm 2.

TABLE I
STATISTICS OF THE DATASET

Benchmark	Training Set		Testing Set	
	#HS	#NHS	#HS	#NHS
ICCAD2012	1204	17096	2524	13503
ICCAD2016	1300	9935	1301	2484
ICCAD2019-1	467	17758	1001	14621
ICCAD2019-2	467	17758	64310	65523

TABLE II
COMPARISON ON ICCAD-2012 BENCHMARK WITH THE STATE-OF-THE-ART METHODS.

Method	Accuracy(%)	FA(%)	Time(s)
DAC'2021 [6]	98.25	6.10	-
TODAES'2022 [14]	98.90	9.90	30.0
TCAD'2022 [10]	98.77	18.50	**3.0**
DATE'2022 [8]	98.42	12.80	3.2
DATE'2023 [5]	96.20	6.40	7.5
Ours	**99.72**	**1.20**	10

IV. EXPERIMENTAL RESULTS

Our experimental framework is implemented using PyTorch, with all models trained on 8 NVIDIA GeForce RTX A6000 GPUs (48 GB memory) for accelerated computation. We evaluate our approach on the ICCAD2012 [15], ICCAD2016 [16], and ICCAD2019 benchmarks [17]. Notably, our experiments on the ICCAD2016 dataset incorporate random cropping of image sizes, a novel augmentation technique that allows for the presence of multiple hotspots within a single image. This increases both the complexity and challenge of hotspot detection, distinguishing our methodology from prior works. Comprehensive dataset details are provided in TABLE I.

A. Comparison with State-of-the-Art Hotspot Detection Works on ICCAD2012

To demonstrate the efficacy of our proposed approach, we present the performance results on the ICCAD2012 dataset. As shown in TABLE II, a comparison is made between our method and the currently state-of-the-art techniques for hotspot detection. Our proposed method demonstrates superior accuracy compared to other state-of-the-art algorithms (98.25% of DAC'2021 [6], 98.90% of TODAES'2022 [14], 98.77% of TCAD'2022 [10], 98.42% of DATE'2022 [8] and 96.20% of DATE'2023 [5] v.s. 99.72% of ours). Moreover, the proposed method effectively addresses the problem of excessive false positives encountered by existing approaches. On the ICCAD2012 benchmark, our method reduces the false alarm rate to 1.2%. The computational efficiency of the proposed algorithm is a key consideration. Our method achieves competitive performance while maintaining an acceptable trade-off between the accuracy and speed.

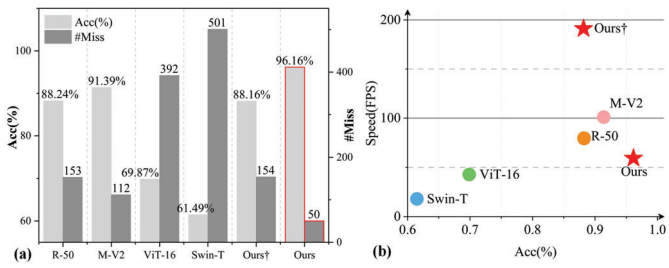


Fig. 5. Comparison with the state-of-the-art deep learning-based methods on ICCAD2016. a) Bars with solid red borders indicate optimal results. † denotes our results of supervised fine-tuning. #Miss denotes the number of false negatives. b) Comparison of model detection speed and accuracy on ICCAD2016.

TABLE III
RESULTS ON ICCAD2019-1 FOR OUR PROPOSED

Benchmark	Accuracy(%)	FA(%)
Benchmark1	76.60	29.66
Benchmark2	86.76	22.53
Benchmark3	88.63	19.45
Benchmark4	83.58	22.65
Benchmark5	93.00	20.43
Benchmark6	88.67	18.52
Benchmark7	85.75	21.36
Benchmark8	86.85	20.37
Benchmark9	93.40	9.71

B. Comparison with State-of-the-Art Deep Learning-based Classification Model on ICCAD2016

We evaluated four state-of-the-art backbone networks (ResNet50 (R-50) [18], MobileNetV2 (M-V2) [19], ViT-16 [12], and Swin-Transformer Tiny (Swin-T) [20]) using supervised learning on the ICCAD2016 datasets, and compared their performance against our proposed method. As shown in Fig. 5(a), our proposed method achieves an optimal accuracy of 96.16% with reducing the number of missed detections to only 50. Specifically, we evaluate the generalization of the proposed method by fine-tuning the learned representations under a supervised learning framework, denoted as Ours†. We attained comparable accuracy to a 50-layer ResNet (88.24% vs 88.16%) by fine-tuning contrastive learning representations with a single-layer neural network. Furthermore, as illustrated in Fig. 5(b), our fine-tuned model demonstrates significantly improved inference speed, achieving nearly 200 FPS, outperforming all comparative methods.

C. Comparison with State-of-the-Art on ICCAD2019

We perform comparative experiments using the ICCAD2019 benchmark, which consists of two subsets, ICCAD2019-1 and ICCAD2019-2, both sharing a common training set. It is considered a more accurate evaluation of the detector's capacity to detect never-before-seen hotspots. TABLE III provides detailed results for the ICCAD2019-1 dataset. TABLE IV reports the performance of our proposed method and four state-of-the-art hotspot detection algorithms, as presented in TCAD'2020 [21], TODAES'2022 [14], DATE'2023

TABLE IV
COMPARISON ON ICCAD-2019 BENCHMARK WITH THE STATE-OF-THE-ART METHODS.

Benchmark	TCAD'2020 [21]		TODAES'2022 [14]		DATE'2023 [5]		GLSVLSI'2024 [22]		Ours	
	ACC(%)	FA(%)	ACC(%)	FA(%)	ACC(%)	FA(%)	ACC(%)	FA(%)	ACC(%)	FA(%)
ICCAD2019-1	80.90	2.50	87.20	9.70	91.60	8.60	62.30	3.70	**96.10**	**1.00**
ICCAD2019-2	89.80	83.90	90.30	84.10	**90.50**	83.90	84.30	64.20	87.02	**20.53**
Average	85.30	43.20	88.75	46.90	91.05	46.25	73.30	33.95	**91.56**	**10.77**

[5], and GLSVLSI'2024 [22]. Firtly, our proposed framework achieves an average accuracy of 91.56%, outperforming TCAD'2020 [21], TODAES'2022 [14], DATE'2023 [5], and GLSVLSI'2024 [22], which report accuracies of 85.30%, 88.75%, 91.05% and 73.30%, respectively. Furthermore, the results indicate that the existing methods exhibit a significant false positive rate on the ICCAD2019-2 dataset. Our framework demonstrates a substantial advantage in effectively suppressing false alarms, achieving a rate of 20.53% on ICCAD2019-2 and an average of 10.77%.

V. CONCLUSION

In this work, we propose CLI-HD, a novel hotspot detection method that leverages the topological representation of layout patterns and a contrastive learning framework. We develop a contrastive learning framework that aligns feature representations across binary images and sequence representations. The Layout2Seq is proposed to integrates geometric information of polygons in the process of pattern representation. We introduce absolute position information to learn the topological structure. Our results show that CLI-HD surpasses previous methods, achieving a 0.82%-4.77% gain in accuracy while reducing false alarm rates by 4.9%-23.18%. Future work will explore the application of topological and contrastive learning techniques across diverse downstream tasks beyond hotspot detection.

REFERENCES

[1] W.-C. Chang and I. H.-R. Jiang, "iclaire: A fast and general layout pattern classification algorithm with clip shifting and centroid recreation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1662–1673, 2019.

[2] S. S.-E. Tseng, W.-C. Chang, I. H.-R. Jiang, J. Zhu, and J. P. Shiely, "Efficient search of layout hotspot patterns for matching sem images using multilevel pixelation," in *Optical Microlithography XXXII*, vol. 10961. SPIE, 2019, pp. 51–58.

[3] Z. Wang, Y. Shen, W. Zhao, Y. Bai, G. Chen, F. Farnia, and B. Yu, "Diffpattern: Layout pattern generation via discrete diffusion," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[4] G. Zhou, B. Korrapati, G. R. Reddy, J. Hu, Y. Chen, and D. G. Thakurta, "Patternpaint: Generating layout patterns using generative ai and inpainting techniques," *arXiv preprint arXiv:2409.01348*, 2024.

[5] Z. Chen, F. Yang, L. Shang, and X. Zeng, "Automated and agile design of layout hotspot detector via neural architecture search," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[6] Y. Xiao, M. Su, H. Yang, J. Chen, J. Yu, and B. Yu, "Low-cost lithography hotspot detection with active entropy sampling and model calibration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 907–912.

[7] A. B. Kahng, C.-H. Park, and X. Xu, "Fast dual-graph-based hotspot filtering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1635–1642, 2008.

[8] S. Sun, Y. Jiang, F. Yang, B. Yu, and X. Zeng, "Efficient hotspot detection via graph neural network," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1233–1238.

[9] L. Wen, Y. Zhu, L. Ye, G. Chen, B. Yu, J. Liu, and C. Xu, "Layoutransformer: Generating layout patterns with transformer via sequential pattern modeling," in *Proceedings of the 41st IEEE/ACM international conference on computer-aided design*, 2022, pp. 1–9.

[10] H. Geng, H. Yang, L. Zhang, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2685–2698, 2022.

[11] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[12] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[13] M.-K. Hu, "Visual pattern recognition by moment invariants," *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962.

[14] Y. Jiang, F. Yang, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via neural architecture search," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 6, pp. 1–16, 2022.

[15] J. A. Torres, "Iccad-2012 cad contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 349–350.

[16] R. O. Topaloglu, "Iccad-2016 cad contest in pattern classification for integrated circuit design space analysis and benchmark suite," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–4.

[17] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[20] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[21] Y. Jiang, F. Yang, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via binarized residual neural network ensemble," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1476–1488, 2020.

[22] C. Wang, Y. Fang, and S. Zhang, "Feature fusion based hotspot detection with r-efficientnet," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 446–451.

GauRast: Enhancing GPU Triangle Rasterizers to Accelerate 3D Gaussian Splatting

Sixu Li[1,2], Ben Keller[2], Yingyan (Celine) Lin[1], and Brucek Khailany[3]
[1]*School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA*
[2]*NVIDIA, Santa Clara, CA, USA*
[3]*NVIDIA, Austin, TX, USA*
{sli941, celine.lin}@gatech.edu, {benk, bkhailany}@nvidia.com

***Abstract*—3D intelligence leverages rich 3D features and stands as a promising frontier in AI, with 3D rendering fundamental to many downstream applications. 3D Gaussian Splatting (3DGS), an emerging high-quality 3D rendering method, requires significant computation, making real-time execution on existing GPU-equipped edge devices infeasible. Previous efforts to accelerate 3DGS rely on dedicated accelerators that require substantial integration overhead and hardware costs. This work proposes an acceleration strategy that leverages the similarities between the 3DGS pipeline and the highly optimized conventional graphics pipeline in modern GPUs. Instead of developing a dedicated accelerator, we enhance existing GPU rasterizer hardware to efficiently support 3DGS operations. Our results demonstrate a 23× increase in processing speed and a 24× reduction in energy consumption, with improvements yielding 6× faster end-to-end runtime for the original 3DGS algorithm and 4× for the latest efficiency-improved pipeline, achieving 24 FPS and 46 FPS respectively. These enhancements incur only a minimal area overhead of 0.2% relative to the entire SoC chip area, underscoring the practicality and efficiency of our approach for enabling 3DGS rendering on resource-constrained platforms.**

***Index Terms*—Hardware Architecture, Graphics Processors, 3D Gaussian Splatting**

I. Introduction

3D intelligence is set to become the next major frontier in AI by leveraging rich 3D features to enhance understanding and interaction within complex environments. As Prof. Fei-Fei Li, co-founder of ImageNet, emphasized, *"...we need spatially intelligent AI that can model the world and reason about objects, places, and interactions in 3D space and time..."* [38]. This underscores the importance of 3D intelligent applications such as autonomous driving [39], robotics [32], and augmented/virtual reality (AR/VR) [4] shown in Fig. 1. A foundational task within 3D intelligence is 3D scene rendering, which provides essential features for downstream AI tasks that require spatial information.

Recently, 3D Gaussian Splatting (3DGS) [14] has emerged as the leading method for 3D scene rendering. As summarized in Table I, unlike previous techniques like triangle meshes [33] and neural radiance fields (NeRF) [20], 3DGS maps 3D scenes onto a set of Gaussian balls, offering superior rendering quality and automated scene reconstruction. Thanks to these advantages, 3DGS has been rapidly adopted in various 3D applications, including autonomous driving [13], [30], [42], robotics [1], [12], [19], and AR/VR [18], [31], [40].



Fig. 1. Representative examples of 3D intelligent applications, including autonomous driving, robotics, and augmented/virtual reality [11], [21], [27].

TABLE I
Comparison of Rendering Methodologies

	Triangle Mesh [33]	NeRF [20]	3D Gaussian [14]
Scene Reconstruction	Manual	Automatic	Automatic
Rendering Quality	Manually Decided	High	Very High
Rendering Speed on GPU [22]	Fast	Slow	Medium

Despite its robust performance on high-powered ($\geq$ 200 W) GPUs [25], achieving real-time rendering of $\geq$ 30 frames per second (FPS), 3D Gaussian Splatting cannot achieve high framerates on edge computing platforms with GPUs, such as those with power limitations ($\leq$ 10 W) like the NVIDIA Jetson Orin NX [22]. These platforms are increasingly crucial due to the growing demand for 3D processing in mobile and embedded systems [11], [27]. Specifically, 3DGS achieves only 2-5 FPS on these platforms [22] with commonly used real-world, large-scale datasets [3], falling short of the performance requirement for most practical applications. This performance gap poses challenges for deploying advanced 3D intelligence in resource-constrained environments, highlighting the need for hardware acceleration of 3DGS.

Existing efforts to accelerate 3D Gaussian Splatting [16] and similar rendering pipelines [10], [17] typically focus on developing dedicated hardware accelerators. While these specialized units can provide performance benefits, they require exclusive hardware resources and often lead to increased system complexity and cost. Moreover, the integration of dedicated accelerators may not be feasible for all edge platforms, especially those with stringent area and power constraints. However, modern GPUs are already equipped with optimized fixed-function graphic acceleration hardware for triangle meshes. As these fixed-function units also target graphics rendering, their inherent capabilities present an opportunity to leverage



Fig. 2. Visualization of a 3D Gaussian representation: (a) Rendered RGB image depicting the scene with realistic color and detail; and (b) the corresponding Gaussian ball representation of the same scene, showing the underlying 3D structure before rendering. Both images are rendered using the 'Bonsai' scene from the NeRF-360 [3] dataset.

existing resources to accelerate 3DGS rendering without introducing significant additional overhead.

To mitigate the challenges associated with designing dedicated memory systems and software stacks for specialized accelerators [6], we propose an alternative solution: enhancing the existing graphics units—specifically, the rasterizer for triangle meshes—within GPUs to accelerate 3DGS rendering. By leveraging the capabilities of current GPU hardware, we aim to enhance performance without the need for additional dedicated accelerators. We begin with an in-depth profiling of the 3DGS rendering pipeline to identify time-consuming operations and analyze its similarities with operators in the triangle mesh rendering pipeline, which are already supported by the GPU's native graphic hardware units. This analysis reveals opportunities to adapt and enhance these units to efficiently process 3D Gaussian operations.

Building on these insights, we propose and develop an enhanced rasterizer, GauRast, for GPUs that enables the efficient execution of the dominant operator in the 3DGS rendering pipeline while preserving the original capabilities for standard triangle mesh rendering. This design ensures compatibility with existing GPU architectures and minimizes disruptions to conventional workflows. By enhancing existing hardware rather than introducing entirely new components, our approach strikes a balance between performance improvements and resource utilization. Our contributions are as follows:

- We advocate an alternative direction for accelerating 3DGS, the leading algorithm in neural rendering, by enhancing existing GPU units originally dedicated to traditional workloads.
- We detail comprehensive profiling and analysis of the 3DGS rendering pipeline [14] to identify the critical operators requiring acceleration, and demonstrate that the bottleneck operator in the 3DGS rendering pipeline shares similarities with the triangle rendering pipeline, which is already well-accelerated by existing GPU's triangle rasterizer.
- We propose an enhanced rasterizer, named GauRast, to support the dominant operator in the 3DGS rendering pipeline while maintaining its functionality for standard triangle mesh rendering tasks.
- We present a GauRast hardware prototype that achieves a 23× speedup and a 24× improvement in energy efficiency on the target SoC for the dominant operator with the original 3DGS rendering pipeline. This leads to a 6× end-to-end speedup in the original 3DGS rendering pipeline [14] and a 4× end-to-end speedup in the latest efficiency-optimized 3DGS rendering pipeline [9], achieving 24 FPS and 46 FPS, respectively. Notably, GauRast incurs an area overhead of only 21% for the enhanced graphics units, corresponding to 0.2% of the total SoC area.

Taken together, our proposals demonstrate a promising approach to enabling real-time 3DGS rendering on future edge devices.

II. 3DGS and Its Rendering Bottleneck

A. 3D Gaussian Splatting Algorithm

3D Gaussian Scene Representation. The state-of-the-art (SOTA) method for 3D scene rendering employs a 3D Gaussian-based representation [14] as illustrated in Fig. 2. This approach strikes a balance between high rendering quality and efficient rendering speed, especially on standard desktop-level GPUs. In this representation, objects are modeled as collections of elliptical 3D Gaussian balls. Each Gaussian ball is characterized by a 3D Gaussian probability density function and an associated color vector. The rendering process distributes colors over the regions covered by the Gaussians' probability densities, collectively forming the complete scene.

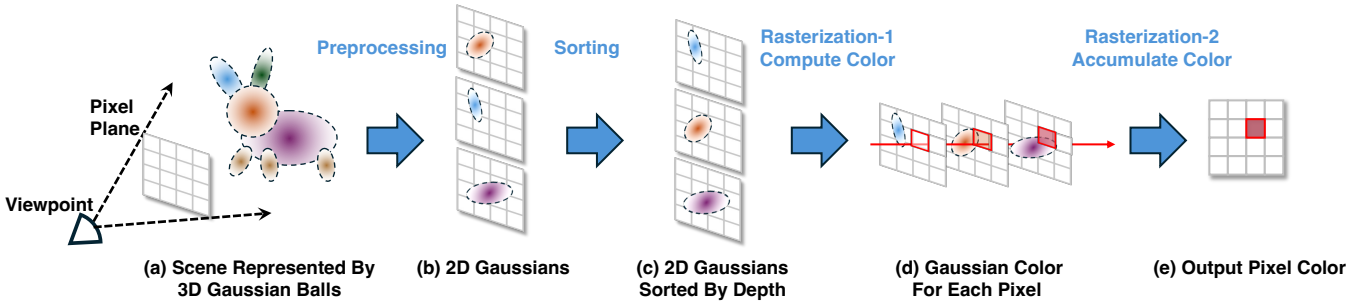


Fig. 3. Overview of the 3DGS pipeline [14]: (a) *Scene representation*: The scene is depicted as 3D Gaussian balls, viewed from a specific viewpoint and projected onto a 2D pixel plane. (b) *Preprocessing*: These 3D Gaussians are projected onto the 2D plane, resulting in 2D Gaussian representations. (c) *Sorting*: 2D Gaussians are ordered by depth to ensure the correct rendering sequence and handle occlusion properly. (d) *Initial rasterization*: Colors for each pixel are calculated based on the Gaussians covering that pixel. (e) *Color accumulation*: Colors are accumulated to produce the final pixel color output.

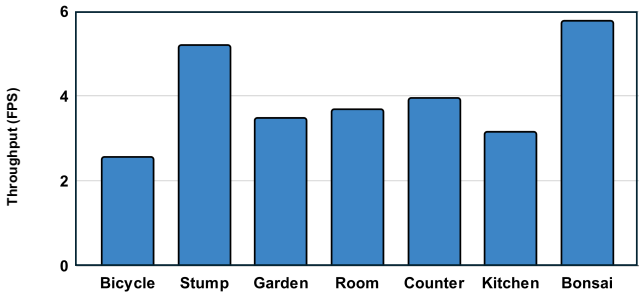


Fig. 4. Throughput achieved by the 3DGS rendering pipeline [14] across all seven scenes from the large-scale, real-world NeRF-360 dataset [3], as measured on the NVIDIA Jetson Orin NX [22] with a 10W power limit.

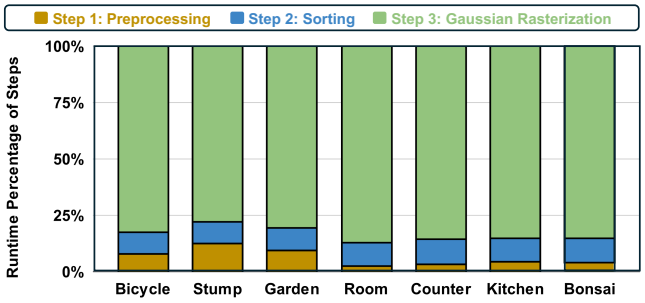


Fig. 5. Runtime breakdown of the 3DGS rendering pipeline [14] across all seven scenes from the large-scale, real-world NeRF-360 dataset [3] as measured on the NVIDIA Jetson Orin NX [22] with a power limit of 10W.

Rendering an image from a specific viewpoint using this representation (described in Fig. 3) involves three main steps:

Step 1: Preprocessing. The first stage of 3DGS rendering, depicted in Fig. 3(a), is preprocessing, which serves three purposes: projecting all 3D Gaussians onto a 2D plane according to the specified viewing position and angle, converting the color vector of each Gaussian to an RGB representation based on the viewing parameters, and computing the depth of each 3D Gaussian relative to the given viewpoint. This step transforms the 3D representation into a set of 2D Gaussians, each characterized by an opacity function o, an assigned RGB color c, a depth value d, a covariance matrix Σ representing the 2D Gaussian probability distribution, and a center point μ.

Step 2: Sorting. The second stage, as illustrated in Fig. 3(b), involves sorting the projected 2D Gaussians. Due to the potential overlap of Gaussians, each pixel may be influenced by multiple Gaussians. The rendering order impacts their visibility since Gaussians rendered earlier can obscure those rendered later. To maintain correct occlusion relationships and ensure visual consistency, the 2D Gaussians are sorted by their depth values. This depth-based sorting ensures that Gaussians nearer to the viewing position are rendered first, preserving proper occlusion handling.

Step 3: Gaussian Rasterization. The final step, Gaussian rasterization, applies the color of each Gaussian to the pixels it covers, based on opacity and Gaussian probability, and follows the depth order determined in Step 2. As shown in Fig. 3(c), for each Gaussian applied to a specific pixel, the density $\alpha_{P,i}$ is computed using the formula:

$$\alpha_{P,i} = o_i e^{-\frac{1}{2}(P-\mu_i)^T {\Sigma_i}^{-1}(P-\mu_i)}$$

where P represents the pixel coordinate, i is the index of the Gaussian, and o_i is the opacity of the Gaussian. This density reflects the contribution of the Gaussian to the pixel. Once the density $\alpha_{P,i}$ is calculated, the RGB color contribution for that pixel is accumulated as follows, as shown in Fig. 3(d):

$$\mathbb{C}_P = \sum_{i=1}^{n} T_{P,i}\alpha_{P,i}c_i$$

The term $T_{P,i} = \prod_{j=1}^{i-1}(1-\alpha_{P,j})$ represents the accumulated density of all previously applied Gaussians at the pixel, accounting for the occlusion effects from preceding Gaussians. This ensures that only visible portions of overlapping Gaussians contribute to the final rendered color in Fig. 3(e).

B. Locating the Bottleneck of 3DGS Rendering

To identify bottlenecks in rendering 3D Gaussians on edge SoCs, we conducted detailed profiling to analyze the runtime distribution across various rendering steps. Profiling was performed on the widely used 3D rendering dataset NeRF-360 [3], which includes seven real-world multi-object scenes. Kernel runtimes for the different rendering steps were measured using NVIDIA Nsight Systems [28] on the NVIDIA Jetson Orin NX [22] operating under a 10W power limit typical for edge devices [8], [41]. The average runtime for rendering all viewpoints in each scene is summarized in Fig. 4. The results indicate that the edge SoC achieves only 2-5 FPS, necessitating significant speedups before it can be used in real-world applications. The detailed runtime breakdown for each rendering step is shown in Fig. 5. Step 3, the Gaussian rasterization stage, dominates the overall runtime, accounting for over 80% of the total rendering time across all scenes, thereby representing the primary performance bottleneck.

III. Leveraging the GPU Rasterizer for 3DGS

A. Opportunities in GPU's Rasterizer

The computational demands of dominant Gaussian rasterization in the 3DGS rendering pipeline have motivated various acceleration efforts [16]. Current research primarily emphasizes the development of dedicated hardware accelerators for emerging 3D rendering pipelines such as 3DGS [16] and NeRF [10], [17]. Although these accelerators offer performance improvements, they can introduce significant trade-offs, including increased complexity in memory systems and software stacks [6] and the need for exclusive hardware resources. Unlike these dedicated approaches, our strategy focuses on enhancing existing GPU hardware to leverage the built-in capabilities of modern GPUs. This approach enhances 3DGS performance while preserving compatibility with conventional GPU operations and the original programming interface, minimizing disruptions to established workflows and fully utilizing the GPU's existing data path.

Modern GPUs have dedicated units optimized for triangle mesh rasterization [23]. This process involves iterating over

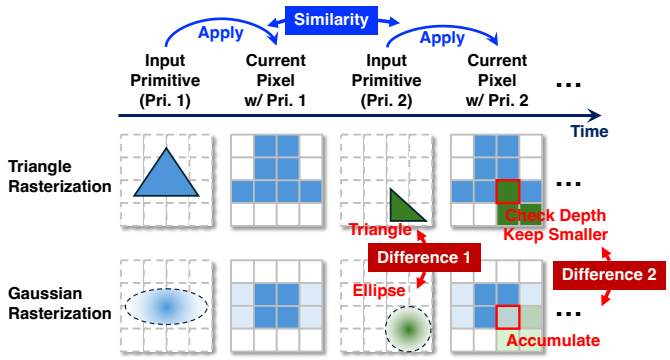


Fig. 6. Comparison of triangle and Gaussian rasterization. Both techniques sequentially apply primitives to pixels with key differences in primitive type (Difference 1) and reduction algorithm (Difference 2).

primitives (here referring to triangles) and mapping their parameters to display pixels. The pipeline is highly parallelized, relying on fixed-function accelerators for rapid computation [2], [5], [37]. Our approach begins with an investigation of the similarities and differences between Gaussian rasterization and triangle rasterization, the latter of which is efficiently supported by these optimized GPU hardware units.

B. Contrasting Gaussian and Triangle Rasterization Methods

3DGS rasterization shares key foundational elements with traditional triangle mesh rasterization. As illustrated in Fig. 6, both processes iterate over primitives—triangles in conventional rendering and Gaussians in 3DGS—and map their respective parameters onto pixels. This similarity in the basic dataflow and computation pattern provides an opportunity to adapt existing GPU rasterizers to support 3D Gaussian operations.

However, there are also differences between the two rasterization processes. First, in triangle rasterization, the algorithm involves determining if a pixel lies within a triangular boundary. In contrast, 3DGS requires calculating Gaussian probability distributions to assess the coverage. This distinction necessitates modifications to the detection function to accommodate elliptical shapes rather than triangles. Second, for triangle mesh rasterization, the reduction method relies on a minimum-depth selection to determine which primitive's parameters should be applied to each pixel, whereas 3D Gaussian Splatting aggregates color contributions from multiple overlapping Gaussians. This approach increases the complexity of the reduction function, as overlapping Gaussians contribute to pixel color based on their calculated densities and occlusion effects. Effectively addressing these differences is essential to adapting current rasterization hardware for 3DGS, ensuring compatibility and preserving performance.

C. Enhancing the Triangle Rasterizer for 3DGS

Building on our identified similarities and differences between Gaussian and triangle rasterization, we propose enhancing the existing triangle rasterizer to support 3DGS. A comparison of the resource requirements for triangle and Gaussian rasterization is summarized in Table II. As shown in the table, both rasterization processes have identical input

TABLE II
COMPUTATIONAL PRIMITIVES FOR RASTERIZATION

Subtask (Operator)	**Triangle Rasterization [33]**	**Gaussian Rasterization [14]**
Input	Vertices' Coordinates (9 FP Numbers)	Σ, o, μ, c (9 FP Numbers)
1	Coordinate Shift (ADD, MUL)	Coordinate Shift (ADD, MUL)
2	Intersection Detection (ADD, MUL, DIV)	Gaussian Probability Computation (ADD, MUL, EXP)
3	UV Weight Computation (ADD, MUL)	Color Weight Computation (ADD, MUL)
4	Min-Depth Color Hold (ADD, MUL)	Color Accumulation (ADD, MUL)
Output	UV Weight, Depth (3 FP Numbers)	Accumulated Color (3 FP Numbers)

and output parameter sizes and follow a similar procedure: initializing pixel storage and then applying primitives to each pixel. Owing to this shared I/O width and access pattern, the existing memory interface can be directly reused from the existing triangle rasterizer.

Regarding computational resources, as highlighted in Table II, both processes primarily require multipliers and adders for their core tasks. This similarity allows us to introduce a reconfigurable datapath capable of supporting both triangle and Gaussian primitives with the same hardware resources. However, each primitive type has specific resource requirements: triangle rasterization requires a divider, while Gaussian rasterization necessitates an exponentiation unit. To address these specialized needs, we propose adding dedicated hardware units for these distinct operations, enabling seamless support for both types of rasterization.

IV. PROPOSED GAURAST HARDWARE

A. Overview of the GPU architecture with GauRast

Building on the analysis in Section III-C, we present the hardware design of GauRast, an enhanced rasterizer that extends existing hardware capabilities to efficiently support 3DGS. Unlike previous approaches that rely on dedicated accelerators, GauRast utilizes the built-in GPU rasterization hardware, specifically the triangle rasterizer, to execute 3DGS rendering tasks. This approach ensures seamless integration with existing GPU workflows, maintaining compatibility with conventional rendering while introducing minimal overhead.

As shown on the left side of Fig. 7(a), the GauRast hardware builds upon the Graphics Processing Cluster (GPC) of modern GPU architectures [24], [26]. Each GPC contains multiple Streaming Multiprocessors (SMs) for general-purpose computations and specialized fixed-function units for graphics processing. Within this architecture, we implemented an enhanced rasterizer capable of handling both triangle and Gaussian rasterization, extending the functionality of the original rasterizer that only supports triangle rasterization. As shown in Fig. 7(a), our modifications are confined to the rasterizer within the GPC, leaving the SMs untouched and preserving the overall GPU structure. The enhanced rasterizer is optimized

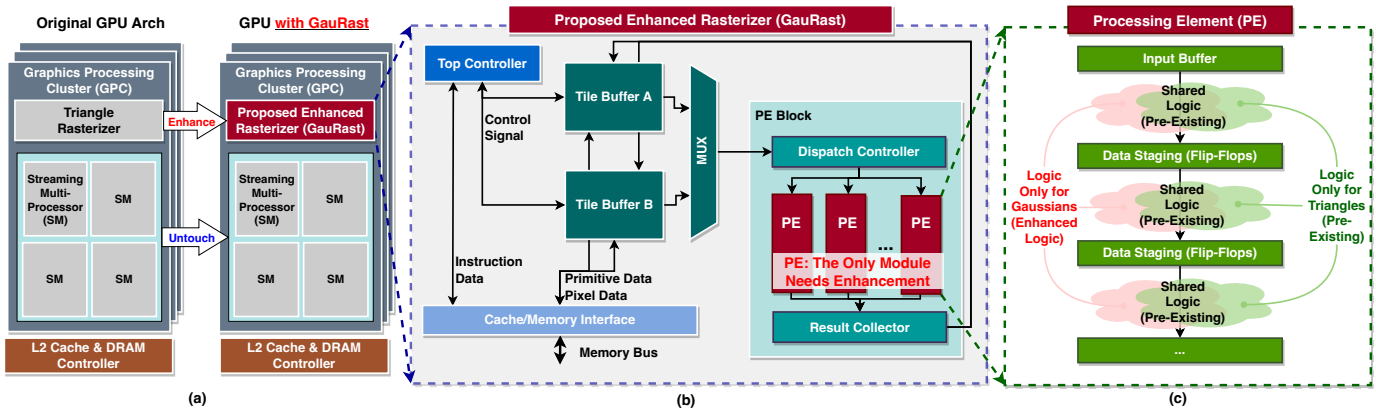


Fig. 7. Overview of the proposed hardware architecture: (a) Block diagram of the GPU's Graphics Processing Cluster (GPC) with proposed enhancements. (b) Detailed view of the enhanced rasterizer. (c) Internal structure of each PE, displaying dedicated and shared logic paths for Gaussian and triangle primitives.

to process Gaussian splatting, incorporating new logic paths alongside the existing hardware inside of each Processing Element (PE) as shown in Fig. 7(c), thus enabling seamless switching between traditional triangle rendering and Gaussian rasterization.

B. Proposed GauRast Hardware

The enhanced rasterizer consists of several key components, as shown in Fig. 7(b):

Tile Buffers (A and B). These ping-pong buffers store primitives (either Gaussians or triangles) and pixel data, enabling efficient data management during the rendering process. By alternating between Tile Buffer A and Tile Buffer B, GauRast minimizes memory latency and allows concurrent data access for both Gaussian and triangle primitives.

PE Block. The core computation of the rasterization process is handled by the PE Block, which contains multiple PEs. Each PE is tasked with computations required for either Gaussian or triangle primitives. The PE Block operates with a high degree of parallelism, leveraging the intrinsic parallel structure of the rendering task for acceleration.

Processing Element (PE). Each PE supports both Gaussian and triangle rasterization and is equipped with a combination of shared and dedicated logic units, as shown in Fig. 7(c). The shared logic handles common operations, while dedicated logic paths manage primitive-specific computations, such as evaluating Gaussian probability distributions for 3DGS rendering and performing depth division for triangle meshes. Importantly, the existing triangle rasterizer includes both shared and triangle-specific logic; thus, only Gaussian-specific logic is added. Compared to a standard triangle rasterizer, our design requires minimal additional hardware in each PE: two adders, one multiplier, and one exponentiation unit for Gaussian support, while reusing nine adders and nine multipliers from the triangle rasterizer. The PE is the only module requiring additional logic to facilitate 3DGS rendering.

The GauRast hardware optimizes performance and resource utilization by reusing existing memory interfaces, such as Tile Buffers and Controllers, and incorporates minimal additional logic for 3DGS tasks. This design ensures that the enhanced rasterizer remains compatible with the existing triangle rendering pipeline, thus minimizing hardware complexity.

C. CUDA-Collaborative Scheduling

In triangle mesh rasterization, CUDA cores and the hardware rasterizer collaborate to complete the rendering process. Specifically, non-dominant tasks such as 3D-to-2D projection and color querying are handled by the CUDA cores, while the rasterization is managed by the hardware rasterizer [23]. Similarly, we adopt a hybrid scheduling strategy to enhance efficiency, assigning non-dominant operations like preprocessing and sorting to the CUDA cores, while the enhanced rasterizer handles the dominant rasterization workload which is the primary bottleneck in 3DGS rendering.

V. Evaluation

A. Evaluation Setup

Dataset & Baselines. To evaluate the processing speedup and efficiency improvement achieved by our proposed GauRast, we conducted experiments using the widely adopted NeRF-360 dataset [3], a large-scale, real-world dataset using both the original 3DGS algorithm [14] and its latest efficiency-optimized variant [9]. For baselines, we consider both the NVIDIA Jetson Orin NX SoC [22], a representative edge GPU, and the only previously published accelerator proposal for 3DGS, GScore [16].

Hardware Implementation. We implemented a prototype of the enhanced rasterizer with 16 PEs using C++ and synthesized the design into RTL using Siemens Catapult [34]. This RTL was then synthesized, placed, and routed using Synopsys Fusion Compiler [36], targeting a 28 nm CMOS technology node (typical corner, 0.9 V, 1 GHz clock), following the methodology described in [15]. Fig. 8 shows the layout and area breakdown of the prototype. The typical power consumption of the prototype is 1.7 W as reported by Synopsys PrimePower [35] based on post-layout netlists and RTL simulation activities using randomly sampled test images from the target dataset. To ensure correct implementation, we validated the functional accuracy of both triangle and Gaussian rasterization against the software implementations, confirming that the RTL implementation's rendering output

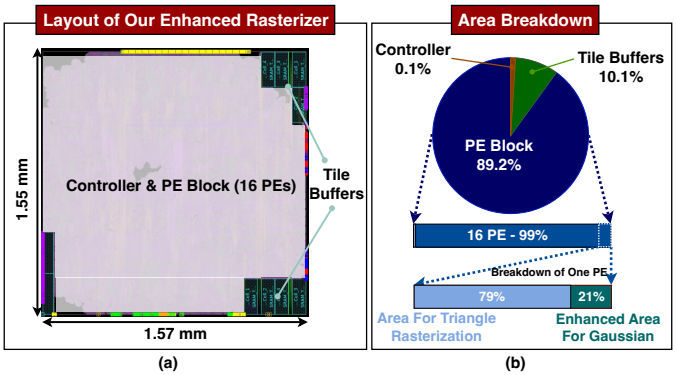


Fig. 8. Layout and area breakdown of the proposed enhanced rasterizer.

for both triangle rasterization [7] and 3DGS rendering [14] matches perfectly without any loss in rendering quality.

Simulator Setup. To match the effective area of the triangle rasterizer units in the baseline SoC [22], we scaled up our GauRast design to include 15 instances of the 16-PE rasterizer module, totaling 300 PEs. Consequently, the enhanced portion of the hardware occupies approximately 0.2% of the total area of the baseline Jetson Orin NX SoC [22]. We further developed a cycle-accurate simulator for fast evaluation of this scaled-up design. The simulator's runtime and power outputs were validated against the aforementioned RTL simulation results to ensure accuracy and reliability.

B. Comparison Against the Baseline SoC

Fig. 9 illustrates the speedup and efficiency improvements achieved by GauRast in Gaussian rasterization compared to CUDA implementations across all scenes in the benchmark dataset, for both the original algorithm [14] and its latest efficiency-optimized version [9]. The results show that our enhanced rasterizer achieves an average runtime reduction of 23× and an average energy efficiency improvement of 24× for the original algorithm. For the efficiency-optimized algorithm, it achieves a 20× runtime reduction and a 22× improvement in energy efficiency. Additionally, as shown in Fig. 10, replacing CUDA-based rasterization with our proposed enhanced rasterizer leads to substantial end-to-end performance improvements. GauRast achieves a 6× reduction in runtime for the original 3D Gaussian Splatting algorithm and a 4× reduction for the optimized algorithm, yielding average frame rates of 24 FPS and 46 FPS, respectively.

C. Comparison Against SOTA 3DGS Accelerator

SOTA work GSCore [16] on 3DGS rendering acceleration achieved a 20× speedup on an edge SoC [29] for Gaussian rasterization, utilizing a dedicated area of $3.95\,\mathrm{mm}^2$ with FP16 precision. If we re-implement GauRast to perform FP16 operations, our design achieves equivalent performance to GSCore but requires only $0.16\,\mathrm{mm}^2$ area, achieving a 24.7× improvement in area efficiency. This efficiency gain is attributed to the reuse of existing resources built in the triangle rasterizer. These results highlight the effectiveness of our enhancement approach in efficiently leveraging existing GPU components.

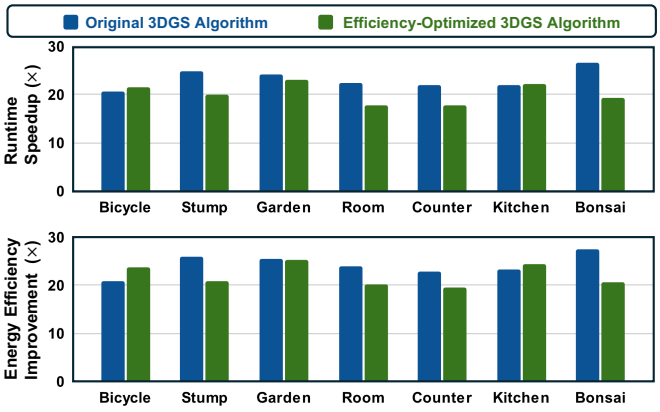


Fig. 9. Speedup and energy efficiency improvements achieved by GauRast in Gaussian rasterization, compared to CUDA implementations on the NVIDIA Jetson Orin NX [22]. Evaluations were conducted using both the original 3DGS algorithm [14] and its latest efficiency-optimized version [9], across all seven scenes of the large-scale, real-world NeRF-360 dataset [3].

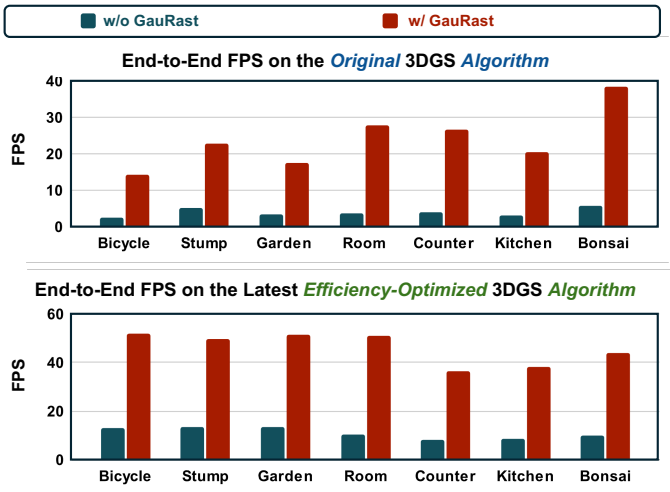


Fig. 10. End-to-end FPS comparison between the baseline SoC [22] and the baseline SoC enhanced with GauRast, using both the original 3DGS [14] and its latest efficiency-optimized version [9]. The evaluation was conducted across all seven scenes from the large-scale, real-world NeRF-360 dataset [3].

VI. Conclusion

This work presents GauRast, an enhancement for existing GPU rasterizers designed to accelerate 3DGS rendering. Experiment results show that our enhanced rasterizer achieves a 6× speedup in end-to-end runtime for the original 3DGS and a 4× speedup for the latest efficiency-optimized pipeline on the NVIDIA Jetson Orin NX, with an area overhead of only 0.2%, demonstrating a promising path forward to enable real-time 3DGS rendering on edge devices.

VII. Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable questions and constructive feedback. This work is supported by an internship at NVIDIA Research, the National Science Foundation (NSF) Computing and Communication Foundations (CCF) program (Award ID: 2312758), and the Department of Health and Human Services Advanced Research Projects Agency for Health (ARPA-H) under Award Number AY1AX000003.

REFERENCES

[1] J. Abou-Chakra, K. Rana, F. Dayoub, and N. Suenderhauf, "Physically Embodied Gaussian Splatting: A Visually Learnt and Physically Grounded 3D Representation for Robotics," in *8th Annual Conference on Robot Learning*, 2024.

[2] H.-J. Ackermann, "Single chip hardware support for rasterization and texture mapping," *Computers & Graphics*, vol. 20, no. 4, pp. 503–514, 1996.

[3] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 5470–5479.

[4] M. Q. Blog, "Tackling telepresence: 'spatial' delivers collaborative computing on oculus quest," https://www.meta.com/nl-nl/blog/quest/tackling-telepresence-spatial-delivers-collaborative-computing-on-oculus-quest, 2020.

[5] C.-H. Chen and C.-Y. Lee, "Reduce the memory bandwidth of 3D graphics hardware with a novel rasterizer," *Journal of Circuits, Systems, and Computers*, vol. 11, no. 04, pp. 377–391, 2002.

[6] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[7] Dmitry V. Sokolov, "TinyRenderer," 2022, https://github.com/ssloy/tinyrenderer, accessed 2022-05-20.

[8] M. Elgamal, D. Carmean, E. Ansari, O. Zed, R. Peri, S. Manne, U. Gupta, G.-Y. Wei, D. Brooks, G. Hills, and C.-J. Wu, "Carbon-efficient design optimization for computing systems," in *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, ser. HotCarbon '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3604930.3605712

[9] G. Fang and B. Wang, "Mini-splatting: Representing scenes with a constrained number of gaussians," in *European Conference on Computer Vision*, 2024.

[10] Y. Feng, Z. Liu, J. Leng, M. Guo, and Y. Zhu, "Cicero: Addressing Algorithmic and Architectural Bottlenecks in Neural Rendering by Radiance Warping and Memory Optimizations," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1293–1308.

[11] D. Graham-SmithMar, "What is the Metaverse?" https://www.techfinitive.com/explainers/what-is-the-metaverse/, (Accessed on 11/02/2024).

[12] J. Hu, X. Chen, B. Feng, G. Li, L. Yang, H. Bao, G. Zhang, and Z. Cui, "Cg-slam: Efficient dense rgb-d slam in a consistent uncertainty-aware 3D gaussian field," in *European Conference on Computer Vision*. Springer, 2025, pp. 93–112.

[13] N. Huang, X. Wei, W. Zheng, P. An, M. Lu, W. Zhan, M. Tomizuka, K. Keutzer, and S. Zhang, "S3Gaussian: Self-Supervised Street Gaussians for Autonomous Driving," *arXiv preprint arXiv:2405.20323*, 2024.

[14] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering." *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139–1, 2023.

[15] B. Khailany, E. Khmer, R. Venkatesan, J. Clemons, J. S. Emer, M. Fojtik, A. Klinefelter, M. Pellauer, N. Pinckney, Y. S. Shao *et al.*, "A modular digital vlsi flow for high-productivity soc design," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[16] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "GSCore: Efficient Radiance Field Rendering via Architectural Support for 3D Gaussian Splatting," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 497–511.

[17] S. Li, C. Li, W. Zhu, B. Yu, Y. Zhao, C. Wan, H. You, H. Shi, and Y. Lin, "Instant-3D: Instant neural radiance field training towards on-device ar/vr 3D reconstruction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[18] S. Ma, Y. Weng, T. Shao, and K. Zhou, "3D gaussian blendshapes for head avatar animation," in *ACM SIGGRAPH 2024 Conference Papers*, 2024, pp. 1–10.

[19] H. Matsuki, R. Murai, P. H. Kelly, and A. J. Davison, "Gaussian splatting slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 18 039–18 048.

[20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[21] NVIDIA, "Isaac Sim - Robotics Simulation and Synthetic Data — NVIDIA Developer," https://developer.nvidia.com/isaac, (Accessed on 11/02/2024).

[22] ——, "Jetson Orin for Next-Gen Robotics — NVIDIA," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, (Accessed on 04/02/2024).

[23] ——, "Life of a triangle - NVIDIA's logical pipeline," https://developer.nvidia.com/content/life-triangle-nvidias-logical-pipeline, (Accessed on 11/02/2024).

[24] ——, "NVIDIA AMPERE GA102 GPU ARCHITECTURE," https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf, (Accessed on 11/02/2024).

[25] ——, "NVIDIA RTX A6000 For Powerful Visual Computing — NVIDIA," https://www.nvidia.com/en-us/design-visualization/rtx-a6000/, (Accessed on 11/02/2024).

[26] ——, "NVIDIA TESLA V100 GPU ARCHITECTURE," https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, (Accessed on 11/02/2024).

[27] ——, "Recreate High-Fidelity Digital Twins with Neural Kernel Surface Reconstruction," https://developer.nvidia.com/blog/recreate-high-fidelity-digital-twins-with-neural-kernel-surface-reconstruction/, (Accessed on 11/02/2024).

[28] NVIDIA, "NVIDIA Nsight Systems," 2024, https://developer.nvidia.com/nsight-systems, accessed 2024-05-20.

[29] NVIDIA Inc., "Jeston Xavier NX Series Modules," 2022, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/, accessed 2022-06-01.

[30] Z. Qi, J. Ma, J. Xu, Z. Zhou, L. Cheng, and G. Xiong, "GSPR: Multi-modal Place Recognition Using 3D Gaussian Splatting for Autonomous Driving," *arXiv preprint arXiv:2410.00299*, 2024.

[31] Z. Qian, S. Wang, M. Mihajlovic, A. Geiger, and S. Tang, "3dgs-avatar: Animatable avatars via deformable 3D gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5020–5030.

[32] W. Shen, G. Yang, A. Yu, J. Wong, L. P. Kaelbling, and P. Isola, "Distilled Feature Fields Enable Few-Shot Language-Guided Manipulation," in *7th Annual Conference on Robot Learning*, 2023.

[33] P. Shirley, M. Ashikhmin, and S. Marschner, *Fundamentals of Computer Graphics*. AK Peters/CRC Press, 2009.

[34] Siemens, "Catapult High-Level Synthesis and Verification," https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/, (Accessed on 11/02/2024).

[35] synopsys, "PrimePower," 2022, https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html, accessed 2022-05-20.

[36] Synopsys, "Fusion Compiler: RTL-to-GDSII Design Solution," 2024, https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html, accessed 2024-05-20.

[37] X. Wang, F. Guo, and M. Zhu, "A more efficient triangle rasterization algorithm implemented in FPGA," in *2012 International Conference on Audio, Language and Image Processing*. IEEE, 2012, pp. 1108–1113.

[38] World Labs, "Hello, World Labs," 2024, https://www.worldlabs.ai/about, accessed 2024-11-01.

[39] Z. Xie, J. Zhang, W. Li, F. Zhang, and L. Zhang, "S-NeRF: Neural Radiance Fields for Street Views," in *The Eleventh International Conference on Learning Representations*, 2023.

[40] Y. Yuan, X. Li, Y. Huang, S. De Mello, K. Nagano, J. Kautz, and U. Iqbal, "Gavatar: Animatable 3D gaussian avatars with implicit mesh learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 896–905.

[41] Y. Zhang, R. Wang, Y. Huo, W. Hua, and H. Bao, "Powernet: Learning-based real-time power-budget rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 10, pp. 3486–3498, 2022.

[42] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "Driv-inggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21 634–21 643.

SeDA: Secure and Efficient DNN Accelerators with Hardware/Software Synergy

Wei Xuan[†‡], Zhongrui Wang[⋆], Lang Feng[§], Ning Lin[¶], Zihao Xuan[†‡], Rongliang Fu[‖], Tsung-Yi Ho[‖], Yuzhong Jiao[†‡], Luhong Liang[†‡]

[†]*ACCESS – AI Chip Center for Emerging Smart Systems, InnoHK Centers, Hong Kong Science Park, Hong Kong, China*

[‡]*The Hong Kong University of Science and Technology, Hong Kong, China*

[⋆]*Southern University of Science and Technology, Shenzhen, Guangdong, China*

[§]*Sun Yat-sen University Shenzhen Campus, Shenzhen, Guangdong, China*

[¶]*The University of Hong Kong, Hong Kong, China*

[‖]*The Chinese University of Hong Kong, Hong Kong, China*

Corresponding authors: wangzr@sustech.edu.cn, flang1994@outlook.com

***Abstract*—Ensuring the confidentiality and integrity of DNN accelerators is paramount across various scenarios spanning autonomous driving, healthcare, and finance. However, current security approaches typically require extensive hardware resources, and incur significant off-chip memory access overheads. This paper introduces SeDA, which utilizes 1) a *bandwidth-aware encryption mechanism* to improve hardware resource efficiency, 2) *optimal block granularity* through intra-layer and inter-layer tiling patterns, and 3) a *multi-level integrity verification mechanism* that minimizes, or even eliminates, memory access overheads. Experimental results show that SeDA decreases performance overhead by over 12% for both server and edge neural processing units (NPUs), while ensuring robust scalability.** [1]

***Index Terms*—Memory protection, secure DNN accelerators, confidentiality and integrity, deep neural networks**

I. Introduction

Securing Deep Neural Networks (DNNs) on neural processing units (NPUs) is increasingly vital for mission-critical applications in areas such as autonomous driving [1], healthcare [2], and finance [3]. The Artificial Intelligence (AI) hardware market was valued at USD 54.10 billion in 2023 and is expected to surge to USD 474.10 billion by 2030, reflecting a remarkable CAGR of 38.73% [4]. Protecting DNN accelerators by ensuring confidentiality and integrity is crucial for several reasons: *Data Confidentiality*: The sanctity of training data is non-negotiable. Protecting training data is essential to prevent unauthorized access and exploitation of private or sensitive information. *Resource Costliness*: The substantial investment in training resources necessitates stringent security protocols. Safeguarding these resources not only ensures their optimal utilization but also protects against financial losses and inefficiencies. *Vulnerability to Malicious Attacks*: Malicious intent poses a significant threat to DNN models. Protecting these models from potential tampering and attacks is critical to maintaining their integrity and functionality, safeguarding against adverse outcomes and ensuring the reliability of AI systems.

To protect traditional DNN accelerators from model theft and malicious tampering, as illustrated in Fig. 1(a) and Fig. 1(b), broad approaches [5]–[11] have been proposed to secure these accelerators, with the primary aim of minimizing off-chip memory access overhead for security metadata. They typically use the counter-mode encryption of Advanced Encryption Standard (AES-CTR) for confidentiality and the message authentication code (MAC) for integrity verification, as shown in Fig. 1(c). The counter value in AES-CTR concatenates the physical address (PA) and a ⓐ version number (VN) of a data block, with the VN stored off-chip and incremented with each write. To ensure the integrity of off-chip memory data, each data block is accompanied by a ⓑ message authentication code (MAC). Additionally, a ⓒ Merkle Tree (MT) [12] and its variants are often utilized, with the root stored on-chip to prevent replay attacks.

[1]SeDA source code: https://github.com/wayne4s/seda.git

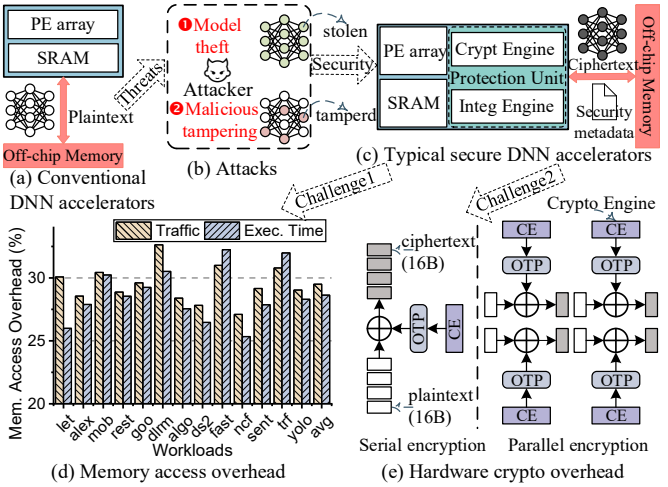


Fig. 1. Insight of typical secure accelerators. (a) Traditional DNN accelerators use untrusted off-chip memory and communication buses, risking model theft and malicious tampering, as shown in (b). (c) Secure DNN accelerators typically protect data confidentiality and integrity with memory protection schemes using AES encryption and Hash function. (d) Accessing security metadata in off-chip memory adds overhead. (e) Parallel hardware encryption incurs overhead to meet bandwidth needs of accelerators.

Motivation ❶ ▶ *Costly Off-Chip Memory Access Overhead for Integrity Verification*. Accessing security metadata (e.g., ⓐⓑⓒ) to ensure the confidentiality and integrity of untrusted off-chip memory significantly increases memory access overhead, as shown in Fig. 1(d). Existing approaches propose several optimizations: using Bonsai Merkle Tree (BMT) [13] instead of traditional MT for smaller version numbers (VNs); employing coarser-grained protection units, such as 512B data blocks instead of 64B cachelines; and dynamically updating VNs based on DNN model state information to eliminate off-chip memory access [8], [9]. Securator [11] uses a layer-level MAC to reduce off-chip memory access for integrity checks. However, it overlooks the overlap of intra-layer tiles and distinct tiling patterns across layers, potentially leading to redundant encryption/decryption and integrity verification overhead. Moreover, not considering inter-layer tiling pattern differences may result in false negatives.

Motivation ❷ ▶ *High Hardware Overhead for Confidentiality Protection*. Due to the AES Engine's limitation of en/decrypting

only a 128-bit data block at one time, typical solutions to meet the high bandwidth demands of DNN accelerators involve increasing the number of AES Engines. For example, Securator [11] uses four AES engines to en/decrypt a 64B data block, as shown in Fig. 2(c). However, this approach adds strain on resource-limited accelerators, requiring a careful balance between hardware resource allocation and security performance. As illustrated in Fig. 1(e), a single Crypto Engine can only perform serial encryption, failing to meet the high bandwidth demands of DNN accelerators, while multiple AES engines can satisfy these demands but result in significant hardware encryption overhead.

In this paper, we introduce SeDA, a novel secure and efficient DNN accelerator architecture with reduced hardware resource consumption and near-zero performance overhead for integrity verification, while maintaining the same level of security and practical applicability. This paper makes the following major contributions:

- **Insight.** Providing an in-depth insight of limitations of current memory protection strategies for DNN accelerators highlights two critical concerns: the substantial *hardware overhead for encryption* and the expensive *off-chip memory access for integrity verification*.
- **Solution.** Through hardware/software synergy, we present SeDA, a secure and efficient accelerator architecture. It incorporates a *bandwidth-aware encryption mechanism* that utilizes a single AES engine with adjustable encryption granularity, minimizing hardware resource overhead. Furthermore, its *multi-level integrity verification mechanism* significantly reduces or eliminates off-chip memory access overhead.
- **Evaluation.** Conducting extensive experiments using cycle-accurate simulators for DNN accelerators, memory protection schemes, and off-chip memory accesses. Experimental results of SeDA demonstrate a performance improvement, reducing overhead by 12.26% for server NPU and 12.29% for edge NPU, while also providing robust scalability with minimal hardware overhead to meet the bandwidth demands of accelerators.

This paper is structured as follows: Section II reviews related work on memory protection and the threat model for our study. In Section III, we provide a detailed introduction to the proposed SeDA architecture. We then conduct extensive experiments on various DNN models in Section IV. Finally, Section V concludes our work.

II. Related Work and Threat Model

Based on the corresponding threat model, secure DNN accelerators typically provides robust confidentiality and integrity guarantees in untrusted environments.

A. Confidentiality Protection

Confidentiality protection typically uses AES-CTR mode due to several advantages: 1) it utilizes the same AES engine for en/decryption, reducing hardware overhead (see Fig. 2(a)); 2) One time pad (OTP) generation can be parallelized with communication and DNN computation, minimizing time overhead; and 3) its block-based streaming mode does not require prior knowledge of data size and scale. The AES-CTR encryption requires a non-repeating and incrementing counter to produce a OTP for each encryption/decryption under the same key. The counter concatenates the physical address of a data block that will be encrypted and version number that is incremented on each write memory operation. Here, let $K_e, \mathcal{P}, \mathcal{C}$ be the AES encryption key, plaintext, ciphertext, respectively. The AES-CTR mode for encryption and decryption can be formulated as following Eq. 1 and Eq. 2, where $AES\text{-}CTR_{K_e}(PA||VN)$ produces

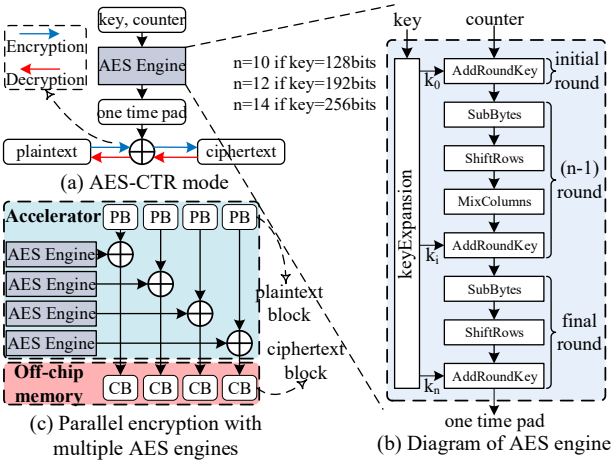


Fig. 2. Summary of AES-CTR mode. (a) Reusing AES engine for encryption and decryption in AES-CTR mode. (b) Diagram of the AES engine, featuring the AddRoundKey, SubBytes, ShiftRows, MixColumns, and keyExpansion modules. (c) Utilization of multiple AES engines for parallel encryption to boost high-bandwidth capabilities.

a OTP, $||$ and $\oplus$ represents a bit-wise concatenation and XOR operator, respectively.

$$\mathcal{C} = \mathcal{P} \oplus AES\text{-}CTR_{K_e}(PA \,||\, VN) \quad (1)$$

$$\mathcal{P} = \mathcal{C} \oplus AES\text{-}CTR_{K_e}(PA \,||\, VN) \quad (2)$$

When an accelerator reads a data block from off-chip memory, the protection module uses the VN to generate an OTP and decrypts the block by XORing it with the OTP (red path in Fig. 2(a), Eq. 2). Similarly, when writing a data block to off-chip memory, the module increments the VN to generate an OTP and encrypts the block by XORing it with the OTP (blue path in Fig. 2(a), Eq. 1). Fig. 2(b) illustrates the diagram of the AES engine along with its key functional modules.

B. Integrity Verification

To ensure data integrity against off-chip memory tampering, an integrity verification engine computes a message authentication code (MAC) by concatenating a data block with its physical address and version number. This MAC is generated during every write operation and verified during reads to ensure data authenticity. Relying solely on MAC for a data block fails to guarantee freshness and opens the door to replay attacks. To counter this threat, prior works utilize an Integrity Tree for hierarchical MAC verification, such as MT [12] and Bonsai Merkle Tree (BMT) [13], with the root stored on-chip to prevent malicious tampering. The overhead of integrity verification is non-trivial since it requires traversing both MACs and the nodes of the Integrity Tree stored in the off-chip memory to prevent the replay attack.

C. Secure DNN Accelerators

Intel SGX [5] creates a secure enclave using CPU hardware mechanisms, using AES-CTR mode and MT with its root in the Trusted Computing Base (TCB). SEAL [6] introduces a criticality-aware smart encryption scheme that selectively bypasses the encryption engine for partial data and colocates data with corresponding counters. GuardNN [7] proposes a secure accelerator architecture by using small TCB and low overhead for privacy-preserving deep learning. MGX [8] introduces a secure DNN accelerator with application-specific version number management, utilizing on-chip status for version number

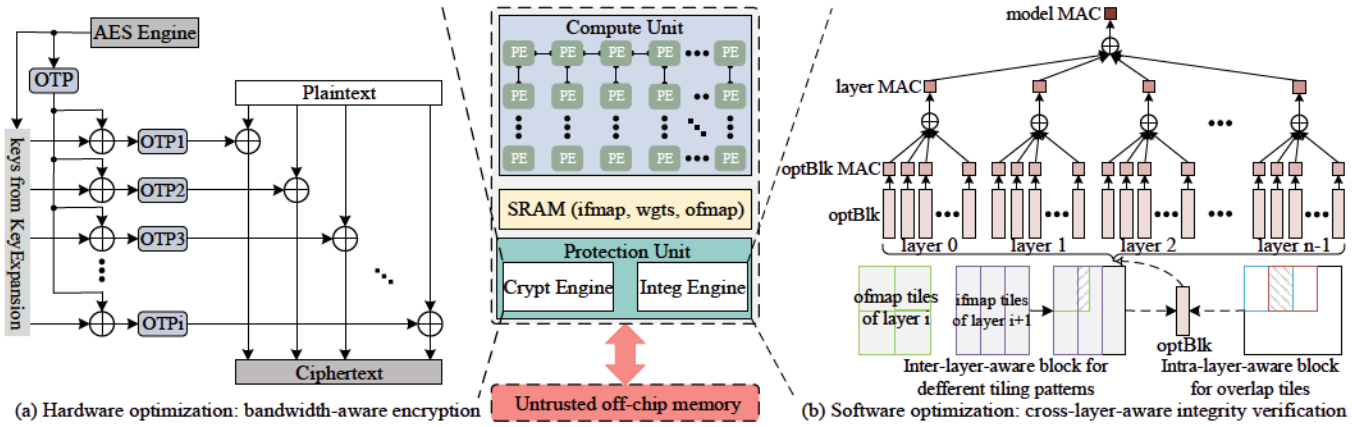


Fig. 3. Overview of SeDA architecture. (a) The Crypt Engine optimizes hardware by XORing keys from the AES Engine's KeyExpansion module with OPT, creating bandwidth-sensitive encryption granularity and reducing hardware overhead compared to using multiple AES Engines. (b) The Integ Engine optimizes software by analyzing overlapping tiles within a layer and patterns across layers to determine the optimal block size, optBlk, for integrity verification. This leads to a multi-level integrity verification mechanism with optBlk MAC, layer MAC, and model MAC.

generation and coarse-grained integrity verification to reduce off-chip memory overhead. In a similar vein, TNPU [9] generates all version numbers within an on-chip cache. Securator [11] introduces a layer-level integrity checks by XORing all message authentication codes (MACs) within a layer.

D. Threat Model

Similar to the threat model proposed in literature [8], [9], we assume that the accelerator itself is secure, meaning that its internal state cannot be directly observed or tampered with. Any external device connected to the accelerator is considered untrusted, such as off-chip memory and communication buses, meaning that attackers could potentially access or manipulate any information stored in these devices through physical or software attacks. Notably, we do not consider other physical side-channel attacks to infer DNN model architectures [14], such as power channel attacks and electromagnetic attacks. Additionally, adversarial attacks [15] against machine learning algorithms are also not within scope to ensure that decryption is restricted to authorized users possessing the key.

III. SeDA Architecture

This section initially provides an overview of SeDA. We then subsequently explores the scalability challenges of cryptographic hardware and the expensive off-chip memory access overhead in integrity verification, providing corresponding solutions for each.

A. Overview

SeDA provides a secure and efficient environment for DNN accelerators by hardware/software co-optimization. The overview architecture of the framework is presented in Fig. 3. The AES Engine's limitation to encrypting only a 128-bit data block at one time poses a challenge in meeting the high bandwidth demands of DNN accelerators. SeDA addresses these issues by utilizing bandwidth-aware encryption mechanism to optimize hardware resources impacted by security concerns, as illustrated in Fig. 3(a). Additionally, the inclusion of security metadata by secure DNN accelerators results in storage and access overheads in off-chip memory. For example, using an 8B MAC to represent a 64B data block alone leads to a 12.5% increase in memory traffic. To mitigate this, SeDA introduces a multi-level integrity verification software optimization mechanism, showcased in Fig. 3(b). Furthermore, securely housing these limited MACs directly in on-chip SRAM can eliminate off-chip memory access overheads entirely.

Algorithm 1: Attack and defense of SECA

```
Attack of SECA
Input  : OTP: one time pad for a data block (blk);
         most_value_p: most used plaintext in blk.
Output : value_p: all plaintext of blk
most_value_c ← CalcFreqValue(blk)
OTP ← most_value_p ⊕ most_value_c
for each encrypted element value_c of blk do
    value_p ← value_c ⊕ OTP

Defense of SECA
Input  : PA, VN: physical address and version number of blk.
Output : OTP_i: generate multiple OTPs for blk
OTP ← AES-CTR_{K_e}(PA ‖ VN)
for each 128-bit key_i in keyExpansion of AES-CTR do
    OTP_i ← (OTP ⊕ key_i)
```

B. Bandwidth-aware encryption mechanism

To meet the high bandwidth demands of accelerators, increasing the number of AES engines incurs significant overhead, while using the same OTP for all data in a block introduces security risks.This dilemma often leads to a suboptimal trade-off between security and computational efficiency.

Challenge 1: High Hardware Resources for Confidentiality Protection. While maintaining an equivalent level of security, numerous approaches have bolstered encryption parallelism by stacking multiple AES engines. For example, Securator [11] uses four AES-128 engines to concurrently encrypt a 64B data block, as shown in Fig. 2(c). Nevertheless, the drawback of sacrificing hardware resources to achieve performance improvements in encryption and decryption is apparent, particularly for edge DNN accelerators constrained by limited hardware capabilities. This situation emphasizes the crucial equilibrium needed to optimize security and operational efficiency in these systems.

Challenge 2: Security Threat for Shared OTP. A straightforward approach is to use each engine once per data block, with each 128-bit

segment within this data block sharing the same OTP. Nevertheless, assigning a OTP to a individual data block poses security risks and could be vulnerable to a Single-Element Collision Attack (**SECA**). The principle of a SECA attack when a data block shares the same OTP is outlined in lines 1-4 of Algorithm 1. By employing the CALCFREQVALUE function, we can identify the most frequently encrypted data within the block, denoted as $most_value_c$. Through comprehensive data analysis, we can infer the most common plaintext data within the block, denoted as $most_value_p$ (e.g., 0). Referring to the AES-CTR encryption formula Eq. 1, we can calculate the OTP for that block as $most_value_c \oplus most_value_p$, as shown in line 2 of Algorithm 1. Since the block shares this OTP, once obtained, we can derive all plaintext values within the encrypted block, as shown in lines 3-4 of Algorithm 1. Extending this principle further, we can extract the data values from every DNN layer and even the entire model.

Solution. We introduce a novel bandwidth-aware encryption mechanism to meet the high bandwidth requirements of accelerators without compromising the security of encryption. This method leverages the inherent features of AES-CTR encryption mode, using just a single AES engine and a minimal number of XOR logic gates, as illustrated in Fig. 3(a). To defend against SECA attacks, we first employ a standard AES-CTR encryption engine to create a shared OTP for a data block, as shown in line 5 of Algorithm 1. Subsequently, utilizing the keys array produced by the keyExpansion module within the AES-CTR encryption engine, we can generate multiple distinct OTPs by XORing the OTP with keys, as shown in lines 6-7 of Algorithm 1. This ensures that each 128-bit data segment within the data block corresponds to a unique OTP, thereby thwarting SECA attacks to safeguard security. Moreover, this mechanism significantly conserves hardware resources by utilizing a small number of XOR logic gates instead of an equivalent number of AES engines, commonly used by traditional methods.

When it comes to securing the entire DNN model, using a single key in the AES-CTR keyExpansion process is typically sufficient to ensure security. We proceed with the assumption that multiple keys are generated through the keyExpansion module using just one key. The keys produced by key expansion are inherently secure. These keys can be stored on-chip or generated in real-time. By XORing the shared OTP from a data block with these different keys, new secure OTPs can be generated. When generating OTPs from multiple keys derived from a single key input for keyExpansion doesn't meet the accelerator's bandwidth requirements, expanding the key expansion input to $key \oplus$ (PA ‖ VN) can solve the issue. This approach produces enough OTPs for a data block, meeting both security and bandwidth needs simultaneously.

C. Multi-level integrity verification mechanism

The granularity of integrity verification, whether too large or too small, can be problematic. Small granularity increases security metadata and off-chip memory overhead, affecting performance. Larger granularity reduces metadata overhead but can lead to redundant processing due to tiling overlaps. SeDA addresses this with a multi-level integrity verification mechanism that combines the flexibility of fine granularity, which avoids redundant security computations, with near-zero overhead of coarse granularity, significantly reducing or eliminating off-chip memory access overhead.

Challenge 1: Expensive Off-Chip Memory Access for Integrity Checks. While recent research efforts [8], [9] have successfully eliminated the off-chip memory access overhead of VNs and Merkle Trees, the overhead introduced by MACs still remains to be adequately addressed. Securator [11] proposed a layer-level freshness and integrity check method to reduce the MACs overhead by XORing MACs of all blocks within a layer, with a block size granularity of 32 bytes. The approach didn't consider tile overlaps within a layer, causing redundant computations like repeated integrity checks, which adds costs. Securator also ignored different tiling patterns between layers. Fig. 3(b) shows that $ofmap$ in layer i and $ifmap$ in layer i+1 use different strategies, which can vary in size and direction. Not addressing inter-layer tiling properly can lead to extra costs or disrupt the model due to inaccurate integrity checks.

Challenge 2: Security Threat for XOR-MAC Scheme. The XOR-MAC scheme offers parallelizability, incrementality, and provable security, with its security being on par with that of chaining MAC [16]. However, XORing all MACs generated by directly hashing the ciphertext within a layer to produce a unique layer MAC could lead to a Re-Permutation Attack (RePA), as illustrated in lines 1-6 of Algorithm 2. SHUFFLEORDER rearranges the order of data blocks in the current layer and computes SUM_MAC_shuffle for that layer using XOR operations. Because XOR is commutative, meaning the order of operands doesn't affect the result, an attacker can successfully pass the VERIFYINTEG(SUM_MAC, SUM_MAC_shuffle) verification. This rearrangement of data blocks within a layer can hinder correct decryption of ciphertext, posing a security risk due to RePA vulnerability.

Solution. To reduce off-chip memory access overhead from integrity verification, we propose a secure multi-level mechanism. It combines block-level and layer-level checks to minimize memory access by leveraging deterministic memory access patterns of DNNs, while ensuring security. We use the scheduling search strategy proposed in the SecureLoop [10] to obtain the optimal authentication block

Algorithm 2: Attack and defense of RePA

```
Attack of RePA
Input  : MAC_i: the MAC of a data block in one layer.
Output : plaintext_e: error plaintext.
1 SUM_MAC ← ⊕_{i=1}^{n} MAC_i
2 SHUFFLEORDER(MACs)
3 SUM_MAC_shuffle ← ⊕_{i=1}^{n} MAC_i'
4 if True = VERIFYINTEG(SUM_MAC, SUM_MAC_suffle) then
5     for each encrypted data block blk of one layer do
6         plaintext_e ← DECRYPT(blk)

Defense of RePA
Input  : layer_id, fmap_idx, blk_idx: layer number, feature
         map index and block index of layer_id.
Output : Secure layer MAC.
7 for each encrypted data block blk of layer_id do
8     MAC_i ← HASH_{K_h}(blk ‖ PA ‖ VN ‖ layer_id ‖ fmap_idx
        ‖ blk_idx)
```

TABLE I
COMPARISON OF MULTI-LEVEL INTEGRITY VERIFICATION GRANULARITY.

Granularity	Flexibility	Off-chip Access Overhead	Storage
optBlk	●	◔	Off-chip
layer	●	◔	Off/On-chip
model	●	○	On-chip

*Note: Color depth shows intensity, while coverage area indicates ratio.

TABLE II
DNN SIMULATION CONFIGURATIONS [20], [21].

Metrics	Server (Google TPU v1)	Edge (Samsung Exynos 990)
PE	256 x 256 in systolic array	32 x 32 in systolic array
Bandwidth	20 GB/s with 4 channels	10 GB/s with 4 channels
Frequency	1 GHz	2.75 GHz
SRAM	24 MB	480 KB
Presision	1-B for per element	1-B for per element

(optBlk), which synergizes with our work orthogonally. We propose a novel multi-level integrity verification mechanism involving three granularities of MAC types: optBlk MAC, layer MAC, and model MAC, as shown in Fig. 3(b). Table I compares their features in detail. The optBlk MAC offers flexibility by accounting for intra-layer tiling overlap and diverse inter-layer tiling patterns, avoiding redundant computations from repeated integrity checks. By XORing all optBlk MACs within a layer to create the layer MAC, we achieve a significant reduction in the number of MACs needed for DNN model integrity verification, despite a slight delay due to layer-specific checks. This allows layer MACs to be stored in on-chip SRAM, eliminating off-chip memory access costs. The Model MAC uses a single MAC to represent the entire model weights on-chip, further reducing off-chip memory costs and conserving on-chip SRAM, with verification results available only at the end of model inference. To prevent RePA attacks, we associate each encrypted optBlk data block with specific location details like PA, VN, $layer_{id}$, $fmap_{idx}$, and blk_{idx}. We then compute the corresponding MAC using hashing, as detailed in lines 7-8 of Algorithm 2. In summary, our multi-level integrity verification mechanism can remove off-chip memory overhead from security metadata, ensuring security while maintaining integrity guarantee.

IV. EVALUATION

A. Experimental Setup

Accelerators. In order evaluate DNN inference behaviours, we use an open-source cycle-level DNN simulator SCALE-Sim2 [17], [18] developed by ARM Research to 1) study the inference execution for various DNN models; 2) analyse the performance overhead of different memory protection schemes. The DNN accelerator can generate detailed computation information of systolic array, and DRAM access traces. After obtaining the DRAM traces, we use various memory protection mechanisms to calculate execution time and bandwidth usage, producing the total DRAM traces after running the security simulator. Finally, we use the DRAM simulator Ramulator2 [19] to simulate the total DRAM access traces.

Configurations. Table II presents the DNN simulation configurations, featuring a server NPU (Google TPU v1) and an edge NPU (Samsung Exynos 990). To balance DNN computation and memory bandwidth, we simulate four 64-bit DDR channels for both the server and edge NPUs.

Benchmarks. For DNN accelerators, we evaluate SeDA across various DNN models, including Lenet (let), Alexnet (alex), Mobilenet (mob), ResNet18 (rest), GoogleNet (goo), DLRM (dlrm), AlphaGoZero (algo), DeepSpeech2 (ds2), FasterRCNN (fast), NCF_recommendation (ncf), Sentimental_seqCNN (sent), Transformer_fwd (trf), Yolo_tiny (yolo). These models are selected from diverse machine learning domains, such as computer vision, speech recognition, natural language processing, gaming, and personalized recommendation.

Memory Protection Simulation. We implemented accelerators based on Intel SGX, MGX, and SeDA security mechanisms, using an unprotected accelerator as a benchmark, and set the size of protected memory to 16GB. SGX uses a multi-level Integrity Tree with 56-bit VNs and 64-bit MACs, along with a 16KB VN cache and 8KB MAC cache, both using an LRU replacement policy for write-back and write-allocate strategies. We use two different protection granularities: 64B and 512B. To ensure fairness, SeDA stores layer MACs off-chip.

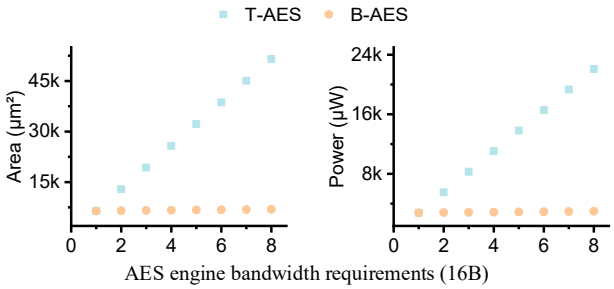
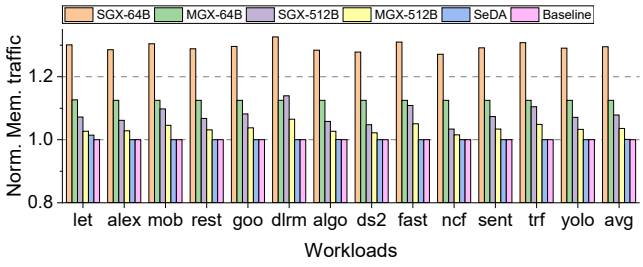
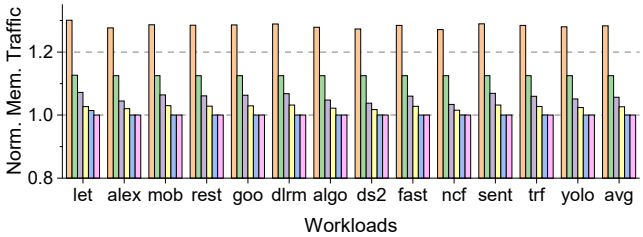


Fig. 4. The area and power with increasing AES engine bandwidth requirements.



(a) Server NPU



(b) Edge NPU

Fig. 5. The normalized memory traffic of memory protection schemes for various workloads

B. Experimental Results

We compare the memory traffic, which refers to the data exchanged between off-chip memory and accelerators, and performance across a unprotected baseline and five protection schemes: SGX-64B, SGX-512B, MGX-64B, MGX-512B, and our proposed SeDA, as shown in Table III. All results are normalized to the baseline without protection.

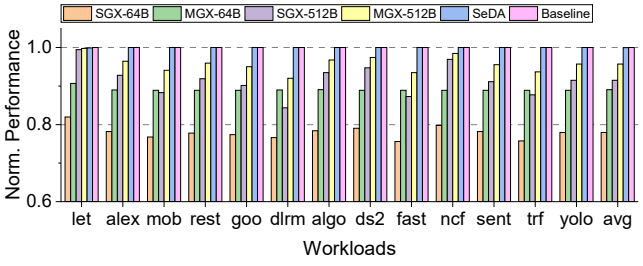
Area and Power. We developed a simulator based on 28nm technology to evaluate area and power, utilizing the AES engine implementations detailed in [22]. We refer to the bandwidth-aware encryption mechanism as B-AES, and other traditional methods using multiple AES engines as T-AES. Through simulation, we assessed the changes in area and power. The x-axis represents the bandwidth required by the encryption engine to match the accelerator's bandwidth needs, expressed as multiples of the bandwidth provided by a single AES engine. As shown in Fig. 4, our proposed B-AES demonstrates strong scalability, with minimal increases in area and power consumption as the accelerator bandwidth increases.

Memory traffic. Fig. 5 compares the memory traffic overhead using the DNN simulation configurations listed in Table II. On average,

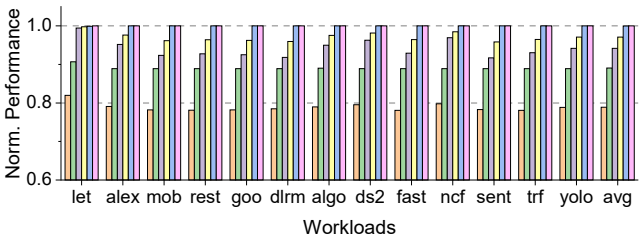
TABLE III
COMPARISON OF MEMORY PROTECTION SCHEMES.

Protection Scheme	Encryption Granularity	Integrity Granularity	Off-chip Memory Access	DNN Tiling Pattern	Encryption Scalability
SGX-64B	16B	64B	MAC,VN,IT	✗	✗
SGX-512B	16B	512B	MAC,VN,IT	✗	✗
MGX-64B	16B	64B	MAC	✗	✗
MGX-512B	16B	512B	MAC	✗	✗
SeDA	bandwidth-aware	multi-level	minimal to no cost	✓	✓

*Note: "IT" signifies integrity tree.



(a) Server NPU



(b) Edge NPU

Fig. 6. The normalized performance of memory protection schemes for various workloads

SGX-64B increases memory traffic by 30% for Server NPU and 28.29% for Edge NPU. In contrast, MGX-64B, without the overhead from additional VNs and MT, increases traffic by only 12.51% and 12.63%, respectively. Increasing the protection granularity from 64B to 512B significantly reduces memory traffic. SGX-512B cuts traffic by 7.83% on Server NPU and 5.13% on Edge NPU. MGX-512B achieves reductions of 3.59% and 2.39% on these devices, respectively. Expanding the protection granularity to 512 bytes reduces memory traffic by minimizing security metadata. However, using larger data blocks can lead to inefficiencies due to misalignment with intra-layer tiling overlaps and varying inter-layer tiling patterns. This mismatch can hinder memory access and resource utilization, emphasizing the need for a balanced approach in selecting granularity to improve system performance and efficiency. Our proposed SeDA uses a multi-level integrity verification mechanism by XORing all optBlk MACs into a layer MAC. Fig. 5 shows that SeDA introduces near-zero memory traffic, with an overhead of only 0.12% for Server NPU and 0.03% for Edge NPU, highlighting the superiority of our proposed protection mechanism.

Performance. Fig. 6(a) presents a performance comparison analysis among the baseline and five memory protection mechanisms on Server NPU. SGX-64B is 22.04% slower, MGX-64B is 10.93% slower, SGX-512B is 8.49% slower, and MGX-512B is 4.28% slower than the unprotected baseline. In contrast, our proposed SeDA impacts performance by less than 1%, making it nearly negligible. This is primarily attributed to proposed multi-level integrity verification mechanism, which can reduce or even completely eliminate the performance overhead of off-chip memory accesses caused by integrity verification. By minimizing the storage and retrieval of MACs in off-chip memory, SeDA effectively manages the security metadata overhead from integrity verification. Additionally, by storing these small layer or model MACs directly in on-chip SRAM, we can remove the performance overhead associated with off-chip access for integrity verification. For Edge NPU, Fig. 6(b) presents similar findings. Compared to the baseline, SGX-64B, MGX-64B, SGX-512B, and MGX-512B slow down by 21.10%, 10.95%, 5.84%, and 2.90%, respectively. Remarkably, our proposed SeDA results in an almost imperceptible performance drop.

V. CONCLUSION

In this work, we introduce SeDA, a secure and efficient DNN accelerator architecture designed to ensure confidentiality and integrity in untrusted environments. Using bandwidth-aware encryption and multi-level integrity verification mechanisms, SeDA provides excellent scalability with minimal overhead to match computational bandwidth of accelerators, and significantly reduces or even eliminates off-chip memory access overhead. Experimental results show that proposed SeDA meets bandwidth requirements with near-zero hardware overhead, and significantly reduces the performance overhead compared to the state-of-the-art approaches.

ACKNOWLEDGMENT

This research was partially supported by the National Key R&D Program of China (Grant No. 2022YFB3608300). It was supported in part by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government. It was also supported in part by Shenzhen Science and Technology Innovation Commission (Grant No. SGDX20220530111405040), Beijing Natural Science Foundation (Grant No. Z210006), Hong Kong Research Grant Council (Grant Nos. 27209621, 17205922, 17212923), the National Natural Science Foundation of China under Grant 62204111.

REFERENCES

[1] D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal, N. Kumar, G. P. Joshi, and W. Cho, "A review on autonomous vehicles: Progress, methods and challenges," *Electronics*, vol. 11, no. 14, p. 2162, 2022.

[2] G. Rong, A. Mendez, E. B. Assi, B. Zhao, and M. Sawan, "Artificial intelligence in healthcare: review and prediction case studies," *Engineering*, vol. 6, no. 3, pp. 291–301, 2020.

[3] F. Rundo, F. Trenta, A. L. Di Stallo, and S. Battiato, "Machine learning for quantitative finance applications: A survey," *Applied Sciences*, vol. 9, no. 24, p. 5574, 2019.

[4] "Artificial intelligence AI hardware market size and forecast," https://www.verifiedmarketresearch.com/product/global-artificial-intelligence-ai-hardware-market/, 2024.

[5] V. Costan and S. Devadas, "Intel SGX explained," *Cryptology ePrint Archive*, 2016.

[6] P. Zuo, Y. Hua, L. Liang, X. Xie, X. Hu, and Y. Xie, "Sealing neural network models in encrypted deep learning accelerators," in *ACM/IEEE Design Automation Conference*, 2021, pp. 1255–1260.

[7] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Guardnn: secure accelerator architecture for privacy-preserving deep learning," in *ACM/IEEE Design Automation Conference*, 2022, pp. 349–354.

[8] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "MGX: Near-zero overhead memory protection for data-intensive accelerators," in *International Symposium on Computer Architecture*, 2022, pp. 726–741.

[9] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, "TNPU: Supporting trusted execution with tree-less integrity protection for neural processing unit," in *International Symposium on High-Performance Computer Architecture*, 2022, pp. 229–243.

[10] K. Lee, M. Yan, J. Emer, and A. Chandrakasan, "SecureLoop: Design space exploration of secure DNN accelerators," in *IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 194–208.

[11] N. Shrivastava and S. R. Sarangi, "Securator: A fast and secure neural processing unit," in *International Symposium on High-Performance Computer Architecture*, 2023, pp. 1127–1139.

[12] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *International Symposium on High-Performance Computer Architecture*, 2003, pp. 295–306.

[13] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly," in *IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 183–196.

[14] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in *USENIX Security Symposium*, 2020, pp. 2003–2020.

[15] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *Ieee Access*, vol. 6, pp. 14 410–14 430, 2018.

[16] M. Bellare, R. Guérin, and P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions," in *Annual International Cryptology Conference*, 1995, pp. 15–28.

[17] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[18] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using scale-sim," in *International Symposium on Performance Analysis of Systems and Software*, 2020, pp. 58–68.

[19] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A modern, modular, and extensible DRAM simulator," *Computer Architecture Letters*, 2023.

[20] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *International Symposium on Computer Architecture*, 2017, p. 1–12.

[21] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "7.1 An 11.5TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile SoC," in *IEEE International Solid-State Circuits Conference*, 2019, pp. 130–132.

[22] U. Banerjee, "Energy-efficient protocols and hardware architectures for transport layer security," Ph.D. dissertation, Massachusetts Institute of Technology, 2017.

CXL-ECC: an Efficient LRC-based on-CXL-Memory-eXpander-Controller ECC to Enhance Reliability and Performance of DRAM Error Correction

Yixuan Liu†, Yunfei Gu†, Junhao Dai†, Xinyuan Wu†, Chentao Wu†*, Xinfei Guo‡, Jieru Zhao†, Jie Li†, Minyi Guo†,
†Shanghai Jiao Tong University, Shanghai, China
‡University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai, China
*Corresponding Authors: wuct@cs.sjtu.edu.cn

***Abstract*—Compute eXpress Link (CXL) offers an effective interface for connecting CPUs with external computing and memory devices. CXL Memory eXpander Controller (CXL-MXC) is gaining attention for its ability to boost memory capacity and bandwidth more efficiently than traditional DDR DIMMs. Despite extensive research on MXC performance and adaptation, DRAM reliability in CXL architecture remains underexplored. Traditional fault tolerance mechanisms like replica or RAID-based systems would significantly increase bandwidth overhead in the CXL fabric, adversely affecting system performance. To address this, we propose the on-CXL-Memory-Expander-Controller ECC (CXL-ECC), by using Locally Recoverable Codes (LRC) as the Inter-Channel-ECC (IC-ECC) and offloading its process to the expander, we eliminate extra memory access requests in the CXL fabric. Consequently, we conduct several experiments to demonstrate that our approach enhances DRAM reliability by more than 10^9, compared to state-of-the-art ECC methods. Relative to RAID-enabled CXL switch, it reduces additional bandwidth overhead from 63.5% to 3.4% and improves system performance by 12%.**

***Index Terms*—DRAM Reliability, ECC, Compute eXpress Link**

I. Introduction

Memory failures are the primary factor affecting server reliability, with over 37% of hardware faults caused by memory errors [1]. To mitigate memory errors, Error Correcting Code (ECC) [2–4] is commonly implemented in the traditional DDR memory devices. Specifically, Rank-Level ECC (R-ECC) and On-Die ECC (OD-ECC) [5] encodes data and stores error correction bits in parity DRAM chips and redundancy on DRAM chips. However, merely increasing redundancy within a DRAM device to improve reliability is insufficient. Therefore, researchers have proposed various solutions, such as Memory Mirror [6], RAIM-3 [7], and RAIM-5 [8], which implement Channel-Level Fault Tolerance (Channelkill) by using replicas or RAID [9] across memory channels. Given that these methods are integrated within memory controllers of computing devices, we categorize them as Compute Managed Type (C-Type).

In recent years, Compute eXpress Link (CXL) [10] has been proposed as an emerging open industry standard interconnect technology, inherits high-bandwidth characteristics of the PCIe protocol and provides additional cache and memory semantics, significantly reducing interface latency. This makes CXL an ideal choice for connecting CPUs and heterogeneous computing and memory devices, and it is applied in the latest storage devices, including CXL Dynamic Random Access Memory (DRAM), Solid State Drive (SSD) or Hybrid [11], and CXL Persistent Memory (PM) [12]. Extensive research has addressed CXL memory prototype devices performance, focusing on operating system adaptation and memory-intensive application optimization [13–15].

TABLE I: Comparison of Channelkill technology applied in CXL architectures

	RAIM-5	Switch-RAID	CXL-ECC
HW Modification	LLC and IMC	CXL Switch	CXL MXC
Bandwidth overhead	Medium	High	None ☺
Compatibility	Low	High	High ☺
System Overhead	Low	High	Low ☺
IC-ECC Code	XOR	XOR	LRC ☺
DRAM Reliability	Low	Low	High ☺

CXL may bring new opportunities for memory reliability improvement. To facilitate Channelkill implementation within the emerging interconnect, some researchers have suggested Switch Managed Type (S-Type) mechanism, where the switch maintains RAID management [16]. However, when applied to the CXL architecture, both C-Type and S-Type still have two major drawbacks: ❶ Parity update introduces a substantial addition of memory requests across the CXL fabric, resulting in significant degradation of system performance. ❷ XOR-based Inter-Channel-ECC (IC-ECC) exhibits poor capability of error correction in addressing multiple random faults.

To enhance high memory reliability and optimal system performance simultaneously, we propose an efficient on-CXL-Memory-eXpander-Controller ECC (CXL-ECC) approach in this paper. ❶ This method confines the parity update related memory requests within the MXC, thus reducing external memory bandwidth usage and improving system efficiency. ❷ It innovatively implements Locally Recoverable Codes (LRC) [17] as the IC-ECC within the CXL MXC for fault tolerance in extensive fine-grained faults while maintaining Channelkill. Table I compares RAIM-5 (SOTA Type-C), Switch-RAID (SOTA Type-S) and CXL-ECC across the compatibility, system performance and reliability. We also employ the Multi-Logic Device (MLD) mechanism from CXL 2.0 [10], enabling the Fabric Manager to allocate protected regions in a granularity of Logic Device for selective protection.

We have the following contributions in this paper,

- We propose an IC-ECC-enabled CXL Memory eXpander Controller with selective protection. It eliminates the bandwidth consumption of parity updates in the CXL fabric by offloading IC-ECC processing logic to CXL MXC.
- We propose a LRC-based IC-ECC, which enhances the fault tolerance of Channelkill while ensuring the tolerance of numerous random faults. Furthermore, our study conducts the first comprehensive reliability analysis of LRC in the

context of DRAM ECC.

- We evaluated our design using the Champsim [18] and Ramulator2 [19] simulators across various workloads. Our proposed CXL-ECC outperforms state-of-the-art methods, achieving a 60% reduction in bandwidth consumption on average and enhancing DRAM reliability by more than 10^9, with negligible system performance cost.

II. Background and Motivation

A. CXL Memory eXpander Controller

Compute Express Link (CXL) [10] is a cutting-edge interconnect protocol designed to optimize communication between processors and various devices, such as accelerators and memory expander. Leveraging the physical layer of PCIe 5.0/6.0, CXL facilitates high bandwidth while enabling low-latency, coherent memory access. CXL Memory eXpander Controller (MXC), classified as Type-3 devices, implements `CXL.io` and `CXL.mem` protocols to deliver high-performance memory expansion and enable hardware memory pooling. Leading memory manufacturers including Samsung, SK Hynix, and Micron have released CXL MXC products. Marvell also supplies MXC chips [20]. Additionally, internet companies are actively testing MXC performance and modifying systems [13, 21], to support this emerging technology.

B. DIMM Organization and DRAM Faults

1) ***Dual-Inline-Memory-Module (DIMM) Organization****:* DRAM chips have multiple data pins (DQs) that transmit data in parallel, and a chip with N DQs is generally referred to as an xN chip. The DDR5 JEDEC [5] DIMM is equipped with two independent 32-bit sub-channels. In x4 ECC-DIMM, as shown in Fig. 1, each sub-channel is equipped with 10 x4 DRAM chips, where 2 parity chips are used to store the check data of Rank-Level ECC (R-ECC). The internal structure of DRAM chip follows a hierarchical organization, where each consists of multiple independent banks, and each bank can be further divided into rows and columns. DDR5 implements On-Die ECC (OD-ECC), utilizing a (136,128) SEC code for data protection within bank.

2) ***DRAM Faults****:* Currently, there are two major types of faults in DRAM devices: **(i) Operational Faults** are caused by manufacturing defects and wear-out during usage, and can manifest as cell, row, column, or bank-level faults at a coarser granularity [22]. **(ii) Scaling Faults** appear in the form of random cell faults. As DRAM manufacturing technology scales down, the scaling fault rate increases, reaching as high as 10^{-4} when the process is scaled down to sub-20nm [23]. Each fault introduces random errors, varying from a single bit to 2 symbols in an ECC codeword, and poses significant challenges to the design of high-reliability memory architectures.

C. Error Correcting Code in DRAM

Error Correcting Code (ECC) detects and corrects errors through the addition of redundant information, known as parity, which is algorithmically generated from the protected data. The combination of data and parity forms an ECC codeword.

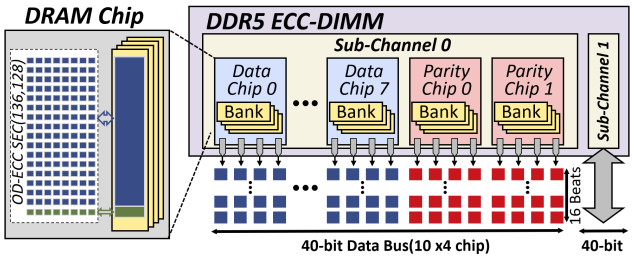


Fig. 1: Organization of x4 DDR5 ECC-DIMM

1) ***Distinctions among various ECCs****:* Single Error Correction (SEC), Single Error Correction-Double Error Detection (SEC-DED), and Double Error Correction (DEC) are a class of commonly used codes in Rank-Level ECC, which can detect and correct single or multiple bit errors. Another category of codes is represented by Reed-Solomon (RS) codes [2], specifically RS(n, k) codes, which are a type of Maximum Distance Separable (MDS) code. In RS(n, k) codes, n and k represent the total number of symbols and data symbols in the codeword, respectively. When multiple parity chips are available, RS codes can be used to implement Chipkill [4], a fault-tolerant technique that can correct errors even when a single chip completely fails.

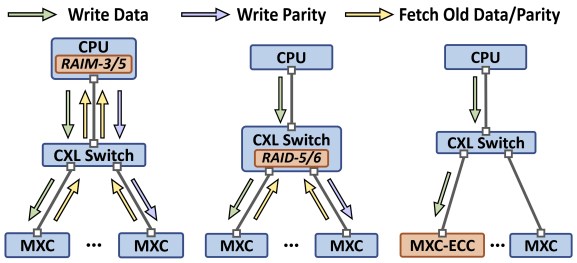
2) ***Recent Advances in R-ECC*** *:* Researchers have made great efforts on the collaborative design of R-ECC and OD-ECC. XED [24] uses SEC-DED as OD-ECC and replaces the original R-ECC with XOR, enhancing the error detection capability of OD-ECC to improve the error correction capability of R-ECC. DUO [25] encodes the data and R-ECC and OD-ECC parity of a single cache line into a long RS code, achieving Chipkill capability while correcting additional random faults. Unity ECC [26] abandons OD-ECC and selects a coding matrix that satisfies both RS$(10, 8)$ and BCH$(80, 64)$ [27], simultaneously achieving Chipkill and DEC fault tolerance.

However, these techniques have three common issues. (i) The simultaneous modification of both DIMM device and memory controller, and potentially even requiring modifications to the DRAM chip's OD-ECC encoding [24, 25, 28], poses notable compatibility challenges to JEDEC DDR standards [5]. (ii) They require pre-setting the redundancy in the DIMM, which cannot adapt to diverse memory reliability requirements. (iii) To achieve Chipkill, two parity chips are essential, requiring the exclusive use of x4 DRAM chips and affecting DIMM specification design.

D. Challenges with Current Channelkill Technology

To address these aforementioned common issues and further enhance the memory reliability beyond Chipkill, the Cross Channel-Level Fault Tolerance Mechanism (Channelkill) has been proposed and extensively studied, including C-Type and S-Type (defined in § I).

1) ***Compute Managed Type (C-Type)****:* Memory Mirror [6] and IBM-RAIM3 [7] achieve Channelkill fault tolerance by constructing channels as RAID-1/3. RAIM-5 [8] encodes 64B cache lines as RAID-5 stripes, introducing gang-writeback



(a) C-Type (b) S-Type (c) CXL ECC

Fig. 2: IO traffic of writeback in three types of Channelkill architecture

and selective protection that enables low-overhead and high-flexibility Channelkill. However, this type of Compute Managed Type (C-Type) Channelkill poses significant challenges to compatibility, because it's nearly impossible for two independent hardware manufacturers to implement a fully compatible complex fault tolerance mechanism.

*2) **Switch Managed Type (S-Type)**:* This type of approaches address the challenge of memory expander failures within the CXL fabric. SMC [16] proposed integrating replication and RAID-5/6 functionality within the CXL switch, referred to as Switch-RAID. It enables the CXL switch to intercept memory requests from upstream devices and manage parity updates and data recovery, effectively resolving compatibility issues with computing devices.

However, existing Channelkill are still remaining challenges: ❶ **Significant overhead incurred by parity update:** The implementation of C/S-Type fault tolerance in CXL fabric, as illustrated in Figures 2(a) and 2(b), introduces a two-step writeback process: reading the original data and parity from cache lines, and writing back their updated versions. These additional operations consume a substantial amount of CXL fabric bandwidth, severely impacting system performance. ❷ **XOR-based IC-ECC suffers from poor reliability:** Current IC-ECC widely employs XOR Code for correcting single cache line errors, which fails to correct errors when multiple channels encounter moderate faults, such as multiple symbol errors in RS codes. The root cause is that XOR does not account for the correlation between error distribution in cache lines and DRAM device failure patterns, a detailed explanation will be presented in § III-A.

*3) **MXCs serve as an optimal position for IC-ECC logic**:* The latest CXL MXCs are equipped with 16 CXL lanes and 4 DDR5 6400MHz channels [20]. Although the total read/write bandwidth of the CXL fabric is limited to 80GB/s, the device's internal bandwidth exceeds 200GB/s. Combining this with the analysis of the above challenges, this motivates us to integrate the IC-ECC processing logic into the memory expander, so that the additional memory requests generated by IC-ECC parity updates can be limited within the expander, effectively alleviating the bandwidth pressure on the CXL fabric. Therefore, we propose the CXL-Memory-Expander Controller ECC (CXL-ECC).

*4) **LRC exhibits potential as IC-ECC**:* Locally Recoverable Code (LRC) [17] is a variant of RS codes, denoted as LRC(n, k, r, p), where n denotes total code length, k denotes data length, r denotes global parity length, and p denotes local parity length. The data is divided into $\frac{k}{p}$ local parity groups, each forming an RS$(\frac{k}{p}+p, \frac{k}{p})$ code with local parity. The r global parity blocks are computed from the total k data blocks. LRC exhibits excellent ① **Local Recovery**, ② **Incremental Parity Update** and ③ **High Global Reliability** properties that satisfy the requirements of IC-ECC. LRC is widely used as an erasure code in cloud storage systems [29], but its application in memory systems for ECC has not been explored yet to the best of our acknowledgment.

III. Design of Proposed CXL-ECC

CXL-ECC consists of two main components: (i) LRC-based Inter-Channel ECC (IC-ECC) algorithm design. (ii) IC-ECC-enabled CXL-MXC controller design.

A. Design of LRC-based IC-ECC Algorithm

Drawing upon the analysis presented in § II-C, we propose LRC-based IC-ECC as a solution to address the challenges outlined in § II-D. LRC maintains Single-Symbol-Correction (SSC) capability within each data channel, eliminating the need to read all cache lines that constitute IC-ECC during error detection and correction. By incorporating local parity into the global decoding process, the system achieves robust correction capabilities for numerous random symbol faults.

*1) **Code Construction**:* Optimal LRC codes [30] conventionally enforce strict constraints on codeword parameters, thereby restricting the number of global parities to not exceed the data symbols within a local group. Therefore, we adopt Pyramid Code [31] for IC-ECC implementation, as it provides equivalent data symbol fault tolerance. Leveraging RS codes constructed with a Vandermonde matrix over GF(256), we implement the code through the following steps:

Step I: We set the number of data channels to be c, then the corresponding Vandermonde matrix for $RS(8\times c+12, 8\times c)$ is calculated as shown in Equation 1:

$$G_V = \begin{bmatrix} (2^0)^0 & (2^0)^1 & \dots & (2^0)^{8\times c-1} \\ (2^1)^0 & (2^1)^1 & \dots & (2^1)^{8\times c-1} \\ \vdots & \vdots & \ddots & \vdots \\ (2^{11})^0 & (2^{11})^1 & \dots & (2^{11})^{8\times c-1} \end{bmatrix} \tag{1}$$

Step II: Let $i = 0, 1, \dots, c-1$. We align and partition the first two rows of G_V into segments of 8 columns each to obtain the encoding matrix A_i within the local parity group. As demonstrated in Equation 2, forms a generator matrix for a local parity group constructing an RS(10,8) codeword, thus satisfying the local recovery requirements of LRC.

$$A_i = \begin{bmatrix} 1^{8i} & 1^{8i+1} & \dots & 1^{8i+7} \\ 2^{8i} & 2^{8i+1} & \dots & 2^{8i+7} \end{bmatrix} \tag{2}$$

Step III: We define the global coefficients of the LRC generator matrix shown in Equation 3:

$$G_G = \begin{bmatrix} (2^2)^0 & (2^2)^1 & \dots & (2^2)^{8\times c-1} \\ (2^3)^0 & (2^3)^1 & \dots & (2^3)^{8\times c-1} \\ \vdots & \vdots & \ddots & \vdots \\ (2^{11})^0 & (2^{11})^1 & \dots & (2^{11})^{8\times c-1} \end{bmatrix} \tag{3}$$

Step IV: We define the generator matrix of the LRC as Equation 4, the construction of our proposed LRC-based IC-ECC is completed:

$$G_{LRC} = \begin{bmatrix} & & I & \\ A_0 & & & \\ & A_1 & & \\ & & \cdots & \\ & & & A_{c-1} \\ & & G_G & \end{bmatrix} \tag{4}$$

*2) **Data Encoding and Organization**:* Identical to the § III-A1, we assume a total of c data cache lines, encoded into $c+1$ cache lines using LRC. Fig. 3 illustrates the data layout of cache lines in the IC-ECC structure: the 80 bits received from two consecutive Beats are first aligned, then divided into 10 symbols according to the 4-DQ width to form an IC-ECC coding unit. Let C_{ij} denote the data transmitted in the j-th coding unit within the i-th cache line, where $i \in \{0, 1, \ldots, c\}$ and $j \in \{0, 1, \ldots, 7\}$. These units are encoded into an LRC$(10 \times (c+1), 8 \times c, 10, 2)$ code, with C_{cj} containing the global parities and each remaining C_{ij} constituting a local parity group. Cache lines are arranged in a RAID5 left-asymmetric layout [9] to minimize the I/O bottleneck caused by parity update operations.

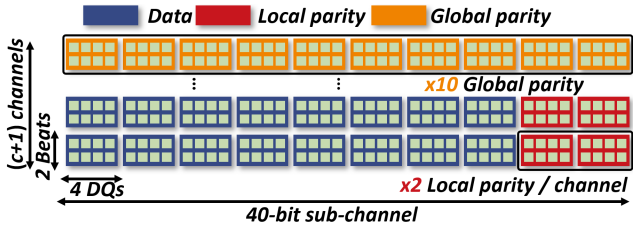


Fig. 3: an example of IC-ECC data layout

*3) **Decoding Process and Correction Capability**:* IC-ECC employs a two-level error correction scheme: **(a) Intra-channel local decoding:** R-ECC possesses SSC capability, which can achieves Chipkill on x4 DIMMs and Half-Chipkill on x8 DIMMs respectively. When R-ECC correction fails, an Uncorrectable Error (UE) is reported, triggering global decoding. **(b) Inter-channel global decoding:** All data and parity bits across channels are combined into a global $RS(c \times 8 + 12, c \times 8)$. This allows for either the recovery of a single faulty channel through erasure decoding or the correction of up to 6 random symbol errors using the RS Decoder.

TABLE II: Configurations and error correction capabilities of IC-ECCs

		w/o	XOR	RS	LRC
IC-ECC		-	$XOR(5,4)$	$RS(40,32)$	$RS(44,32)$
Global Parities		-	$64B \times 1$	$1B \times 8$	$1B \times 12$
Correctable symbol errors	min	1	3	4	6
	max	$c+1$	$c+10$	$c+10$	$c+10$
Chipkill w/ x4 DIMM		1	3	4	6
Chipkill w/ x8 DIMM		0	1	2	3

*4) **Compare performance of IC-ECCs within Channelkill**:* XOR-based IC-ECC generates parity by XORing data chips within each channel across 64B cache lines. RS-based IC-ECC, similar to LRC-based IC-ECC, uses data chips in the parity channel to store parity symbols that protect data chips in other channels. Table II presents the configurations and error correction capabilities of the three IC-ECCs with 4 data channels ($c = 4$). Due to its coarse encoding granularity, XOR-based IC-ECC exhibits limited tolerance for fine-grained errors, failing to correct dual-symbol errors occurring simultaneously in two channels. While RS-based IC-ECC offers superior fine-grained error correction compared to XOR, its local parity does not contribute to global decoding, limiting its correction capability to 4 arbitrary symbol errors. By contrast, LRC-based IC-ECC integrates local and global redundancy to form 12 parity symbols.

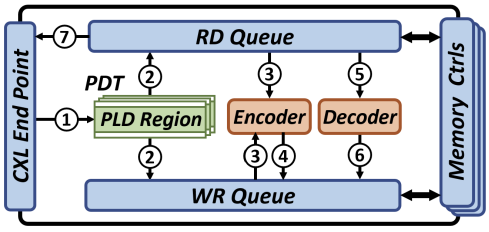


Fig. 4: Architecture of the CXL-ECC controller

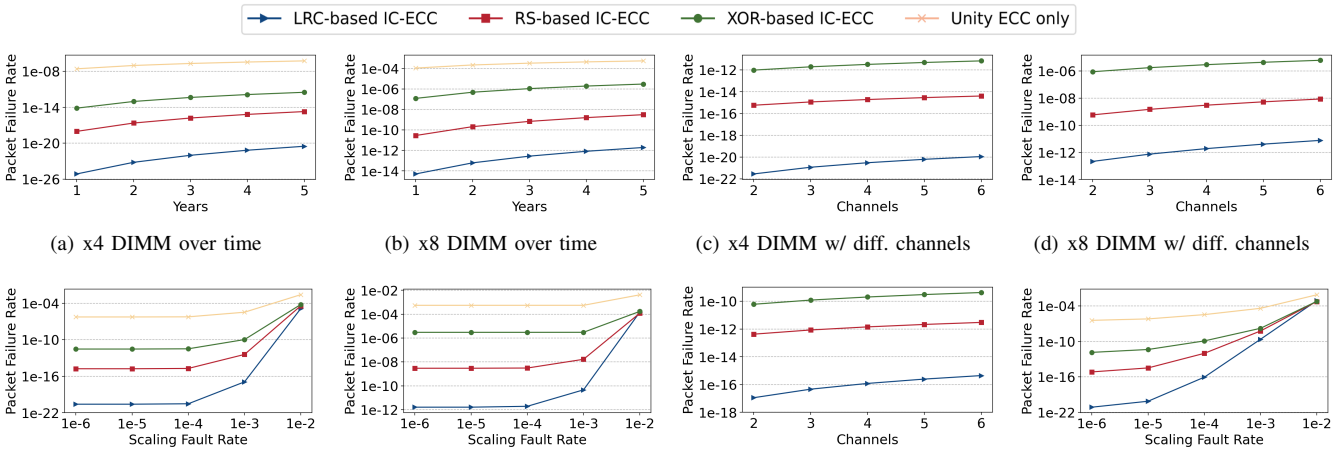
B. Design of CXL-ECC Controller System Architecture

The system architecture of CXL-ECC implements selective protection management, parity update procedures, and data recovery mechanisms.

*1) **Selective protection with Protected LD Table (PDT)**:* Multiple Logic Device (MLD) characteristic enables each Type-3 device to be logically partitioned into Logic Devices (LDs) and independently assigned to computing devices. Memory space is protected by IC-ECC at the LD granularity, with all protected LDs being recorded in the Protected LD Table (PDT). During high-reliability memory allocation by the Fabric Manager (FM), a sequential process is executed wherein: (1) the FM initiates an allocation request to the MXC, (2) the MXC performs physical space allocation within its internal DDR channels and registers the allocation in the PDT, and (3) the resultant protected LD is subsequently exported and assigned through the FM.

*2) **Parity update and data recovery via En-/De-coder**:* The processes for parity updates and data recovery of IC-ECC are executed by the En-/De-coder modules. **(i) Request Serve** begins with ① a filtering in the PDT. If an operation falls within the PDT, ② then while the request is placed in the RQ/WQ, the protected flag needs to be set in the corresponding entry. Read requests for previously written data and parity required for the incremental update are inserted into the RQ. ③ Upon completion of all read requests for a writeback operation, both new and previous data, along with the old parity, are passed to the Encoder. ④ A write request is issued for the encoded result. **(ii) Data Recovery** is triggered upon the detection of a R-ECC UE [1] in an entry with protected flag set. Subsequently, read requests associated with data recovery are enqueued in the RQ. ⑤ When all data required for the recovery operation has been retrieved, the Decoder fetches the data and performs

[1]To enable the IC-ECC En-/De-coder to access data from the parity chips, we relocate the R-ECC encoding and decoding operations from the memory controller to the MXC.



(a) x4 DIMM over time (b) x8 DIMM over time (c) x4 DIMM w/ diff. channels (d) x8 DIMM w/ diff. channels

(e) x4 DIMM w/ diff. error rates (f) x8 DIMM w/ diff. error rates (g) w/ diff. channels & w/o OD-ECC (h) w/ diff. error rates & w/o OD-ECC

Fig. 5: The packet failure rate of different IC-ECC schemes over time , with different channels and scaling fault rates.

the decoding. ⑥ Following this, write requests for the decoded results are queued in the WQ. If the R-ECC UE is not in the PDT or the Decoder fails to decode, an UE is reported to the CXL End Point.

*3) **Scheduler optimization and conflicts management**:* The First-Ready First-Come-First-Serve (FR-FCFS) scheduling policy is modified to assign higher priority to CPU load requests when multiple read requests have equivalent priority levels. This optimization minimizes the impact on load latency caused by the data reads required for IC-ECC parity updates. To maintain data consistency during simultaneous parity updates and data recovery operations on the same IC-ECC codeword, operations are serialized, prioritizing data recovery to avoid deadlocks.

TABLE III: Operational fault rates[22]

Mode	Transient	Permanent
Single bit	14.2	18.6
Single word	1.4	0.3
Single column	1.4	5.6
Single row	0.2	8.2
Single bank	0.8	10.0

IV. Evaluation Method

A. Reliability Simulation Setup

To evaluate the reliability of LRC-based IC-ECC, we compare it with schemes using only Unity ECC (SOTA Rank-level ECC) as R-ECC, Xor-based IC-ECC (incorporated by both RAIM-5 and Switch-RAID) and RS-based IC-ECC with Unity ECC. The experiments employ 2-6 data channels plus one parity channel to construct IC-ECC with DDR5 x8 and x4 ECC-DIMM [2]. During fault injection process, random cell faults are inserted with a probability between 10^{-6} and 10^{-2} to simulate Scaling Faults, then Operational Faults are injected using the failure rates shown in Table. III [22] to simulate wear-out after 1-5 years of device operation. As the probability of UE occurring in LRC-based IC-ECC is extremely low (in some cases less than 10^{-20}), making it difficult to simulate the failure rate using Monte Carlo methods. We theoretically calculate the packet failure rate corresponding to 5×80 bits data constituting an IC-ECC.

[2]In the default configuration, the system employs 4 data channels, maintains a scaling fault rate of 10^{-4}, operates over a 5-year period, and implements OD-ECC protection for DRAM chips.

B. Performance Simulation Setup

TABLE IV: The system simulation configuration

Processor	
Cores	16 cores 4GHz, 4-wide OoO
L1I	32KB, 64B line, 8 way, 4 cycles, 8 MSHRs
L1D	48KB, 64B line, 12 way, 5 cycles, 16 MSHRs
L2C	512KB, 64B line, 8 way, 10 cycles, 32 MSHRs
LLC	32MB, 64B line, 16 way, 20 cycles, 1024 MSHRs
CXL Fabic [13]	
Latency	CPU Port: 25ns, Switch: 130ns, Memory Port: 40ns
Bandwidth	x8 CXL (PCIe5.0) lanes, 32 GB/s (bi-dierctional)
Memory eXpander Controller	
Queue Depth	MXC: 1024, DDR controller: 32
DDR Spec.	1 rank , DDR5 [5] 6400MHz, 32Gb ×8 chip, 8 bank groups, 4 banks per bank group
Channels	5 channel (4-data + 1-parity)

To evaluate the system-level performance of the CXL-ECC architecture, we integrate the Champsim [18] CPU simulator with the Ramulator2 [3] [19] Memory simulator, implementing the CXL protocol in Ramulator2. We also implement RAIM-5, RAID5-enabled Switch (Switch-RAID), and CXL-ECC. We evaluate 145 traces sampled from 41 workloads from SPEC CPU2006 [32], SPEC CPU2017 [33], PARSEC 2.1 [34], and Ligra [35]. During simulation execution, we run the same trace on all CPUs, measuring the execution time and CXL fabric traffic for around 100M instructions (after 10M warmup). Table IV summarizes all configurations used in the system simulation.

V. Evaluation Results

A. Impact on Reliability

(a) CXL-ECC substantially enhances DRAM reliability. Fig. 5(a) and Fig. 5(b) show the packet error rate over time

[3]Ramulator2 only support single sub-channel per DDR5 channel, we halve the CXL fabric bandwidth to simulate realistic DDR5 performance.

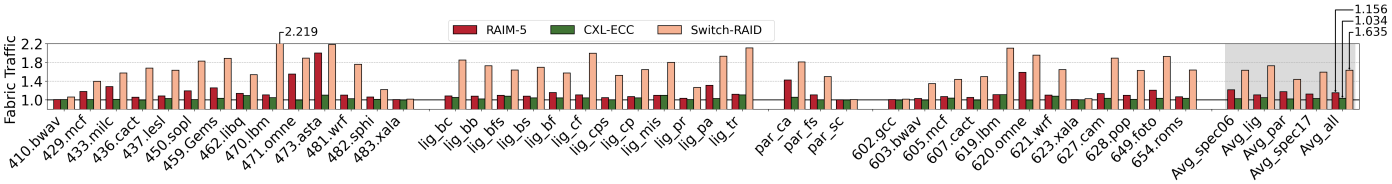


Fig. 6: Normalized fabric traffic of RAIM-5, Switch-RAID and CXL-ECC on each benchmark suite and the overall average

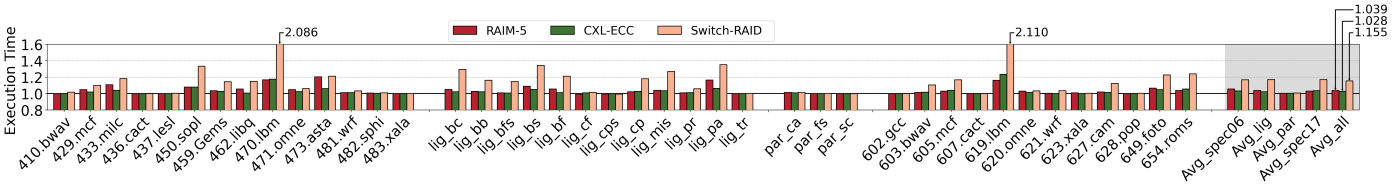


Fig. 7: Normalized execution time of RAIM-5, Switch-RAID and CXL-ECC on each benchmark suite and the overall average

for x4 and x8 DDR5 ECC-DIMM using different IC-ECC mechanisms. Equipped with x4 DIMM, CXL-ECC can achieve a packet failure rate below 3.1×10^{-21} while providing tolerance for complete failures of individual DIMMs, enabling highly reliable memory expansion for computing devices in CXL fabric. In comparison to the XOR-based ECC employed in existing Channelkill mechanisms, LRC-based IC-ECC yields reliability improvements of 6.3×10^5 and 10^9 in x8 and x4 DIMMs, respectively. Furthermore, LRC exhibits superior global error correction capabilities compared to refined RS codes, resulting in reliability improvements of 1.6×10^3 and 1.5×10^6 in x8 and x4 DIMMs, respectively.

Although x8 DIMMs inherently lack Chipkill capabilities, the use of LRC-based IC-ECC enables them to tolerate up to 3 random chip failures, thereby demonstrating robust resilience against both scaling and operational faults. Furthermore, the packet failure rate of x8 DIMMs employing LRC-based IC-ECC remains below 2×10^{-12}, which is lower than the 3×10^{-12} rate of x4 DIMMs protected by XOR-based IC-ECC.

(b) CXL-ECC demonstrates robust reliability across numerous channels and high failure rates. Fig. 5(c), 5(d), 5(e), and 5(f) illustrate the reliabilities of various IC-ECC mechanisms under different channel configurations and scaling error rates. Experimental results demonstrate that LRC-based IC-ECC-protected x4 DIMMs maintain an exceptionally low failure rate of 1.2×10^{-17} even when the scaling fault rate reaches 10^{-3}. Even with up to 6 data channels in x8 DIMMs, a minimal error rate of 7.8×10^{-12} can be maintained.

(c) CXL-ECC sustains exceptional reliability even in the absence of OD-ECC. Fig. 5(g) and 5(h) show the relationship between failure rates and varying parameters (channel count and error rate) for x4 DDR5 DIMMs without OD-ECC. Compared to OD-ECC-enabled configurations, all IC-ECC schemes show diminished error correction capabilities. Among these, LRC-based IC-ECC exhibits the most severe reliability degradation under increased scaling errors. This phenomenon can be attributed to the SSC-only mechanism of R-ECC within data channels, exhibiting inferior random error tolerance compared to the DEC-enabled Unity ECC deployed in other IC-ECCs. At an error rate of 10^{-3}, LRC maintains superior reliability despite these limitations, showing 29x and 98x improvements over RS-based and XOR-based implementations, respectively.

B. Impact on Performance

(a) CXL-ECC significantly reduces fabric bandwidth overhead. As shown in Fig. 6, the average additional bandwidth consumed by CXL-ECC across all workloads is 3.4%, which is significantly lower than the 15.6% and 63.5% of RAIM-5 and Switch-RAID, respectively. This is because all extra memory requests related to parity updates are confined within the MXC. For write-intensive applications such as `GemsFDTD`, `lbm`, and `BFS`, CXL-ECC can reduce the additional CXL fabric bandwidth overhead by up to 114% compared to S-type Switch-RAID technology.

It is worth noting that some applications, such as `omnetpp`, have a large additional bandwidth overhead but do not affect system performance. This is because they are not memory-intensive applications, and the additional memory requests do not put pressure on the CXL fabric.

(b) CXL-ECC has negligible impact on overall system performance. Fig. 7 shows the normalized execution time of CXL-ECC, Switch-RAID, and RAIM compared to the baseline system without IC-ECC. The S-Type Switch-RAID mechanism causes an execution time increase of over 13.4%, significantly impacting system performance. This is because a large amount of fabric bandwidth is used for parity updates. The workloads with severe performance degradation in Fig. 7 generally generate high fabric traffic in Fig. 6.

CXL-ECC and RAIM-5 exhibit marginal performance degradation of 2.8% and 3.9% respectively, significantly outperforming Switch-RAID with its substantial 15.5% overhead. Furthermore, our approach surpasses traditional C-type solutions like RAIM-5 while eliminating the need for extensive CPU modifications. Experimental results indicate that CXL-ECC delivers superior reliability and system-level performance while maintaining architectural compatibility.

VI. Conclusion

This paper presents on-CXL-Memory-Expander ECC (CXL-ECC) which integrates LRC-based IC-ECC in the CXL

memory controller. This approach effectively leverages the robust global error correction capabilities of LRC and mitigates the bandwidth demands imposed by parity updates on the fabric, thereby enabling the realization of highly reliable memory, which enhances a UE rate of lower than 3.1×10^{-21}, with minimal system performance overhead of 2.8%.

VII. Acknowledgements

We thank all the anonymous reviewers for their valuable comments. This work was supported in part by the National Key R&D Program of China (Grant No. 2023YFB4502900).

References

[1] I. Corporation. Intel mca+mfp helps jd stable and efficient cloud services.
[2] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
[3] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.
[4] AMD. Bios and kernel developers guide (bkdg) for amd family 15h models 00h-0fh processors.
[5] M. JEDEC. ddr5 sdram standard, jesd79-5c.01.
[6] I. Corporation. Address range partial memory mirroring.
[7] P. J. Meaney *et al.*, "Ibm zenterprise redundant array of independent memory subsystem," *IBM Journal of Research and Development*, vol. 56, no. 1.2, pp. 4–1, 2012.
[8] R. Zheng and M. C. Huang, "Redundant memory array architecture for efficient selective protection," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 214–227, 2017.
[9] P. M. Chen *et al.*, "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys (CSUR)*, vol. 26, no. 2, pp. 145–185, 1994.
[10] C. Consortium. Compute express link (cxl) specification, revision 3.1. [Online]. Available: https://computeexpresslink.org/cxl-specification.
[11] S. Eletronics. Samsung cxl solutions – cmm-h.
[12] S. N. I. Association. Persistent memory in cxl. [Online]. Available: https://www.snia.org/sites/default/files/PM-Summit/2021/snia-pm-cs-summit-Rudoff-PM-in-CXL-2021.pdf.
[13] H. Li *et al.*, "Pond: Cxl-based memory pooling systems for cloud platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 574–587.
[14] J. Liu *et al.*, "Exploring and evaluating real-world cxl: Use cases and system adoption," *arXiv preprint arXiv:2405.14209*, 2024.
[15] Y. Tang *et al.*, "Exploring performance and cost optimization with asic-based cxl memory," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 818–833.
[16] D. D. Sharma, "Novel composable and scaleout architectures using compute express link," *IEEE Micro*, vol. 43, no. 2, pp. 9–19, 2023.
[17] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, 2014.
[18] N. Gober *et al.*, "The championship simulator: Architectural simulation for education and competition," *arXiv preprint arXiv:2210.14324*, 2022.
[19] H. Luo *et al.*, "Ramulator 2.0: A modern, modular, and extensible dram simulator," *IEEE Computer Architecture Letters*, 2023.
[20] M. Inc. Structera x 2504 memory-expansion controller.
[21] Y. Sun *et al.*, "Demystifying cxl memory with genuine cxl-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 105–121.
[22] V. Sridharan and D. Liberty, "A study of dram failures in the field," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.
[23] P. J. Nair *et al.*, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 72–83, 2013.
[24] P. J. Nair *et al.*, "Xed: Exposing on-die error detection information for strong memory reliability," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 341–353, 2016.
[25] S.-L. Gong *et al.*, "Duo: Exposing on-chip redundancy to rank-level ecc for high reliability," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 683–695.
[26] D. Kim *et al.*, "Unity ecc: Unified memory protection against bit and chip errors," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–16.
[27] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.
[28] S. Jeong *et al.*, "Pair: Pin-aligned in-dram ecc architecture using expandability of reed-solomon code," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
[29] G. Yang *et al.*, "Xhr-code: An efficient wide stripe erasure code to reduce cross-rack overhead in cloud storage systems," in *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2022, pp. 273–283.
[30] G. Zhang, "A new construction of optimal (r, δ) locally recoverable codes," *IEEE Communications Letters*, vol. 24, no. 9, pp. 1852–1856, 2020.
[31] C. Huang *et al.*, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Transactions on Storage (TOS)*, vol. 9, no. 1, pp. 1–28, 2013.
[32] Spec cpu 2006. [Online]. Available: https://www.spec.org/cpu2006/.
[33] Spec cpu 2017. [Online]. Available: https://www.spec.org/cpu2017/.
[34] Parsec. [Online]. Available: https://parsec.cs.princeton.edu/.
[35] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2013, pp. 135–146.

OutlierCIM: Outlier-Aware Digital CIM-Based LLM Accelerator with Hybrid-Strategy Quantization and Unified FP-INT Computation

Zihan Zou[1], Shikuang Chen[1], Chen Zhang[1], Xing Wang[1], Zhichao Liu[1], Haoran Du[1], Xin Si[1,2], Hao Cai[1,2], Bo Liu[1,2*]
[1]School of Integrated Circuits, Southeast University, Nanjing, China
[2]National Center of Technology Innovation for EDA, Nanjing, China

***Abstract*—Activation outliers in Large Language Models (LLMs), which exhibit large magnitudes but small quantities, significantly affect model performance and pose challenges for the acceleration of LLMs. To address this bottleneck, researchers have proposed several co-design frameworks with outlier-aware algorithms and dedicated hardware. However, they face challenges balancing model accuracy with hardware efficiency when accelerating LLMs in a low bit-width manner. To this end, we propose OutlierCIM, the first algorithm and hardware co-design framework for the compute-in-memory (CIM) accelerator with outlier-aware quantization algorithm. The key contributions of OutlierCIM are 1) an outlier-clustered tiling strategy that regulates memory access and reduces inefficient workloads which are both introduced by outliers, 2) a hybrid-strategy quantization and a reconfigurable double-bit CIM macro array that overcome the low storage utilization and high latency of outlier-based LLM quantization, and 3) a quantization factor post-processing strategy and a dedicated quantizer that efficiently unify the multiplication and accumulation of outlier-caused FP-INT workloads. Implemented in a 28nm CMOS technology, OutlierCIM occupies an area of 2.25 mm². When evaluated at comprehensive benchmarks, OutlierCIM achieves up to 4.54× energy efficiency improvement and 3.91× speedup compared to the state-of-the-art outlier-aware accelerators.**

***Index Terms*—large language model, compute-in-memory, neural network accelerator, activation outlier, quantization.**

I. Introduction

Large Language Models (LLMs), which are scaled from pre-trained language models [1], [2], have demonstrated exceptional performance across various natural language processing tasks, including machine translation [3], text summarization [4], and question answering [5]. Despite these advancements, LLMs' increasing size and complexity pose significant challenges to their application and deployment. For instance, the LLaMA-65B model, containing 65 billion parameters, necessitates 260 GB (twelve RTX 3090Ti GPUs) of VRAM for inference.

Quantization reduces LLM size by converting parameters to low bit-width, thereby improving model inference efficiency and memory utilization [6]–[9]. For example, the weight parameters of the LLaMA-65B model can be compressed to 4-bit integer precision, reducing the model size to 32.5 GB [11]. However, the activations of LLMs are challenging to quantize to low bit-width due to outliers which exhibit small quantities but large magnitudes. These outliers introduce large data distribution, resulting in limited quantization bit-width.

Considering the bottleneck caused by activation outliers, several algorithms have been proposed to smooth the distribution of activations [6]–[9]. However, these algorithm-only methods introduce additional mathematical space transformations that can not be fully leveraged by general computing platforms, creating a mismatch between hardware and algorithm. As a result, hardware and algorithm co-design frameworks have been developed to enhance the benefits of quantization [12]–[15]. These frameworks quantize activation outliers with high data precision and design dedicated hardware accelerators. However, the balance of hardware and algorithm is still not optimally achieved. GOBO [12] and OLAccel [13] treat outliers as sparse matrices which require additional complex control logic. OliVe [14] eliminates neighboring values of outliers to enable localized outlier storage, resulting in significant model accuracy loss at low bit-width quantization. Oltron [15] increases data precision of normal values in outlier channels and permutes channels to maintain regular bit-width for each input tiling block. However, it undergoes low efficiency at nonlinear workloads. Furthermore, the hardware implementations of these approaches face a memory wall challenge due to the memory-bound characteristic of LLMs [11], creating an additional bottleneck.

In this paper, we propose OutlierCIM to address the above challenges. The contributions are listed as follows:

- **Outlier-clustered tiling strategy** that regulates memory access and reduces inefficient workloads. This method overcomes element bit-width variation by clustering all outliers in a single input tiling block, ultimately enhancing data movement and computation efficiency.
- **Hybrid-strategy quantization and reconfigurable double-bit CIM macro array** are designed to address low CIM storage utilization and high latency caused by outlier-based LLM quantization.
- **Quantization factor post-processing strategy and dedicated quantizer** are designed to efficiently perform the multiplication and accumulation of outlier-caused FP-INT workloads with low data bit-width.
- **OutlierCIM** is evaluated at comprehensive benchmarks and achieves up to 4.54× energy efficiency improvement

*The corresponding author: Bo Liu, liubo_cnasic@seu.edu.cn
This work is supported by the National Key Research and Development Program of China under Grant 2023YFB4403103.

and 3.91× speedup compared to baselines. To our best knowledge, **OutlierCIM is the first work to address LLMs acceleration with CIM-based accelerator and outlier-aware quantization algorithm co-design**.

II. Background and Motivation

A. Related Works

Numerous quantization algorithms have been proposed to reduce the memory and computational costs of LLMs by addressing activation outliers [6]–[10]. These methods observe that outliers cluster in specific channels and transfer their extreme magnitudes to weights. However, achieving low bit-width quantization requires complex matrix scaling and transformations [8], [9], which results in low efficiency on general computing platforms. To address this issue, co-design frameworks for digital accelerators and mixed-precision quantization algorithms have been proposed [12]–[15]. These frameworks encode or quantize outliers using large bit-width, enabling near 4-bit LLM inference with dedicated accelerators. However, the balance between hardware and algorithm remains suboptimal. For example, GOBO [12] and OLAccel [13] process outlier-based activation matrices sparsely, leading to complex control logic and irregular memory access. OliVe [14] solves this issue by eliminating neighboring normal values of outliers, but this approach sacrifices model accuracy and introduces element bit-width variation within a single input tiling block. In contrast, Oltron [15] enhances the precision of normal values in outlier channels and regulates memory access through channel-wise permuting. Nevertheless, it encounters inefficiency at post-nonlinear workloads, as only a portion of its processing element (PE) array is capable of FP processing. Moreover, the memory-bound nature of LLMs further restricts the acceleration efficiency of these approaches. Compute-in-memory (CIM) has emerged as a promising solution for memory-bound applications [18]–[20]. However, existing CIM-based accelerators are not optimal for mixed-precision workloads in LLMs, leading to low hardware efficiency.

B. Challenges and Motivation

OutlierCIM seeks to enhance hardware efficiency through a CIM-based accelerator and balance model accuracy with specialized algorithmic strategies. However, several challenges remain to be addressed, as depicted in Fig. 1.

Firstly, large bit-width outliers result in irregular memory access and reduced CIM utilization. On one hand, a significant memory bandwidth is required to maintain a consistent input size. On the other hand, the computation of normal elements must wait additional cycles for outliers due to the bit-serial input nature of the CIM macro. In the case of LLaMA-7B [26] on the WikiText2 [29] dataset, the CIM utilization drops to 58%, and memory energy consumption increases by 23.1%.

Secondly, the outlier-based quantization strategies of current advanced algorithms lead to a conflict between model accuracy and hardware efficiency when processing normal value workloads [7]–[9]. A given 4-bit asymmetric quantized element should be aligned with zero point (ZP) before the multiplication and accumulation (MAC). This operation leads to high input latency and wastes up to 50% storage density due to the expanded bit-width. Although symmetric quantization introduces lower overheads, it fails to maintain model accuracy at low-bit quantization [15].

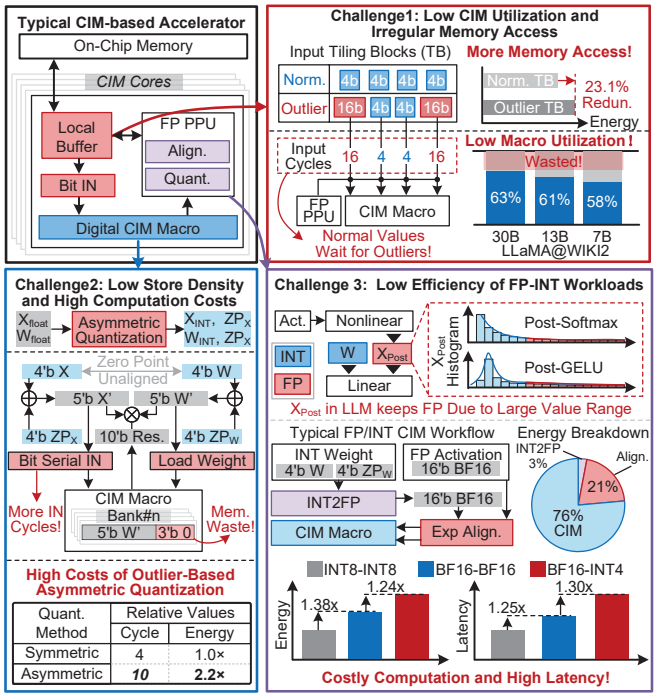


Fig. 1. Typical CIM-based FP/INT accelerator and challenges caused by activation outliers in LLMs

Thirdly, outlier-caused FP-INT workloads incur hardware under-efficiency. Given the expanded activation distribution range caused by outliers, the nonlinear layers in LLMs further aggravate this problem. To maintain model accuracy, the activations of post-nonlinear layers should be kept in FP format [22]. Since the missed support of FP-INT computation in current CIM-based accelerators, the INT weights should be converted to FP format and subsequently perform exponent alignment. In the case of LLaMA-7B@WIKI2, this workflow leads to 1.24× energy cost and 1.30× latency.

III. OutlierCIM Co-design Framework

The overview of the proposed OutlierCIM, a comprehensive CIM-based accelerator and algorithm co-design framework, is presented in Fig. 2. The structure of this paper is organized as follows. Sec. III-A introduces the outlier-clustered tiling strategy, which regulates memory access and reduces outlier workloads. Sec. III-B introduces the hybrid-strategy quantization and reconfigurable double-bit CIM macro array, which are designed to overcome low storage utilization and high latency. Sec. III-C introduces the quantization factor post-processing strategy and dedicated quantizer, which are designed to boost the calculation efficiency of FP-INT workloads.

A. Outlier-Clustered Tiling Strategy

Given the large dimensions of LLMs, input activations must be loaded and computed in a tiling manner. However, high-

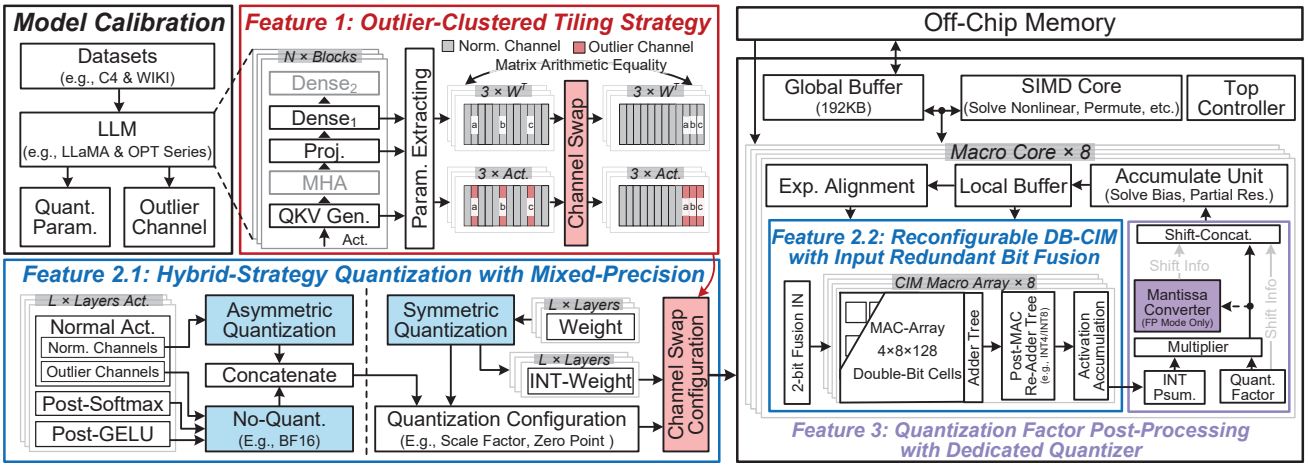


Fig. 2. The overview of proposed algorithm and hardware co-design framework for LLM acceleration

precision outliers significantly increase memory access costs when using the original tiling strategy, illustrated in Fig. 3. OliVe [14] mitigates this issue by nullifying normal values and encoding outliers with large bit-width. However, this tiling strategy is not well-suited for CIM-based architectures. For instance, if a single outlier is quantized using BF16 within a 128-element input vector, the remaining 127 normal values computed in 4 cycles must wait for additional cycles to complete the processing of the current vector.

To address this problem, the Outlier-Clustered Tiling Strategy (OCTS) is proposed, as illustrated in Fig. 3. First, OCTS increases the data precision of normal values within outlier channels. Second, OCTS clusters all outlier channels into the last tiling block of a given activation matrix by swapping outlier channels with the last normal channels. The overall workflow of the proposed OCTS is detailed in Fig. 2 (Feature 1). Outlier channel positions are generated through the model calibration process with the empirical 3σ rule [14]. The activations of the second layer of the Multi-Layer Perceptron (MLP) and the Multi-Head Attention (MHA) layer are not calibrated due to the FP format of post-nonlinear activations [10]. Each tiling block consists of 128 channels, which is sufficiently large to accommodate all outlier channels. As the number of outlier channels may not equal 128, we increase all other normal channels' precision within the last tiling blocks to maintain regular memory access and CIM utilization. Although this strategy introduces additional costs, the small proportion of these last tiling blocks does not significantly affect overall performance (e.g., 128/4096@LLaMA-7B). Finally, the weight dimensions are permuted to ensure the correctness of the general matrix multiplication (GEMM).

The primary advantages of OCTS stem from two key aspects. First, memory access becomes regular compared to the original pattern. Second, low-utilization workloads are eliminated and replaced with uniform data-type tiling blocks. Fig. 4 presents the evaluation of OCTS. The baseline here is a rebuilt bit-serial CIM macro with the original input strategy. OliVe and OCTS tiling strategies are then deployed on the baseline hardware. For OliVe, it fails to reduce latency due

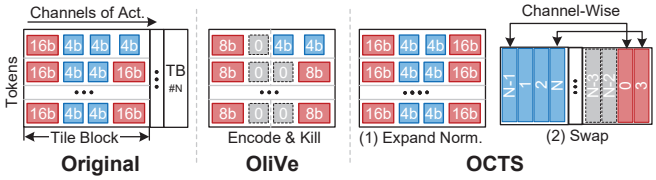


Fig. 3. Comparison of proposed OCTS and other strategies

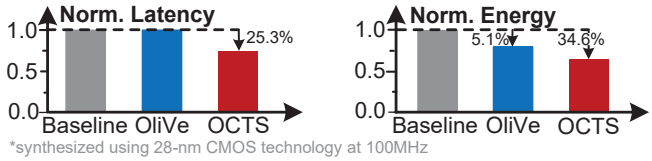


Fig. 4. Performance comparison of outlier-clustered tiling strategy and previous works. The model and dataset here are LLaMA-7B and WikiText2

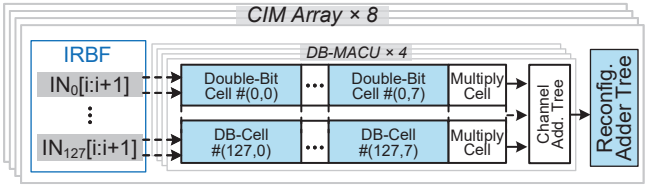
to low CIM utilization caused by variable element bit-width. However, it achieves energy savings from its regular memory access. In contrast, the proposed OCTS reduces latency by 25.3% and achieves up to 34.6% energy savings compared to the baseline. Nevertheless, these improvements are not ideal due to the inefficient INT-INT and FP-INT workloads, which are further addressed in the following sections.

B. Hybrid-Strategy Quantization and Reconfigurable Double-Bit CIM macro array

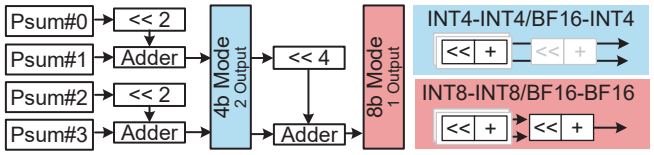
To address the CIM-unfriendly quantization in LLMs, OutlierCIM introduces a Hybrid-Strategy Quantization (HSQ) depicted in Fig. 2 (Feature 2.1). At the weight aspect, we deploy symmetric quantization, which does not require zero point alignment. The quantization and requantization functions are defined as g_w and g_w^{-1} in Eq. 1, where W is the original FP weight matrix, Q_W is the quantized weight, and Δ_W is the scale factor.

$$\begin{aligned} Q_W &= g_w(W) = \text{round}(\frac{W}{\Delta_W}) \\ W &= g_w^{-1}(Q_W) = \Delta_W \times Q_W, \end{aligned} \tag{1}$$

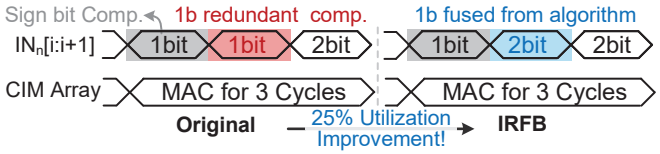
The CIM storage density is therefore improved at 2×. Additionally, the performance loss of symmetric quantization is



(a) Proposed double-bit CIM array architecture



(b) Post-MAC reconfigurable adder tree



(c) Comparation of IRFB and original input strategy

Fig. 5. Detailed double-bit CIM array architecture with reconfigurable adder tree (Re-DBCIM) and input redundant bit fusion strategy (IRBF)

compensated by group-wise weight quantization technology co-design, which is then introduced in Sec. III-C. At the activation aspect, asymmetric quantization is still used to maintain the model performance. The quantization and re-quantization functions are defined as f_x and f_x^{-1}, following:

$$\begin{aligned} Q_X &= f_x(X) = \text{round}(\frac{X}{\Delta_X}) + Z_X \\ X &= f_x^{-1}(Q_X) = \Delta_X \times (Q_X - Z_X), \end{aligned} \tag{2}$$

where X is the original FP activation matrix, Q_X is the quantized activation matrix, Δ_X is the scale factor, and Z_X is the zero point. Notably, the activations of post-nonlinear layers and outlier channels are kept in BF16 format due to their large data distribution range. The quantization scale factors and zero points are generated following the methodology described in OmniQuant [6]. Finally, the quantization configuration is combined with the OCTS strategy to enable regular memory access and high CIM utilization.

Since the expanded bit-width input still introduces high latency, a Double-Bit Reconfigurable CIM macro (Re-DBCIM) array is designed to solve this problem from the hardware aspect. The architecture of Re-DBCIM is shown in Fig. 5 (a), which consists of 4 Double-Bit Cell MAC Units (DB-MACUs). For one DB-MACU, there are 128×8 DB-Cells, which store 2-bit of 128×8 weight matrix. For instance, DB-MACU#0 stores the two most significant bits of a given 128×8 INT4 weight matrix, and DB-MACU#1 stores the remaining two least significant bits. The input 2-bit is then broadcasted to these 2 DB-MACUs, and the MAC operation of each DB-MACU is serially processed by 2-bit multiply-cells and channel-wise adder trees. Subsequently, the MAC results of 2-bit activations and 4-bit weights are generated by weight-wise shift-add. The remaining input bits are then processed similarly and accumulated to operate GEMMs in a tiling manner.

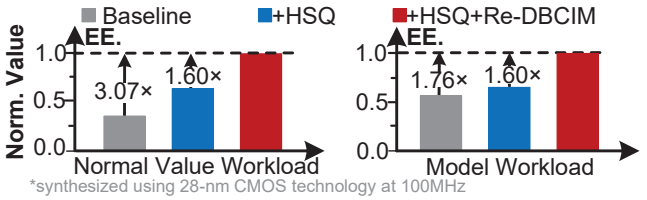


Fig. 6. Energy Efficiency (EE) improvement of gradually deploying proposed algorithm and hardware strategies

To enhance hardware flexibility and expand application scenarios, a reconfigurable adder tree is designed to handle INT8-INT8 or BF16-BF16 workloads, as shown in Fig. 5 (b). At the INT4/BF16-INT4 mode, the first-level shift-add is activated to generate two MAC results as two 128×8 4-bit matrices are stored in the CIM macro array. At the INT8/BF16 mode, all four DB-MACUs store a 128×8 matrix in INT8 or 8b two's complement mantissa format. Subsequently, the second-level shift-add is activated to complete the 8-bit MAC operation, producing one result from the four DB-MACUs.

An Input Redundant Bit Fusion strategy (IRBF) is proposed to address bit-redundancy in the conventional DB-CIM macro [23], as illustrated in Fig. 5 (c). The original DB-CIM macro workflow requires an additional cycle to process the sign bit, resulting in a 25% utilization loss [23]. The IRBF is designed to merge the low-utilization cycle with another bit derived from our proposed hybrid-strategy quantization, where the input is a cardinal bit-width after zero point alignment. Consequently, the utilization of our Re-DBCIM is improved by 25% compared to the conventional DB-CIM.

Fig. 6 illustrates the improvement achieved by gradually deploying the proposed algorithm and hardware strategies. The baseline and model configuration are consistent with Sec. III-A, and OCTS is then deployed for a comprehensive comparison. Compared to the baseline, the HSQ achieves limited energy efficiency improvement due to the high latency caused by expanded input bit-width. The Re-DBCIM is subsequently deployed to address this issue, resulting in a 3.07× energy efficiency improvement at normal value workloads. However, overall model energy efficiency remains constrained by FP-INT workloads, leading to suboptimal results, which is further addressed in Sec. III-C.

C. Quantization Factor Post-Processing Strategy with Dedicated Quantizer

The benefits of quantization strategies are derived from mapping the FP parameters to integers. However, the process of FP-INT workload requires data from these two numerical spaces, resulting in limited efficiency at general computing platforms and conventional FP/INT CIM-based accelerators [20], [21]. The detailed computation process is depicted in Eq. 3, where g_w^{-1} is the requantization function.

$$X_{FP} \times W_{FP} = X_{FP} \times g_w^{-1}(Q_W), \tag{3}$$

The only benefit of the FP-INT process here is low bit-width weight movement. The computation of GEMMs is then operated in the costly FP format.

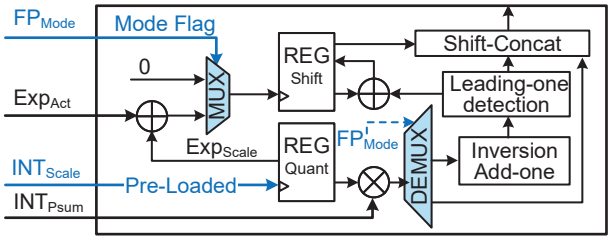


Fig. 7. Proposed quantizer for quantization factor post-processing

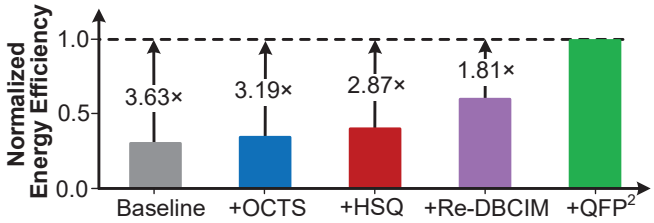


Fig. 8. Improvement breakdown analysis of OutlierCIM

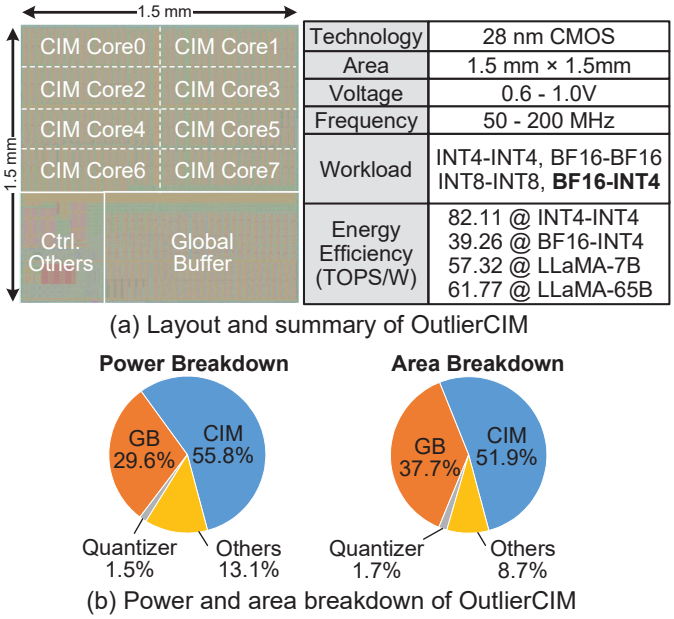


Fig. 9. Layout and breakdown analysis of OutlierCIM accelerator

To this end, the Quantization Factor Post-processing (QFP^2) strategy is proposed to leverage the benefits of integer data movement and computation. We define that the quantized weights are within the integer numerical space $\mathbb{Z}$, while the original weights exist within the floating-point numerical space $\mathbb{R}$. It is evident that the mathematical spaces $\mathbb{Z}$ and $\mathbb{R}$ are linear homomorphism spaces. Given that the mapping function g_w^{-1} defined in Eq. 1 is a linear function, the linear mapping operation $g_w^{-1} \in Hom(\mathbb{Z}, \mathbb{R})$ satisfies both the associative and commutative properties [17]. Consequently, the computation of FP-INT workloads is optimized to Eq. 4:

$$X_{FP} \times W_{FP} = g_w^{-1}(X_{FP} \times Q_W), \tag{4}$$

For the MAC operation of FP and INT, the block float point (BFP) computation method is introduced, and the aligned exponent of INT values is set to 0 to match the BFP [20], [23]. For the map function of g_w^{-1}, QFP^2 follows Eq. 5:

$$\begin{aligned} g_w^{-1}(X_{FP} \times Q_W) = \Delta_W(X_{FP} \times Q_W) &= \frac{A}{2^B}(X_{FP} \times Q_W) \\ &= \frac{A}{2^B}\frac{1}{2^{E_X}}(M_X \times Q_W), \end{aligned} \tag{5}$$

where the phase $\frac{A}{2^B}$ serves as the integer approximation of the scale factor Δ_W, which can be implemented using multiplication and shift operations [24]. Based on the BFP method, E_X, and M_X represent the aligned exponent and two's complement mantissa values of the input X, respectively. Group-wise weight quantization is supported by adjusting Δ_W according to the group size of the weights. Therefore, the MAC operations for FP-INT workloads are efficiently unified and processed in a low bit-width manner.

As shown in Fig. 7, a dedicated quantizer is designed to enhance the efficiency of QFP^2. INT_{Psum} represents the MAC results of two's complement mantissas M_X and quantized weights Q_W from the CIM macro array. The scale factor Δ_W is decomposed to Exp_{Scale} and INT_{Scale} following Eq. 5. Since the proposed CIM array has an accumulation length of 128, group-wise quantization for 128 elements is inherently supported in an efficient hardware configuration. For the FP mode, the MAC results of $M_X \times Q_W$ are multiplied by the scale factor parameter and subsequently perform leading-one detection for shift configuration. Finally, the aligned input exponent, scale factor parameter, and leading-one information are fused to get the standard IEEE 754 float-point format by configuring the shift-concat module. For the INT mode, only scale factor parameters are used to execute the QFP^2 and group-wise quantization strategies.

The main benefits of QFP^2 and quantizer arise from three aspects: 1) the computation of FP-INT workloads is optimized by fully leveraging low bit-width precision; 2) memory access costs are significantly reduced by moving quantized INT4 weights; and 3) model accuracy is maintained by the co-designed group-wise quantization. Fig. 8 demonstrates the improvement breakdown of unified OutlierCIM. The baseline here remains consistent with Sec. III-A. The proposed OutlierCIM achieves an overall 3.63× energy efficiency increase compared to the baseline.

IV. Evaluation

A. Experimental Setup

Algorithm. The algorithm part of OutlierCIM is evaluated on LLaMA series (7B-65B) [26] and OPT series (6.7B-66B) [27]. The perplexity values of language generation tasks are evaluated on WikiText2 [29] and C4 [28]. For hybrid-strategy quantization and group-wise weight quantization, we reconfigure the OmniQuant [6] to our proposed methods.

Hardware. The OutlierCIM accelerator is implemented using commercial 28nm CMOS technology, occupying an area of 2.25 mm^2. A cycle-accurate simulator is developed to simulate the performance of the OutlierCIM accelerator, with energy costs derived from the post-layout simulation.

TABLE I
ALGORITHM RESULTS OF OUTLIERCIM AND PREVIOUS WORKS

Model/PPL↓		LLaMA-7B		LLaMA-13B		LLaMA-30B		LLaMA-65B		OPT-6.7B		OPT-13B		OPT-30B		OPT-66B	
Method	A/W bits	WIKI	C4	WIKI	C4	WIKI	C4	WIKI	C4	WIKI	C4	WIKI	C4	WIKI	C4	WIKI	C4
FP16	A16 /W16	5.68	7.08	5.09	6.61	4.10	5.98	3.53	5.62	10.86	11.74	10.12	11.19	9.56	10.69	9.34	10.28
OliVe	A4/W4	144.78	117.49	42.24	43.13	36.55	33.78	1.4e7	1.8e7	107.15	61.24	416.57	994.64	334.70	572.02	4058.83	2926.87
SmoothQuant	A4/W4	25.25	32.32	40.05	47.18	192.40	122.38	275.53	244.35	1.8e4	1.5e4	7.4e3	5.6e3	1.2e4	8.3e3	2.2e5	1.8e5
Oltron	A4.1/W4	11.67	15.21	8.20	10.84	6.68	8.65	5.82	8.19	12.00	13.02	11.35	12.27	10.51	11.63	10.49	11.05
OmniQuant	A4/W4	11.26	14.51	10.87	13.78	10.33	12.49	9.17	11.28	12.24	13.56	11.65	13.46	10.60	11.89	10.29	11.35
This Work	A4.1/W4	11.83	15.79	8.63	10.93	6.82	8.74	5.97	8.32	12.17	13.15	11.43	12.35	10.64	11.74	10.52	11.23
	A4.1/W4g128	**6.62**	**8.32**	**6.49**	**7.83**	**5.17**	**7.25**	**4.32**	**6.92**	**11.32**	**12.23**	**11.13**	**11.97**	**10.12**	**11.13**	**9.92**	**10.89**

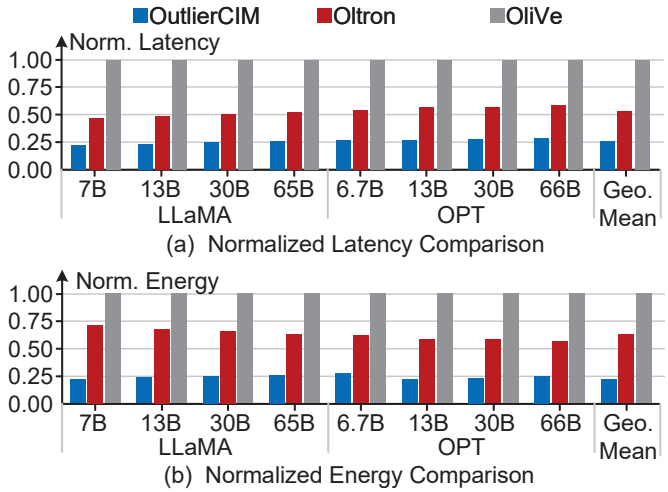


Fig. 10. Comparison of baseline accelerators and OutlierCIM on WikiText2.

Fig. 9 (a) displays the layout and summary of the OutlierCIM accelerator. Notably, the counterpart of the CIM macro array is derived from post-simulation in Virtuoso.

B. Algorithm Result

The comparison between OutlierCIM algorithm and other methods is shown in Table I. The activation bit-width of OutlierCIM is 4.1-bit, as partial activations are stored in BF16 format, following the same definition as Oltron [15]. The perplexity of OutlierCIM without group-wise weight quantization is slightly higher than that of Oltron due to the loss introduced by symmetric weight quantization. However, the co-design framework of OutlierCIM with group-wise weight quantization surpasses the baselines across all benchmarks.

C. Hardware Result

Detailed Configuration. The detailed configuration of OutlierCIM is shown in Fig. 9 (a). The area and power breakdown are presented in Fig. 9 (b).

Performance. For a fair comparison of different architectures, the throughput values of the baseline accelerators are scaled to the same level, following the methodology described in [30]. Fig. 10 (a) highlights the superior performance of OutlierCIM, which achieves an average speedup of 2.06× and 3.91× compared to Oltron and OliVe, respectively.

Energy. Fig. 10 (b) displays the energy comparison between OutlierCIM and the baseline accelerators. OutlierCIM achieves lower energy consumption than OliVe and Oltron while delivering higher performance. Specifically, OutlierCIM achieves an average energy efficiency improvement of 4.54× and 2.83× compared to OliVe and Oltron, respectively.

TABLE II
COMPARISON OF OUTLIERCIM AND PREVIOUS WORKS

	OliVe ISCA'23 [14]	Oltron DAC'24 [15]	ReDCIM JSSC'23 [20]	This Work
Architecture	Digital	Digital	Digital CIM	Digital CIM
Technology	22nm	28nm	28nm	28nm
Voltage	-	-	0.6-1.0V	0.6-1.0V
Frequency	-	-	50-220 MHz	50-220 MHz
Workload	Encoded INT4-INT4	Encoded MFP-INT4 INT4-INT4	FP32-FP32 INT16-INT16 BF16-BF16 INT8-INT8	INT4-INT4 **BF16-INT4** BF16-BF16 INT8-INT8
Power	-	-	12.5-69.4 mW	24.5-184.3 mW
Energy Efficiency (TOPS/W)	12.31*1	19.32*1	36.5@INT8-INT8*2 9.10@INT16-INT16 29.2@BF16-BF16 3.70@FP32-FP32	82.11@INT4-INT4 39.26@BF16-INT4 **57.32@LLaMA-7B** **61.77@LLaMA-65B**

*1: Results are derived from respective papers and work [16].
*2: Results are quoted from its paper which include sparsity benefits.

Comparison with State-of-the-Art Works. Table II provides a detailed comparison with state-of-the-art works. ReDCIM [20] is a unified FP/INT CIM-based accelerator, while OliVe [14] and Oltron [15] are digital outlier-aware accelerators. Compared to the baseline CIM-based accelerator, OutlierCIM supports additional computation paradigms and achieves superior energy efficiency for LLM workloads. Moreover, OutlierCIM demonstrates top-tier energy efficiency compared to the baseline outlier-aware accelerators.

V. CONCLUSION

This paper proposes OutlierCIM, the first framework to address LLMs acceleration with a CIM-based accelerator and outlier-aware quantization algorithm co-design. The key insights of OutlierCIM are: 1) At the hardware aspect, a reconfigurable double-bit CIM array is designed to overcome the low efficiency of previous accelerators. 2) At the algorithm aspect, a hybrid-strategy quantization algorithm and a map strategy are proposed to boost the efficiency of the CIM array at workloads caused by outliers. Combining these two parts, the proposed OutlierCIM achieves up to 4.54× energy efficiency and 3.91× speedup compared to baseline accelerators at comprehensive benchmarks.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," Proceedings of the NAACL-HLT, 2019, pp. 4171-4186.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, et al., "Language models are unsupervised multitask learners," OpenAI blog, vol. 1, no. 8, 2019, p. 9.

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, arXiv:1409.0473, [Online]. Available: https://arxiv.org/abs/1409.0473

[4] R. Nallapati, B. Zhou, C. N. dos Santos, C. Gulcehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence rnns and beyond," 2016, arXiv:1602.06023. [Online]. Available: https://arxiv.org/abs/1602.06023

[5] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on freebase from question-answer pairs," Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1533-1544.

[6] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, et al., "OmniQuant: Omnidirectionally calibrated quantization for large language models," 2023, arXiv:2308.13137. [Online]. Available: https://arxiv.org/abs/2308.13137

[7] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," Proceedings of the 40th International Conference on Machine Learning, vol. 202, 2023, pp. 38087–38099.

[8] Y. Ma, H. Li, X. Zheng, F. Ling, X. Xiao, et al., "AffineQuant: Affine transformation quantization for large language models," 2024, arXiv:2403.12544. [Online]. Available: https://arxiv.org/abs/2403.12544

[9] H. Lin, H. Xu, Y. Wu, J. Cui, Y. Zhang, et al., "DuQuant: Distributing outliers via dual transformation makes stronger quantized LLMs," 2024, arXiv:2406.01721. [Online]. Available: https://arxiv.org/abs/2406.01721

[10] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale," 2022, arXiv:2208.07339. [Online]. Available: https://arxiv.org/abs/2208.07339

[11] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, et al.,"A Survey on efficient inference for large language models," 2024, arXiv:2404.14294. [Online]. Available: https://arxiv.org/abs/2404.14294

[12] A. H. Zadeh, I. Edo, O. M. Awad and A. Moshovos, "GOBO: Quantizing attention-based NLP models for low latency and energy efficient inference", Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), 2020, pp. 811-824.

[13] E. Park, D. Kim and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation", in Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA), 2018, pp. 688-698.

[14] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, et al., "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization", in Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA), 2023.

[15] C. Xue, C. Zhang, X. Jiang, Z. Gao, Y. Lin, et al., "Oltron: Software-Hardware Co-design for Outlier-Aware Quantization of LLMs with Inter-/Intra-Layer Adaptation", 61st ACM/IEEE Design Automation Conference (DAC), 2024.

[16] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating large language models via tensor decomposition and runtime requantization," 51st International Symposium on Computer Architecture (ISCA), 2024, pp. 1048-1062.

[17] Greub W-H, "Linear algebra", Springer Science & Business Media, 2012, pp. 16-17.

[18] J. Chen, F. Tu, K. Shao, F. Tian, X. Huo, et al., "AutoDCIM: An Automated Digital CIM Compiler," 2023 60th ACM/IEEE Design Automation Conference (DAC), 2023.

[19] X. Xiang, Z. Yue, Y. Li, L. Lv, S. Wei, et al., "Dyn-Bitpool: A Two-sided Sparse CIM Accelerator Featuring a Balanced Workload Scheme and High CIM Macro Utilization", 2024 61th ACM/IEEE Design Automation Conference (DAC), 2024.

[20] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, et al., "ReDCIM: Reconfigurable digital computing-in-memory processor with unified FP/INT pipeline for cloud AI acceleration", IEEE J. Solid-State Circuits (JSSC), vol. 58, no. 1, 2023, pp. 243-255.

[21] J. Jang, Y. Kim, J. Lee and J. Kim, "FIGNA: Integer unit-based accelerator design for FP-INT GEMM preserving numerical accuracy", 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2024, pp. 760-773.

[22] X. Hu, Y. Cheng, D. Yang, Z. Yuan, J. Yu, et al., "I-LLM: Efficient Integer-Only inference for Fully-Quantized Low-Bit large language models,"2024, arXiv:2405.17849. [Online]. Available: https://arxiv.org/abs/2405.17849

[23] A. Guo, C. Xi, F. Dong, X. Pu, D. Li et al., "A 28nm 64-kb 31.6-TFLOPS/W digital-domain floating-point-computing-unit and double-bit 6T-SRAM computing-in-memory macro for floating-point CNNs", IEEE Int. Solid-State Circuits Conf. (ISSCC), 2023, pp. 128-130.

[24] Z. Li and Q. Gu, "I-ViT: Integer-only quantization for efficient vision transformer inference," Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 17065–17075.

[25] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey et al., "IEEE standard for floating-point arithmetic." IEEE Std 754.2008, 2008, pp. 1-70.

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M-A Lachaux et al., "LLaMA: Open and efficient foundation language models," 2023, arXiv:2302.13971. [Online]. Available: https://arxiv.org/abs/2302.13971

[27] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, et al., "OPT: Open Pre-trained transformer language models," 2022, arXiv:2205.01068. [Online]. Available: https://arxiv.org/abs/2205.01068

[28] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," Journal of machine learning research, vol. 21, 2020, pp. 1-67.

[29] S. Merity, C. Xiong, J. Bradbury, R. Socher, "Pointer sentinel mixture models," arXiv:1609.07843, 2016, [Online]. Available: https://arxiv.org/abs/1609.07843.

[30] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "DOTA: Detect and omit weak attentions for scalable transformer acceleration," Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2022, pp. 14-26.

SSFT: Algorithm and Hardware Co-design for Structured Sparse Fine-Tuning of Large Language Models

Miao Yu
National University of Singapore
miao.yu@u.nus.edu

Trevor E. Carlson
National University of Singapore
tcarlson@comp.nus.edu.sg

***Abstract*—A significant number of users depend on Large Language Models (LLMs) for downstream tasks, but training LLMs from scratch remains prohibitively expensive. Sparse fine-tuning (SFT) has emerged as an effective strategy to reduce both the time and memory requirements of fine-tuning LLMs, achieving accuracy on par with fully fine-tuned models. Although SFT has the potential to achieve superior performance by minimizing computational requirements, SFT on GPUs often underperforms compared to dense algorithms like LoRA due to sparse data accesses that modern GPUs cannot efficiently handle. To address these issues, we propose Structured Sparse Fine-Tuning (SSFT). It comprises a novel algorithm, SSFT-Alg, which introduces predictable sparsity patterns to reduce memory access overhead and enhance regularity in the SFT process. To support SSFT-Alg, an accelerator, SSFT-Hw, is proposed to optimize SSFT-Alg through an innovative sparsity-aware design, avoiding the overhead of sparsity operations on GPUs and optimizing latency and energy efficiency. Experiments with relevant models and benchmarks demonstrate that SSFT achieves comparable accuracy to state-of-the-art models on BERT, LLaMA 2 7B, and LLaMA 2 13B. Moreover, SSFT-Hw outperforms both GPUs and the state-of-the-art sparsity-aware transformer accelerators in throughput by $51.0\times$ and $1.32\times$, respectively, while improving energy efficiency by $19.0\times$ and $1.48\times$.**

I. Introduction

The scale of Large Language Models (LLMs), such as BERT [8], GPT-4 [1], and LLaMA [22], is fundamental to their cutting-edge performance. Adopting these models on downstream tasks is a universal demand, but the training cost makes it inaccessible for most users. Fine-tuning is the most practical approach for general users and companies to improve the accuracy of LLMs for these downstream tasks. However, fully fine-tuning these large models is hindered by their extensive memory and time requirements. Recent advancements in fine-tuning methods, particularly those employing Sparse Fine-Tuning (SFT) strategies, have demonstrated significant promise in reducing the memory requirements and latency while retaining accuracy [3], [4], [21]. These methods improve the balance between a minimal number of parameters and superior fine-tuning accuracy without introducing additional parameters into the LLM architectures like LoRA [11]. This approach effectively eliminates extra memory and computation overhead in practical applications.

Although SFT algorithms exhibit significant promise to minimize the number of parameter updates compared with dense fine-tuning techniques [12], their unique sparsity characteristics pose compatibility issues with contemporary processors, such as GPUs and Transformer accelerators, due to the irregularity in gradient calculation, the most time- and energy-intensive aspect of LLM fine-tuning. Specifically, as shown in Fig. 1, SFT algorithms either dynamically or statically calculate masks from the information derived from previous training steps. These masks are then applied to the gradients to update the current parameters. According to the principle of SFT algorithms, the bulk of gradients do not contribute to weight updates, leading to a large number of unnecessary memory accesses and MAC operations, given that a substantial majority of parameters ($> 95\%$) are frozen. Therefore, efficiently managing the masked gradients is key to improving the performance of SFT. However, neither the GPUs nor the state-of-the-art sparsity-aware transformer accelerators can efficiently manage the irregularity and high sparsity of masked gradients. As depicted in Fig. 1, computing the gradients matrix $\frac{\partial L}{\partial W}$ for parameter matrix W requires 16 memory accesses and 64 MAC operations but 12 accesses and 48 MAC operations are unnecessary for this algorithm, resulting in inefficiencies in both latency and energy consumption.

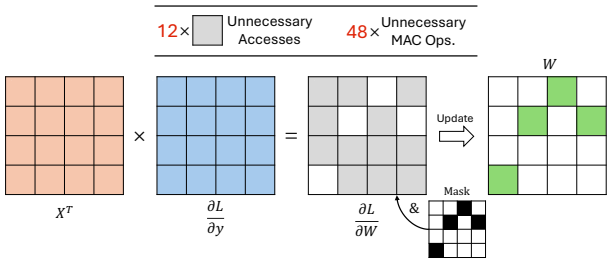


Fig. 1. An example of weight gradient generation and weight update stages with the previous sparse fine-tuning algorithm. While all weight gradients are calculated, only 4 are utilized to update weights, leading to 12 unnecessary accesses and 48 unnecessary MAC operations. SSFT can eliminate unnecessary accesses and operations.

The design of current accelerators, particularly those for Transformers (the foundational architecture of LLMs), fails to efficiently support sparsity throughout fine-tuning [20], [23], [24], [28]. The state-of-the-art sparsity-aware transformer accelerator, TransCODE [24], demonstrates limitations in handling LLM fine-tuning with high sparsity because of the

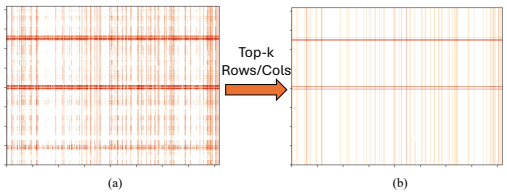


Fig. 2. (a) The distribution of top-k gradients in the parameters of Attention-Q in the first layer of LLaMA 2 7B. Similar structures are observed in crucial parameter matrices across Transformer-based LLMs like BERT and LLaMA. (b) SSFT selects the top-k rows and columns of crucial parameters to generate the structured mask of Attention-Q.

high energy cost and latency of processing irregular sparse data and the requirement to load all weights into the on-chip buffer for computation. Even though the design has proven its effectiveness for mid-size language models such as $\text{BERT}_{\text{TINY}}$ [25](∼4M params) and $\text{BERT}_{\text{BASE}}$ [8](∼110M params), it still struggles with larger size models such as $\text{BERT}_{\text{LARGE}}$ [8] (∼340M params) or LLaMA 2 [22] ($>$ 7B params) due to the high memory requirements. Other designs, such as accelerators for SpM*SpM [19], fail to accelerate Transformer-based models adequately due to their inability to perform specific Transformer operations like Softmax and layer normalization [20].

A key insight is that *previous work has not been able to take advantage of structured information in SFT,* given that structured sparsity is recognized for its hardware compatibility in AI workloads [5], [6], [10]. To bridge the gap between SFT algorithms and hardware, we present Structured Sparse Fine-Tuning (SSFT), a collaborative approach at the algorithm (SSFT-Alg) and hardware levels (SSFT-Hw). Fig. 2(a) illustrates the distribution of original top-k parameters based on gradients, revealing row and column patterns that previous work has overlooked. Building on this observation, SSFT-Alg structurally selects rows and columns containing a top-k number of values, shown in Fig. 2(b), while maintaining comparable accuracy. This is because structured sparsity retains most of the essential information from the original crucial parameters. Meanwhile, given that structured sparsity is recognized for its hardware compatibility in AI workloads [5], [6], [10], attributable to uniform memory access patterns and computation, it gives SSFT-Alg the opportunity to surpass the performance and energy efficiency of unstructured algorithms.

To leverage the potential of SSFT-Alg and address the inefficiencies of existing hardware in handling high sparsity, we propose SSFT-Hw, an accelerator designed with highly parallel dataflow and sparsity-aware modules that bypass unnecessary gradient computations and flexibly handle various stages of the fine-tuning process. In addition, SSFT-Hw can process Softmax and layer normalization operations, which are not supported by general SpM*SpM accelerators. This hardware is the first accelerator that can support sparse fine-tuning on LLMs, providing higher efficiency than GPUs and existing accelerators in handling high-sparsity fine-tuning. The main contributions of our work follow.

- We introduce a novel structured sparse fine-tuning algorithm called SSFT-Alg. Based on the inherent structural characteristics in parameter matrices, SSFT-Alg selects the top-k important rows or columns in the parameter matrix based on the gradient during the sampling process, establishing a pattern that allows for uniform access and processing of gradients while maintaining accuracy.
- We propose the SSFT-Hw accelerator, which leverages the sparse masks resulting from SSFT-Alg with a dataflow providing high parallelism and sparsity-aware modules. This design enables SSFT-Hw to achieve high performance and a low memory footprint with the regularity provided by SSFT-Alg.
- SSFT-Alg is verified on both BERT and LLaMA 2, demonstrating accuracy comparable to that of current algorithms. SSFT-Hw achieves $1.32\times$ higher throughput and $1.48\times$ greater energy efficiency compared to the SOTA while maintaining comparable accuracy.

TABLE I
DETAILED PROCESSES FOR LLM FINE-TUNING. VARIABLES INCLUDE L (LOSS), X (INPUT ACTIVATIONS), Y (OUTPUT ACTIVATIONS), W (WEIGHTS), γ (LEARNING RATE), AND W^* (UPDATED WEIGHTS).

Forward	$X \times W = Y$
Backward	$\frac{\partial L}{\partial Y} \times W^T = \frac{\partial L}{\partial X}$
Weight Gradient	$X^T \times \frac{\partial L}{\partial Y} = \frac{\partial L}{\partial W}$
Weight Update	$W - \gamma * \frac{\partial L}{\partial W} = W^*$

II. BACKGROUND AND MOTIVATION

A. Fine-tuning process of LLMs

Taking stochastic gradient descent, one of the standard fine-tuning algorithms for LLMs, as the example, it comprises four stages, illustrated in Table I:

- The forward stage runs the inference algorithm to determine the model's predictions for training inputs and calculate the loss L (i.e., the training error). This consists of convolving the input tensor X with a set of weights W to obtain the output tensor Y.
- The backward stage propagates the loss gradient across the model's layers. This is done by convolving the loss gradient with respect to the output gradients $\frac{\partial L}{\partial Y}$ with transposed weights W^T.
- The weight gradient generation stage calculates the gradients that are utilized in updating weights. This consists of convolving the backpropagated loss gradients with respect to the gradient $\frac{\partial L}{\partial Y}$ with the transposed input X^T.
- The weight update stage determines how much weight should be decreased by computing the multiplication of learning rate γ and gradients $\frac{\partial L}{\partial W}$.

B. Motivation

To further illustrate the suboptimal performance of the SFT algorithm on GPUs, we compare SFT with LoRA (set to rank 64 to achieve the same sparsity as SFT) in terms of the number of operations, latency, and efficiency (ops/cycle). As shown in Fig. 3, despite original SFT processes 41.6% fewer number of operations than LoRA, which has the potential to outperform LoRA on GPUs, its latency and efficiency

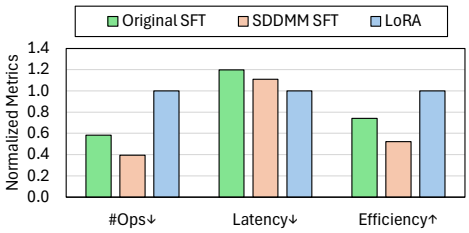


Fig. 3. Comparison of original SFT, SDDMM SFT, and LoRA in terms of number of operations, latency, and efficiency (ops/cycle). All metrics are normalized to LoRA. ↑ indicates that taller bars are better, while ↓ indicates the opposite. While the SDDMM version of SFT reduces 32.6% #Ops, it reduces latency by only 7.2% and decreases efficiency by 25.7%.

is 20.0% and 25.9% lower, respectively. Then we optimize the core operation, gradient generation, in SFT with a more efficient implementation called Sampled Dense-Dense Matrix Multiplication (SDDMM) [16], which is supported by the NVIDIA cuSPARSE library. Enhanced with SDDMM, GPUs calculate only the necessary weight gradients instead of the entire weight gradient matrix. Despite SDDMM reducing 32.6% operations, as shown in Fig. 3, total latency decreases less than only 7.2% over the original SFT, and the efficiency even becomes worse. This decline is due to low-computation-density operations such as gather_scatter and topk in SFT prevent optimal performance. Specifically, gather_scatter and topk account for 32.2% of cycles but only 0.8% of operations, which significantly harms efficiency. Therefore, based on current GPUs and optimization libraries like cuSPARSE, the potential of SFT cannot be fully realized. Both an efficient algorithm and a customized accelerator capable of efficiently processing SFT are critically required.

III. Structured Sparse Fine-Tuning

One of the main contributions of this paper is a method for pre-computing a structured sparse subset of a model's parameters to update over many subsequent training iterations. To construct such a subset, we calculate the average of gradients as a signal of how important the parameter is for a given task during sampling. We then select the top-k rows and columns from the gradient matrix to create a subset of it. We refer to the resulting subset (i.e., a list indicating which parameters are included in the subset) as a $mask$. In this section, we detail the steps necessary for computing a structured sparsity mask. This process is diagrammed in Fig. 4 and listed in Algorithm 1.

A. Crucial Parameter Detection

Unlike previous works that utilize complex and time-consuming methodologies such as Fisher information [21], lottery-ticket [2], and delta vectors [3], SSFT-Alg adopts a static top-k gradient selection method to reduce running overhead while maintaining accuracy. The first step of SSFT-Alg is to select the subset of parameters that are crucial to update. One way to generate this subset is to consider how much each of the parameters will be impacted by the training data. In the general training process of neural networks, gradients are utilized to update the parameters with different

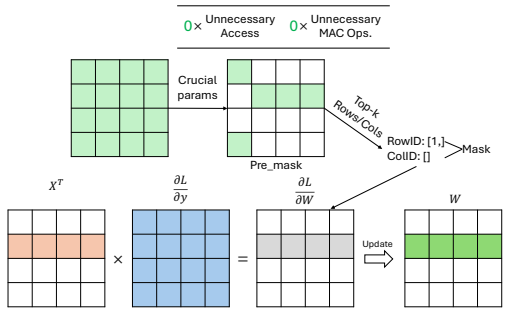


Fig. 4. An example of SSFT-Alg involves calculating trainable parameters of 4×4 weight matrix. SSFT-Alg selects the crucial parameters based on gradients after sampling. It selects rows or columns with the highest number of crucial parameters, incorporating them into a list of trainable elements. This list, comprising all trainable rows and columns, is employed during gradient computation to minimize unnecessary access and processing.

Algorithm 1 Structured Sparse Fine-Tuning Algorithm

1. **Input:** X, $W_{original}$, `sampling_steps`, `sparsity`
2. **Output:** W
3. *// Crucial Parameter Detection*
4. **for** $i = 1$ to `sampling_steps` **do**
5. $loss \leftarrow$ Forward$(X[i], W)$
6. $grad_X \leftarrow$ Backward$(W, loss, grad_Y)$
7. $grad_W \leftarrow$ Weight_Grad$(X[i], W, grad_Y)$
8. $W \leftarrow$ Weight_Update$(W, grad_W)$
9. $grad_mask \leftarrow grad_mask + grad_W$
10. **end for**
11. $W \leftarrow W_{original}$
12. $grad_mask \leftarrow grad_mask$/`sampling_steps`
13. **for all** l in layers **do**
14. pre_mask$[l] \leftarrow$ top-k$(grad_mask[l],$ `sparsity`$)$
15. **end for**
16. *// Mask Generation*
17. **for all** mkl in pre_mask **do**
18. **for all** row in mkl **do**
19. row_grad$[row] \leftarrow$ count_nonzero(row)
20. **end for**
21. **for all** col in mkl **do**
22. col_grad$[col] \leftarrow$ count_nonzero(col)
23. **end for**
24. mask$[mkl] \leftarrow$ top-k$([$row_grad, col_grad$],$ `sparsity`$)$
25. **end for**
26. *// Weight Gradient Generation and Weight Update*
27. **for** iter $= 1$ to `training_steps` **do**
28. $loss \leftarrow$ Forward$(X[i], W)$
29. $grad_X \leftarrow$ Backward$(W, loss, grad_Y)$
30. $grad_W \leftarrow$ Weight_Grad$(X[i], W, grad_Y, mask)$
31. $W \leftarrow$ Weight_Update$(W, grad_W, mask)$
32. **end for**
33. **return** W

types of optimizers. As lines 4-9 show, SSFT-Alg takes the $sampling_steps$ as a parameter to decide the number of steps required by sampling. The parameters are updated in the same way as done in the general training process. The gradients of each of the parameters are then accumulated. After collecting all gradients, weights are reset to the original weights to avoid the impact of the sampling step. After completing the sampling process, the gradient matrices of each layer are scanned. The top-k gradients are selected from the matrices based on the

sparsity requirement from the input. These generated masks are named *pre-masks*. Despite current SFT algorithms that can generate or detect crucial parameters, they leave irregularity and dynamic detection overhead that harm both performance and memory access. In the next section, we describe how SSFT-Alg takes this data to introduce structured sparsity.

B. Mask Generation

Recall that our goal is to get subsets of trainable parameters in each layer with structured sparsity to update over many iterations of training while keeping the remainder of the parameters fixed. SSFT-Alg scans all pre-masks of each layer horizontally and vertically. During the scan process, SSFT-Alg counts non-zero values of each row and column (lines 17-25). After scanning all pre-masks, SSFT-Alg selects rows or columns with the top-k number of non-zero values as the final masks. Fig. 2 shows an example of this step. Intuitively, the pre-mask with irregular distribution is converted to a structured distribution with more regularity. Taking Fig. 4 as an example, there are 5 parameters selected as the crucial parameters. Two of these values are in the first column and three of these values are in the second column. If k is set to 1, which means 1 row or column with the maximal number of values is selected, the second row is selected as the *mask*. The mask generation stage of SSFT-Alg can be utilized not only with our crucial parameter detection method but also can cooperate with any of the mask selection methodologies as a plug-in, like Fish [21], Delta [3], etc.

C. Weight Update

After generating the structured sparse masks, these masks are utilized in the weight-updating stage. As shown at the bottom of Fig. 4, the gradients are computed based on the generated masks. Specifically, only rows and columns recorded in the list are required to be computed in the weight-updating stage. Therefore, SSFT-Alg only needs to access specific rows or columns of input matrices, which are X^T and $\frac{\partial L}{\partial Y}$, to generate necessary gradients ($\frac{\partial L}{\partial W}$). Furthermore, to update the weight matrix with the gradients of each weight, only specific rows and columns are updated.

IV. Microarchitecture

Fig. 5(a) shows an overview of SSFT-Hw. Two critical sparsity-aware modules, Memory Fetcher and Sparse-Dense Scheduler, are highlighted for clearer visualization of their details. Data is brought to the on-chip buffers from DRAM by the direct memory access (DMA) controller. Memory Fetcher fetches distinct types of data according to the stages and transfers data to the on-chip buffer. It also sends information about mask and sparse-dense mode to the Sparse-Dense Scheduler. The buffers store the activations, gradients, and weights, respectively. Sparse-Dense Scheduler passes data from on-chip buffers to the PE array. The PE array uses the compressed data and compression information in the $mask$ to perform the main compute operations in the LLM fine-tuning.

A. Sparsity-aware Design

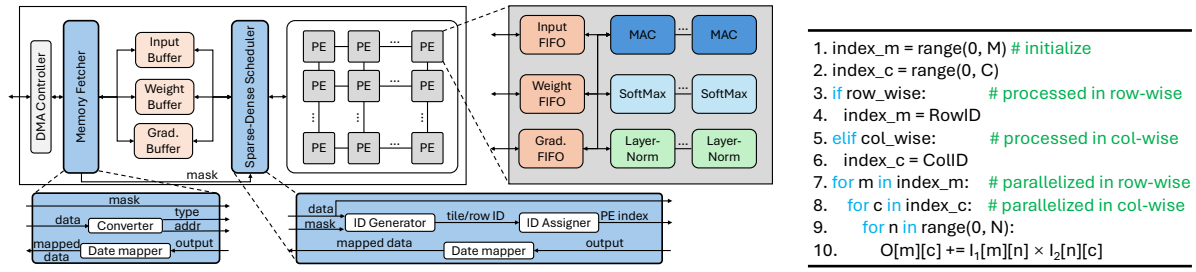
Due to irregularity and limited sparsity support, existing hardware platforms struggle with sparse fine-tuning. To efficiently support the structured sparse fine-tuning algorithms, two sparsity-aware modules, Memory Fetcher (Fetcher) and Sparse-Dense Scheduler (Scheduler), are introduced into the SSFT-Hw. Combined with a novel structured sparse dataflow shown in Fig. 5(b), Fetcher and Scheduler enable SSFT-Hw to process SSFT-Alg with high efficiency. There are four steps in fine-tuning LLMs: forward, backward, weight gradient generation, and weight update. SSFT-Hw processes weight gradient generation and weight update with sparse mode and forward and backward with dense mode.

In the sparse mode, to address the irregularity and sparsity issues shown in Fig. 1, Fetcher first takes the $mask$ generated by Algorithm 1 as the additional input, along with transposed input X and gradient $\frac{\partial L}{\partial Y}$ from previous layer in the weight gradient generation stage. In $mask$, gradients to be updated are represented in a structured way, which stores only row and column indices to avoid the overhead of unstructured data storage in a general compressed format. Fetcher parses data from DRAM based on the information in $mask$. Parsed values are stored separately in on-chip buffers. $Mask$ is then passed to the Scheduler for further processing. Scheduler reads data stored in on-chip buffers according to the information in $mask$, assigning data to different PEs based on row/column IDs for parallel processing, as shown at the bottom of Fig. 5(a). SSFT-Hw generates output matrix row by row or column by column following the information in $mask$. This also means that only several rows or columns of the input matrix (matrices X^T and $\frac{\partial L}{\partial Y}$) are required during the computation. After completing the computation of gradients, Fetcher fetches the relevant weights and updates weights with the structurally masked gradients without any unnecessary computation and memory access. In dense mode, SSFT-Hw performs dense-dense matrix multiplication during forward and backward stages in a tiled approach, a technique widely used in current accelerators [24].

The main modules present inside a PE are shown on the right of Fig. 5(a), which is the basic compute block in our accelerator. The PE design is based on an open-source design [24]. The data are stored in local registers of the PE by the input first-in-first-out (FIFO), weight FIFO, and gradient FIFO registers. PE takes the structured data and data information from DRAM. The PE then forwards data and information to the MAC lanes (for matrix multiplication), SoftMax modules (for SoftMax operation), or the Layer-Norm module (for layer normalization operation). MAC lanes and SoftMax modules can be utilized simultaneously with the schedule of the Scheduler to achieve high utilization. Finally, PE will write the results into input FIFO and then write the results back to the DRAM.

B. Dataflow

As shown in Fig. 5(b), a novel dataflow is proposed to enable high parallelism in SSFT-Alg. An $M \times N$ matrix



(a) Hardware architecture

```
1. index_m = range(0, M) # initialize
2. index_c = range(0, C)
3. if row_wise:          # processed in row-wise
4.   index_m = RowID
5. elif col_wise:        # processed in col-wise
6.   index_c = ColID
7. for m in index_m:     # parallelized in row-wise
8.   for c in index_c:   # parallelized in col-wise
9.     for n in range(0, N):
10.      O[m][c] += I1[m][n] × I2[n][c]
```

(b) Dataflow

Fig. 5. (a) The hardware architecture of SSFT-Hw consists of the DMA controller, Memory Fetcher, on-chip buffers, Sparse-Dense Scheduler, and PE array. Each PE contains three types of FIFOs to store inputs, weights, and gradients. These FIFOs are connected with three types of computing units: MAC units, SoftMax units, and layer normalization units. (b) The dataflow used in SSFT-Hw to process the sparse weight gradient generation. Row or column are processed by a single PE in parallel.

TABLE II
SPECIFICATION OF SSFT-HW.

Module	Configuration	Module	Configuration
DRAM Bandwidth	256GB/s	LayerNorm Units	1/PE
PEs	512	Act. Buffer	16MB
MAC Units	32/PE	Weight Buffer	64MB
SoftMax Units	32/PE	Grad. Buffer	16MB

(I_1) is multiplied by an $N \times C$ matrix (I_2) to generate an $M \times C$ matrix (O). Rows or columns to be processed are specified by array RowID or ColID (line 3 to line 6), which store mask information as shown in Fig. 4. Each row or column is processed by a dedicated PE in parallel, maximizing parallelism (lines 7 and 8). Consequently, each PE outputs a complete row or column (lines 9 and 10). This dataflow maximizes resource utilization, achieving high efficiency by eliminating zero values and irregularity.

V. EVALUATION

A. Experiment Setup

1) Algorithm: We demonstrate that our approach matches the performance of standard gradient-based training on the GLUE benchmark [26] while updating only 5% of the model's parameters per task. All experiments for GLUE are run with the $\text{BERT}_{\text{LARGE}}$ variant of BERT [8]. Sampling_step is 5.

To demonstrate the effectiveness of SSFT-Alg on larger models, we instruction-tune LLaMA 2 7B and 13B on Flan v2 [14] and GPT4-Alpaca [17] with 2.3% density mask. We evaluate instruction-tuned LLMs on the following benchmarks to capture a range of abilities: MMLU [10], TyDiQA [6], and HumanEval [5]. Furthermore, we compare SSFT-Alg with LoRA [11], as it offers the best performance–efficiency trade-off and is arguably the most widespread pre-existing PEFT method [7], as well as IA^3 [13]. We use the LoRA and IA^3 implementations in the PEFT library [15]. Experiments of BERT and LLaMA 2 7B are run on an NVIDIA A100-40GB and LLaMA 2 13B is run on an NVIDIA H100-96GB.

2) Hardware: To have a fair comparison with the state-of-the-art work, TransCODE, we set our hardware configuration the same. Table II shows the associated design choices. The RTL model is synthesized using Synopsys Design Compiler version P-2019.03, targeting the 14-nm FinFET technology library [9], which provides power and area information in the synthesis report. We fixed the clock rate to 700 MHz based on the delay of all modules in the proposed architecture. We set the number of multipliers to 16 in each PE. We run the simulation to get the latency based on the open-source simulator proposed by TransCODE [24] along with our hardware modules. We compare the performance of SSFT-Hw with many previously proposed accelerators.

TABLE III
GLUE TEST SERVER EVALUATION RESULTS WITH $\text{BERT}_{\text{LARGE}}$. TRAINING WITH THE SSFT-ALG GAINS COMPETITIVE ACCURACY WHILE UPDATING ONLY 5% OF MODEL PARAMETERS.

Method	FullFT	Bit Fit	Diff Pruning	Monarch	Fish Mask	SSFT-Alg
QNLI	93.4	90.4	91.9	87.0	93.3	**93.1**
SST-2	94.0	94.5	93.8	92.4	93.6	**93.1**
MNLImm	87.0	84.8	85.5	82.3	85.6	**86.0**
CoLA	66.0	60.3	61.0	59.6	64.8	**64.1**
MRPC	88.8	86.3	86.2	90.1	89.6	**89.9**
STS-B	90.6	85.0	85.6	88.3	90.6	**90.1**
RTE	73.6	69.6	67.5	75.0	75.4	**75.4**
QQP	91.2	78.5	80.1	87.7	89.1	**88.6**
AVG	85.6	81.2	81.5	82.8	85.2	**85.0**

B. Accuracy of SSFT-Alg

1) BERT: Our results on SSFT-Alg can be seen in Table III. SSFT-Alg training results in effectively the competitive accuracy (85.0%) with SOTA SFT [21] (85.2%) and standard "dense" fine-tuning (85.6%), despite updating just 5% of $\text{BERT}_{\text{LARGE}}$ parameters. Specifically, SSFT-Alg reduces maximally 0.7% accuracy in each task and achieves only 0.2% accuracy reduction compared with Fish Mask. SSFT-Alg training is competitive with all other parameter-efficient transfer learning approaches.

2) LLaMA 2: We present the main results of our experiments in Table IV. For LLaMA 2 7B, we find that SSFT-Alg outperforms LoRA and IA^3 consistently across evaluation benchmarks and instruction tuning datasets, including Flan v2 and GPT4-Alpaca. SSFT-Alg achieves better accuracy than LoRA on 5 out of 6 tasks. We also find that 4-bit quantization

TABLE IV
ACCURACY OF PEFT AND QUANTIZED PEFT ALGORITHMS ON A RANGE OF EVALUATION TASKS FOR VARIOUS INSTRUCTION TUNING DATASETS.

Method	Flan v2 MMLU				Flan v2 TyDiQA				GPT4-Alpaca HumanEval			
	LLaMA 2 7B		LLaMA 2 13B		LLaMA 2 7B		LLaMA 2 13B		LLaMA 2 7B		LLaMA 2 13B	
	Original	Quant	Original	Quant	Original	Quant	Original	Quant	Original	Quant	Original	Quant
Original	45.8	44.8	55.3	54.7	50.9	50.2	60.3	59.6	13.5	11	17.8	15.5
FullFT	50.5	-	-	-	55.5	-	-	-	15.2	-	-	-
IA^3 [13]	46.7	-	55.1	-	51.6	-	60.1	-	14.7	-	18.5	-
LoRA [11]	49.3	47.7	55.8	55.0	55.3	54.3	61.4	60.7	15.7	13.3	19.8	18.2
SpIEL [3]	50.4	48.8	56.1	55.4	56.3	54.9	62.6	61.1	15.6	15.3	20.1	18.5
SSFT-Alg	50.2	48.8	56.3	55.2	56.2	54.8	62.5	61.2	15.6	14.3	20.5	18.6

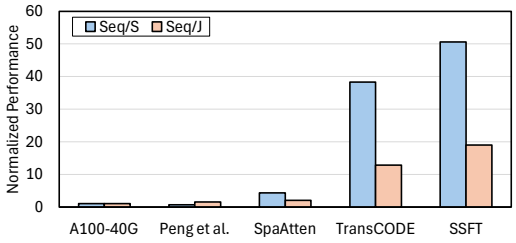


Fig. 6. Performance and energy efficiency comparisons. Seq/S represents sequences per second, and Seq/J represents sequences per joule. Implementation on A100-40GB is based on original $\text{BERT}_{\text{base}}$ [8].

TABLE V
LATENCY AND ENERGY EFFICIENCY OF RUNNING LARGER MODELS ON DIFFERENT PLATFORMS. TRANSCODE CANNOT SUPPORT LLAMA 2 7B DUE TO THE OUT-OF-MEMORY (OOM) ISSUE.

Model	Design	Seq/S	Seq/J
$\text{BERT}_{\text{LARGE}}$	A100	31.90	0.13
	TransCODE	357.90	2.06
	SSFT-Hw	476.61	3.13
LLaMA 2 7B	A100	2.21	0.01
	H100	6.40	0.02
	TransCODE	OOM	OOM
	SSFT-Hw	8.70	0.04

results in only a modest reduction in performance across PEFT methods, and their relative performance remains similar. These results demonstrate that SSFT-Alg is a competitive PEFT method even in extremely resource-constrained scenarios. Meanwhile, SSFT-Alg can also achieve competitive accuracy with the SOTA SFT, SpIEL [3], with less than a 0.2% accuracy drop for all but one task. On LLaMA 2 13B, SSFT-Alg archives less than 0.2% accuracy drop. On some of the tasks, such as MMLU, SSFT-Alg even achieves better accuracy than SpIEL. Overall, SSFT-Alg is competitive in all 6 combinations of scales and instruction tuning datasets on LLaMA 2 with less than a 1% accuracy drop. Hence, SSFT-Alg appears to be a strong method that provides competitive accuracy with a hardware-friendly algorithm.

C. Performance and Energy Efficiency of SSFT

Fig. 6 shows performance comparisons of SSFT with various platforms. These platforms include NVIDIA A100, which is widely used in LLM fine-tuning. We also add co-design methods implemented on ASIC-based accelerators [24], [27] and FPGAs [18]. The throughput and energy values are normalized with respect to the A100 GPU. Since some of the baselines do not support training, we report the performance values based on the inference performance scaling with the percentage of computation in the inference stage proposed in [24]. SSFT achieves $51.0\times$ higher throughput and $19.0\times$ lower energy consumption when compared to the off-the-shelf A100 GPU. SSFT also outperforms the state-of-the-art sparsity-aware Transformer training accelerator TransCODE $1.32\times$ on throughput and $1.48\times$ on energy consumption, respectively. Furthermore, to demonstrate the scalability of our methodology, we present the latency and energy consumption on $\text{BERT}_{\text{LARGE}}$ and LLaMA 2 7B in Table V. On $\text{BERT}_{\text{LARGE}}$, SSFT-Hw outperforms A100 and TransCODE $14.94\times$ and $1.33\times$, respectively, in terms of latency (Seq/S). And it outperforms TransCODE $1.52\times$ on energy efficiency (Seq/J). On LLaMA 2 7B, TransCODE fails to support fine-tuning due to memory limitations. Consequently, SSFT-Hw is $3.9\times$ and $1.4\times$ faster than A100 and H100, respectively, while saving $4.0\times$ and $2.0\times$ energy compared to the A100 and H100. These performance gains can be attributed to the support of structured gradient generation.

VI. CONCLUSION

This work introduces Structured Sparse Fine-Tuning (SSFT), a significant advancement in fine-tuning LLMs that bridges the gap between algorithmic efficiency and hardware capability. SSFT-Alg shows that integrating structured sparsity into the fine-tuning process is possible. Complemented by the SSFT-Hw accelerator which is designed to efficiently utilize sparse gradients with sparsity-aware modules and a novel dataflow, our approach not only enhances computational efficiency and reduces memory usage but also demonstrates superior performance and energy efficiency compared to existing methods. Validated on prominent models like BERT, LLaMA 2 7B and 13B, SSFT outperforms GPUs and the state-of-the-art sparsity-aware transformer accelerators by $51.0\times$ and $1.32\times$ in throughput and $19.0\times$ and $1.48\times$ in energy efficiency, respectively, with less than 1% accuracy reduction.

VII. ACKNOWLEDGMENTS

We thank the anonymous reviewers. This work is supported by the NUS Artificial Intelligence Institute (NAII-SF-2024-004) and the Singapore Ministry of Education (T2EP20222-0026). This work is also supported by IAF, A*STAR, SOITEC, NXP, and the National University of Singapore under FD-fAbrICS: Joint Lab for FD-SOI Always-on Intelligent & Connected Systems (I2001E0053).

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Ansell, E. Ponti, A. Korhonen, and I. Vulić, "Composable sparse fine-tuning for cross-lingual transfer," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022, pp. 1778–1796.

[3] A. Ansell, I. Vulić, H. Sterz, A. Korhonen, and E. M. Ponti, "Scaling sparse fine-tuning to large language models," *arXiv preprint arXiv:2401.16405*, 2024.

[4] E. Ben Zaken, Y. Goldberg, and S. Ravfogel, "BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1–9.

[5] M. Chen, J. Tworek, H. Jun *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[6] J. H. Clark, E. Choi, M. Collins, D. Garrette, T. Kwiatkowski, V. Nikolaev, and J. Palomaki, "TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 454–470, 2020.

[7] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] A. Guler and N. K. Jha, "Hybrid monolithic 3-d ic floorplanner," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1868–1880, 2018.

[10] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[11] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[12] V. Lialin, V. Deshpande, and A. Rumshisky, "Scaling down to scale up: A guide to parameter-efficient fine-tuning," *arXiv preprint arXiv:2303.15647*, 2023.

[13] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, 2022.

[14] S. Longpre, L. Hou, T. Vu, A. Webson, H. W. Chung, Y. Tay, D. Zhou, Q. V. Le, B. Zoph, J. Wei, and A. Roberts, "The flan collection: Designing data and methods for effective instruction tuning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 631–22 648.

[15] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "PEFT: State-of-the-art parameter-efficient fine-tuning methods," https://github.com/huggingface/peft, 2022.

[16] I. Nisa, A. Sukumaran-Rajam, S. E. Kurt, C. Hong, and P. Sadayappan, "Sampled dense matrix multiplication for high-performance machine learning," in *2018 IEEE 25th International Conference on High Performance Computing*. IEEE, 2018, pp. 32–41.

[17] B. Peng, C. Li, P. He, M. Galley, and J. Gao, "Instruction tuning with GPT-4," *arXiv preprint arXiv:2304.03277*, 2023.

[18] H. Peng, S. Huang, S. Chen, B. Li, T. Geng, A. Li, W. Jiang, W. Wen, J. Bi, H. Liu, and C. Ding, "A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1135–1140.

[19] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture*. IEEE, 2020, pp. 58–70.

[20] H. Shao, J. Lu, M. Wang, and Z. Wang, "An efficient training accelerator for transformers with hardware-algorithm co-optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[21] Y.-L. Sung, V. Nair, and C. A. Raffel, "Training neural networks with fixed sparse masks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 193–24 205, 2021.

[22] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[23] S. Tuli and N. K. Jha, "Acceltran: A sparsity-aware accelerator for dynamic inference with transformers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[24] ——, "Transcode: Co-design of transformers and accelerators for efficient training and inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[25] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv preprint arXiv:1908.08962*, 2019.

[26] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355.

[27] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 2021, pp. 97–110.

[28] M. Zhou, W. Xu, J. Kang, and T. Rosing, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *2022 IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 2022, pp. 1071–1085.

STREAMINGGS: Voxel-Based Streaming 3D Gaussian Splatting with Memory Optimization and Architectural Support

Chenqi Zhang[1*], Yu Feng[1*], Jieru Zhao[1†], Guangda Liu[1], Wenchao Ding[2], Chentao Wu[1], Minyi Guo[1]
[1] School of Computer Science, Shanghai Jiao Tong University, [2] Academy for Engineering & Technology, Fudan University
{zhangchenqi123, y-feng, zhao-jieru}@sjtu.edu.cn [*]Equal contribution. [†]Corresponding author.

***Abstract*—3D Gaussian Splatting (3DGS) has gained popularity for its efficiency and sparse Gaussian-based representation. However, 3DGS struggles to meet the real-time requirement of 90 frames per second (FPS) on resource-constrained mobile devices, achieving only 2 to 9 FPS. Existing accelerators focus on compute efficiency but overlook memory efficiency, leading to redundant DRAM traffic. We introduce STREAMINGGS, a fully streaming 3DGS algorithm-architecture co-design that achieves fine-grained pipelining and reduces DRAM traffic by transforming from a tile-centric rendering to a memory-centric rendering. Results show that our design achieves up to 45.7× speedup and 62.9× energy savings over mobile Ampere GPUs.**

***Index Terms*—3D gaussian splatting, rendering, accelerator**

I. INTRODUCTION

In virtual and augmented reality (VR/AR), photo-realistic and real-time rendering is essential for delivering immersive user experiences. Among various rendering techniques [1]–[6], 3D Gaussian Splatting (3DGS) [7]–[12] has quickly gained its popularity for its efficient rendering process and sparse Gaussian representation (also called Gaussian ellipsoids) of scene geometries.

Nevertheless, 3DGS still falls short of the real-time requirement in the VR/AR domain, namely 90 frames per second (FPS) [13]–[15], on mobile devices like VR headsets. Our experiments show that 3DGS merely achieves 2 to 9 FPS on the Nvidia Orin NX [16] [1]. Although several studies have proposed dedicated accelerators for 3DGS [18], their designs primarily optimize compute efficiency and often overlook off-chip memory traffic during rendering, making them hard to satisfy the real-time requirement in real-world scenarios.

Algorithmically, existing methods [7], [18] focus on a tile-centric rendering paradigm which requires excessive intermediate data communication between stages, leading to frequent off-chip traffic in Fig. 1a. Our characterization shows that, under a 90 FPS real-time requirement, the DRAM traffic of the current rendering paradigm would exceed the bandwidth limits of today's mobile devices on real-world scenes (Sec. II-B).

[1]The performance of the Ampere GPU (3.7 TFLOPS, released in February 2023) is comparable to that of the GPU (3.5 TFLOPS) in the latest Snapdragon XR2 [17], released in September 2023 for VR.

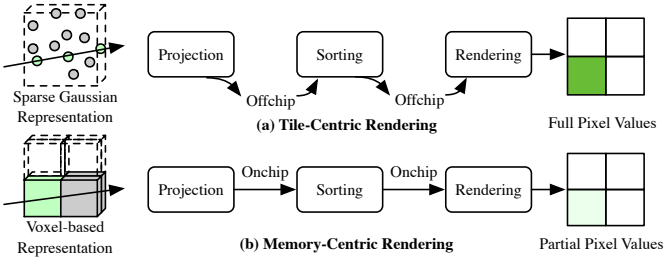


Fig. 1. Tile-centric rendering (a) vs. memory-centric rendering (b). The computation pattern is highlighted in green. Tile-centric rendering computes the complete pixel values tile-by-tile. Memory-centric rendering computes the partial pixel values voxel-by-voxel and regularizes the memory accesses.

The high data communication in the conventional tile-centric rendering comes from sequential processing ray-Gaussian intersections for each individual pixel, leading to redundant off-chip traffic and memory irregularity [18]–[21]. To address the bandwidth limitation, we argue to shift the rendering paradigm from a *tile-centric* to a *memory-centric* approach. Here, we introduce a fully streaming algorithm for 3DGS to address memory access irregularity (Sec. III). Our key idea is to regularize sparse Gaussian representations by partitioning the scene into voxels and performing on a voxel basis as shown in Fig. 1b. With all the points within a voxel stored contiguously in DRAM, we allow all points within a voxel to be streamed onto the on-chip buffer and only write the final pixel values off-chip, completely eliminating the intermediate off-chip traffic between stages.

While our memory-centric approach eliminates the intermediate off-chip traffic, our approach potentially introduces unnecessary off-chip traffic at the voxel streaming, as all points within a voxel are loaded on-chip as a whole. To reduce the unnecessary off-chip traffic, we propose hierarchical filtering to progressively exclude irrelevant Gaussians (Sec. III-B).

Specifically, we split each Gaussian feature into two halves and intentionally keep the first half lightweight. We then perform coarse yet lightweight computation to filter out irrelevant points based on the first half (i.e., the position and radius). Next, we perform precise filtering to further remove the irrelevant Gaussians by loading the second half. To further

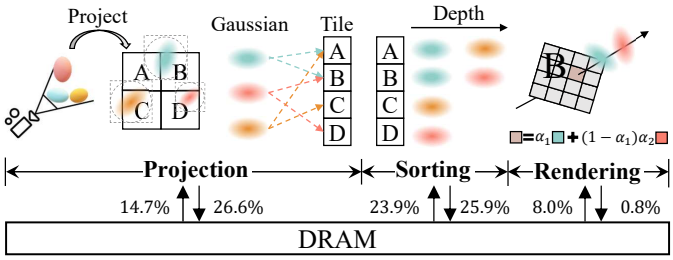


Fig. 2. The overall rendering pipeline of 3DGS. The percentage numbers represent the DRAM traffic proportion across stages.

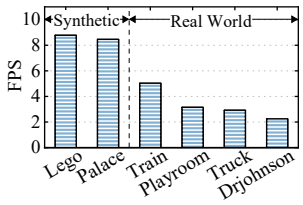


Fig. 3. FPS of different scenes on a mobile SoC, Orin NX. 3DGS cannot achieve real-time on both synthetic and real-world scenes.

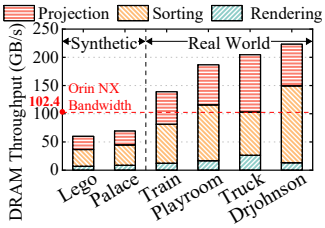


Fig. 4. The DRAM bandwidth requirement for 90 FPS under different scenes. The red dashed line highlights the bandwidth limit of Orin NX.

reduce the overhead of feature loading, we compress the second half using vector quantization. By storing a compact codebook on-chip, we just need to fetch their lightweight indices from DRAM. In this way, we drastically reduce the DRAM traffic at the voxel streaming.

To unleash the full potential of our fully streaming algorithm, we co-design an efficient accelerator (Sec. IV), including a voxel sorting unit to accelerate the determination of voxel rendering order and a dedicated hierarchical filtering unit to support Gaussian filtering at the voxel streaming. Coupled with our algorithmic optimizations, our design achieves a 45.7× speedup and 62.9× energy savings compared to a Nvidia mobile Ampere GPU [16]. Even compared against the state-of-the-art accelerator, GSCore [18], we can achieve a 2.1× speedup and 2.3× energy savings.

In summary, our contributions are as follows:

- We introduce a *memory-centric* rendering paradigm that can achieve fully streaming 3DGS rendering, completely avoiding intermediate off-chip traffic.
- We propose a hierarchical filtering scheme to further reduce the data traffic of the voxel streaming.
- With our architectural supports, our optimizations achieve an order of magnitude speedup and energy savings compared to a mobile Ampere GPU. We also achieve a 2.1× speedup and 2.3× energy savings against GSCore.

II. Background and Characterization

A. Background

Overall, 3DGS uses Gaussian ellipsoids to render a scene in a tile-by-tile fashion. Fig. 2 shows that this process can be classified into three stages:

- **Projection.** This stage projects each Gaussian ellipsoid onto the 2D screen and determines its intersected tiles as shown on the left side of Fig. 2. Each Gaussian might intersect with multiple tiles, as the colored dashed arrow lines show. Meanwhile, this stage calculates Gaussian properties, such as color and depth, for the later stages.
- **Sorting.** Each tile then sorts its intersected Gaussians based on Gaussian depths to ensure the correct rendering order from the closest to the farthest.
- **Rendering.** Every pixel within a tile traverses the same sorted Gaussian list and accumulates Gaussian colors to itself via alpha blending. For instance, we show tile B needs to blend two Gaussians, a blue one and a red one.

B. Characterization

Performance. We first show that 3DGS algorithms cannot achieve real-time performance on mobile devices. Fig. 3 characterizes the FPS of 3DGS [7] across various scenes on NVIDIA's Jetson Orin NX [16]. As the rendering scenes become more complex, shifting from synthetic to real-world scenes, performance drops from 8.5 to 4.9 FPS.

DRAM Traffic. We then show that DRAM traffic has a strong correlation with performance. Fig. 4 illustrates the DRAM throughput required to achieve the real-time requirement (90 FPS) for AR/VR applications [13], [14] across different rendering scenes. For real-world scenes, the DRAM bandwidth demand exceeds the bandwidth limit of a recently released mobile device, Nvidia Orin NX [16], as highlighted in the red dashed line. This means that the data communication requirement alone has made it impossible to achieve real-time.

We further dissect the DRAM traffic contributions across different scenes. As shown in Fig. 4, *projection* and *sorting* dominate the overall off-chip data communication, together accounting for 90% of the total DRAM traffic. The reasons that *projection* and *sorting* become the major contributors are:

- In projection, each Gaussian requires loading 59 parameters into the on-chip buffer and writing the processed features and intersection metadata back to DRAM, accounting for 41% of DRAM traffic.
- In sorting, repetitive read and write operations for progressive Gaussian sorting lead to repetitive DRAM access, taking up over 49% of DRAM traffic.

Overall, the intermediate data communication accounts for 85% of the total off-chip traffic in Fig. 2. This result motivates us to propose a fully-streaming algorithm that minimizes the off-chip traffic. Sec. III presents our algorithm design and Sec. IV explains the architectural support for our algorithm.

III. Algorithm Design

Sec. III-A first gives an algorithm overview and challenges that we need to address. Then, Sec. III-B and Sec. III-C explain the key components in our algorithm and the data layout that supports our algorithm, respectively.

A. Overview

The main downside of the original 3DGS pipeline is that it renders a frame tile-by-tile and each tile executes three stages sequentially. The reason is that the intermediate data of each

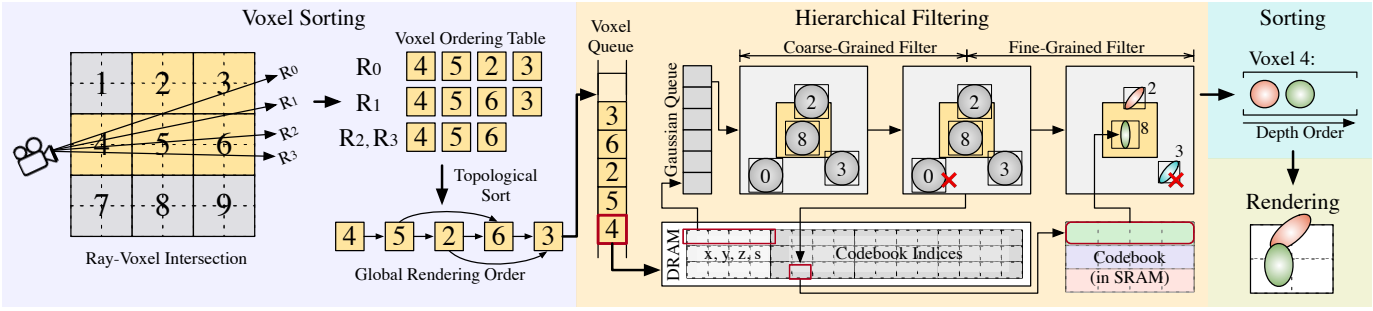


Fig. 5. The overview of our fully-streaming algorithm. Our algorithm initially partitions the entire 3D scene into small voxels. For each image tile, we first establish a rendering order of the intersected voxels via topological sort, then, process these voxels sequentially. During each voxel processing, we apply hierarchical filtering to further reduce the Gaussian loading within a voxel before the remaining sorting and rendering stages. The final pixel values are obtained by accumulating all the partial results across voxels.

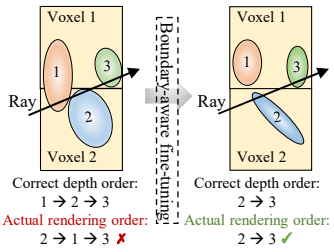


Fig. 6. An example of incorrect rendering order among Gaussians and the effect of boundary-aware fine-tuning.

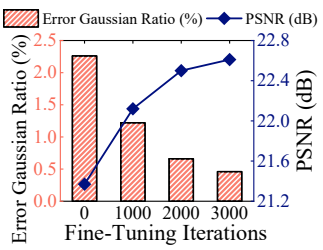


Fig. 7. Changes in the ratio of Gaussians with incorrect depth order and PSNR during the boundary-aware fine-tuning process.

stage is too large to be accommodated on-chip, inevitably, introducing frequent off-chip traffic during the execution.

Idea. To eliminate frequent off-chip communications at intermediate stages, our idea is to partition an entire 3D scene into small voxels and incrementally render the entire scene voxel-by-voxel. By processing one voxel at a time, our algorithm guarantees the intermediate data between stages can be completely held on-chip, thus, no off-chip traffic occurs.

Fig. 5 gives an overview of our algorithm. Our algorithm renders a group of pixels together, e.g., R_0 to R_3 are in the same group. For each group, we first identify which voxels intersect with these pixel rays and store the results in the voxel rendering table. Once each pixel's intersected voxels are determined, we establish their global rendering order. Then, each voxel goes through hierarchical filtering, sorting, and rendering in order, and obtain the partial pixel values of this tile. The final pixel values are obtained by accumulating all the partial results across voxels. By rendering one voxel at a time, each stage requires minimal data communication, thus, no off-chip traffic is required throughout the execution.

Challenges. However, there are three challenges when applying our algorithm:

- *Inter-Voxel Order.* Pixels within a single group often intersect with multiple voxels, as shown in Fig. 5. The rendering order for each pixel should be from the closest voxel to the farthest. The first challenge is to ensure the correct global order for all intersected voxels.
- *Cross-Voxel Gaussian Order.* Recall, each pixel is required to render Gaussians from the closest to the farthest in 3DGS, but the correct voxel-level ordering does not always guarantee the rendering order correctness at Gaussian-level granularity. For instance, on the left side of Fig. 6, Gaussian 2 in voxel 2 renders before Gaussian 1 in voxel 1, whereas the correct order should be reversed.
- *Redundant Gaussians in Voxels.* Lastly, each voxel includes Gaussians within close proximity; however, loading an entire voxel might unintentionally bring unrelated Gaussians on-chip. Leaving these redundant Gaussian without care would introduce excessive DRAM traffic.

B. Fully Streaming Algorithm

In this section, we explain how we address these challenges.

Inter-Voxel Order. To establish the correct voxel order across pixels in each tile, we first collect the rendering order of individual pixels in a voxel ordering table in Fig. 5. Based on individual rendering orders, we construct a direct acyclic graph (DAG), where each voxel is represented as a node, and edges represent the rendering dependencies. We then apply a topological sort [22] on the DAG to guarantee that no cycles in this graph and all the render dependencies are satisfied. The sorted order is then used in the remaining process.

Cross-Voxel Gaussian Order. To ensure correct rendering order among Gaussians, we observe that the incorrect order occurs only when a Gaussian spans across multiple voxels. In Fig. 6, Gaussians should be rendered from Gaussian 1 (the closest) to Gaussian 3 (the farthest). However, with voxel-by-voxel rendering, Gaussian 2 is incorrectly rendered before Gaussian 1. Fig. 7 shows that incorrect ordering can lead to a degradation in rendering quality.

To alleviate this issue, we introduce a *boundary-aware fine-tuning* to adjust cross-voxel Gaussians. Specifically, we keep each Gaussian position fixed to retain the scene geometry while fine-tuning its remaining trainable parameters, such as scale, orientation, etc. The fine-tuning reduces the size and modifies the orientation of Gaussians, minimizing their likelihood of spanning across other voxels.

In our fine-tuning, we propose a new loss function to punish the cross-boundary Gaussians,

$$\mathcal{L} = \mathcal{L}_{origin} + \beta \mathcal{L}_{CBP}, \tag{1}$$

where $\mathcal{L}_{origin}$ is the original loss in 3DGS and $\mathcal{L}_{CBP}$ is the cross-boundary penalty loss which penalizes Gaussians that span across voxel boundaries. β is a hyper-parameter to balance the influence of $\mathcal{L}_{CBP}$.

$\mathcal{L}_{CBP}$ minimizes the scale of the Gaussians that cross voxel boundaries and is expressed as,

$$\mathcal{L}_{CBP} = \frac{1}{N}\sum_{i=0}^{N-1} S_i T_i, \tag{2}$$

$$\text{where } T_i = \begin{cases} 1, & \text{if depth}_i < \max_{j\in[0,i)}\{\text{depth}_j\}, \\ 0, & \text{otherwise}, \end{cases}$$

where N is the total number of Gaussians, S_i is the maximum scale of Gaussian i. T_i is an indicator that checks if current Gaussian i has the largest depth compared to all previously rendered Gaussians. If this condition is not satisfied, we penalize it by setting T_i to be 1.

Fig. 7 illustrates the effectness of our boundary-aware fine-tuning. After 3,000 iterations, the percentage of cross-boundary Gaussians decreases from 2.3% to 0.4%, while the rendering quality regains from 21.37 dB to 22.61 dB.

Redundant Gaussians in Voxels. Recall that during projection, each Gaussian requires 59 parameters. To avoid irrelevant Gaussians, we propose a two-phase *hierarchical filtering* to progressively fetch Gaussian parameters in Fig. 5.

The first phase serves as a lightweight but coarse-grained filter that approximates whether Gaussians are intersected with the image tile. In this phase, each Gaussian only requests 4 parameters (3D coordinates and maximum scale) from off-chip memory to compute the projected center and the maximum projected radius with respect to the image tile. If a Gaussian does not pass the coarse-grained filter, we skip its remaining computations and avoid loading its remaining 55 parameters. For example, Gaussian 0 in Fig. 5 does not intersect with the image tile in yellow, we can safely remove it from the subsequent computation.

In the second fine-grained filtering, we identify the actual Gaussians that are intersected with the image tile by computing precise projection. For example, after getting shapes and orientations, we can know that Gaussian 3 in Fig. 5 does not intersect with the image tile actually and hence can skip its subsequent sorting and rendering. To further reduce the off-chip traffic, we propose a novel data compression scheme to encode the remaining parameters into a compact codebook. This way, we only need to load codebook indices from DRAM instead of the original parameters. We explain our data compression scheme in Sec. III-C. Combining coarse- and fine-grained filtering, our hierarchical filtering reduces the number of Gaussians processed in each voxel by 76.3%.

C. Customized Data Layout

To reduce data communication in hierarchical filtering, we propose a customized data layout in Fig. 8. Overall, we divide all Gaussian features into two halves, which are stored separately in DRAM and used at different filtering phases. The first half, containing 4 parameters (the 3D coordinates and maximum scale), is used for the coarse-grained filter. The second half, used in the fine-grained filter, includes the remaining parameters, such as rotation matrices.

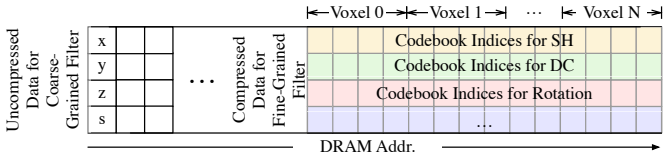


Fig. 8. The DRAM Data layout. We separate the Gaussian features into two halves for two filtering phases and compress the data for the second phase using vector quantization to reduce the off-chip communication.

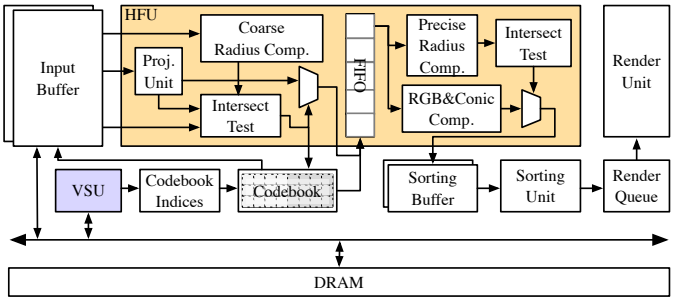


Fig. 9. Overview of StreamingGS accelerator, consisting of four components: a voxel sorting unit (VSU), a hierarchical filtering unit (HFU), a sorting unit, and a rendering unit. Our contributions are highlighted in colors: HFU is in yellow and VSU is in purple. The detailed design of VSU is in Fig. 10.

To alleviate DRAM traffic, we apply the vector quantization to compress the second half, so that we store only codebook indices off-chip while keeping the codebook itself on-chip for decoding. In addition, we encode different parameters into separate codebooks to preserve quantization precision and adopt the quantization-aware fine-tuning [9] to update the Gaussian parameters, allowing the indices to capture feature variations without loss of detail. Our result shows that, during the voxel streaming, only the codebook indices are fetched from DRAM and reduce 92.3% DRAM traffic. Note that, we only compress the second half, because we empirically find that compressing the first half leads to accuracy loss.

IV. Architectural Support

A. Overview

To support our fully-streaming algorithm, we propose our co-designed accelerator in Fig. 9. Our architecture consists of four components: a voxel sorting unit (VSU), a hierarchical filtering unit (HFU), a sorting unit, and a rendering unit.

Given a set of ray directions, VSU first identifies and sorts the intersected voxels. Then, for each intersected voxel, HFU applies a hierarchical filter to identify intersected Gaussians. Both designs will be explained in Sec. IV-B and Sec. IV-C, respectively. Once HFU obtains the filtered Gaussians, sorting and rendering units execute the sorting and rendering stage from their individual buffer, respectively. Both sorting and rendering designs are adopted from GSCore [18]. However, we simplify the sorting unit by just adopting the bitonic sorting unit from GSCore, as our voxel-based rendering only requires establishing the rendering order for Gaussians within a voxel.

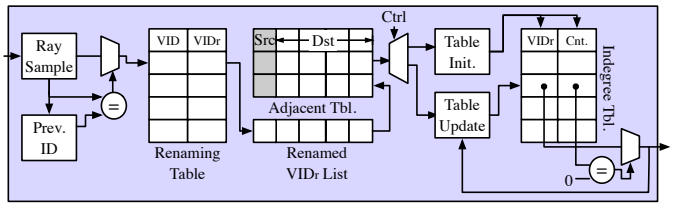


Fig. 10. The design of voxel sorting unit (VSU). It first identifies the intersected voxels for every pixel within a tile and then establishes the global rendering order of these voxels for sorting and rendering.

B. Voxel Sorting Unit (VSU)

The architecture of VSU is illustrated in Fig. 10. For pixels within a group, VSU samples along each pixel ray to identify the intersected voxels and performs a topological sort to establish the global rendering order of these voxels. Since we partition the entire scene into voxels offline, the voxel ID (VID) can be directly computed by the coordinate of a ray sample. To reduce the VID range, we further filter out those empty voxels with no Gaussians by augmenting a renaming table in VSU to reassign each VID with a new VID_r.

Each time we obtain a renamed VID_r list for one pixel, we use this list to construct an adjacent table. This adjacent table functions like a cache to help the dependency lookup among voxels. Each table entry has a tag and a value, representing a source voxel and all its corresponding destination voxels, respectively. The adjacent table is constructed once VSU finishes iterating all pixel rays in a group.

VSU then uses the adjacent table to initialize an in-degree table. Each entry in the in-degree table is directly indexed by a VID_r value and stores its in-degree count, which records the number of source voxels this VID_r has. If an in-degree of a VID_r becomes zero, this VID_r has no source dependency and can be written to the later stage. Each time VSU outputs a VID_r, VSU also updates the in-degree table accordingly. The topological sort finishes when the in-degree table is empty.

C. Hierarchical Filtering Unit (HFU)

Once VSU generates the rendering order of intersected voxels, HFU starts to load one voxel at a time and process the Gaussians within this voxel. Overall, HFU, highlighted in yellow in Fig. 9, consists of a frontend and a backend, responsible for coarse-grained and fine-grained filtering, respectively.

Coarse-Grained Filter. HFU only reads each Gaussian coordinate and maximum scale in the coarse-grained filter, computing its projected position and coarse radius. These results are sent to the intersection test unit to determine if the Gaussian intersects with the current image tile. If the condition is met, HFU signals the codebook to decode the remaining parameters from the codebook buffer, as shown in Fig. 9. The decoded parameters, along with the uncompressed parameters, are then fed into a FIFO for fine-grained filtering.

Fine-Grained Filter. This step computes the exact radius computation and performs the intersection test. Meanwhile, the fine-grained filter unit also computes the RGB values and conic matrix from decoded parameters, such as scales and rotation. Based on the intersection result, HFU conditionally outputs a Gaussian RGB and conic results to the sorting buffer if this Gaussian passes the intersection test.

In our design, the coarse-grained filter largely reduces the computation (from 427 MACs to 55), compared to fine-grained filters. In addition, the coarse-grained filter avoids decoding and reading unnecessary Gaussians to FIFO in HFU.

TABLE I
CONFIGURATION AND AREA

Unit	Configuration	Area [mm^2]
Voxel Sorting Unit	1 Unit	0.06
Hierarchical Filtering Unit	4 Units	0.79
Sorting Unit	2 Units	0.04
Rendering Unit	64 Units	2.53
SRAM (Input Buffer, Codebook, others)	355KB	1.95
Total		5.37

V. EVALUATION

A. Experimental Setup

Hardware Setup. Our accelerator consists of 1 VSU, 4 HFUs, 2 sorting units, and $4 \times 4 \times 4$ rendering units at 1GHz clock frequency. Each HFU has 4 coarse-grained filter units (CFUs) and 1 fine-grained filter unit (FFU). The sorting unit and the rendering unit are identical to the bitonic sorting unit and the volume rendering unit in GSCore, respectively [18]. The overall on-chip buffer consists of a double-buffered input buffer with a size of 16 KB, a 250 KB codebook buffer, and the remaining 89 KB to store intermediate data between stages.

Our architectural design is developed using an EDA process and synthesized with Synopsys and Cadence tools on TSMC's 32 nm FinFET technology. SRAM components are estimated by CACTI 7.0 [23] at 32 nm technology. The DRAM model in our simulations is based on Micron's 16 Gb LPDDR3 with 4 channels according to its datasheet [24], with energy consumption data sourced from Micron System Power Calculators [25].

Software Setup. We configure a codebook with 4096 entries for scale, rotation, and DC, and a codebook with 512 entries for SH coefficients in Sec. III-C. The voxel size is set to 2 for real world scenes and 0.4 for synthetic scenes. The hyper-parameter, β, in Sec. III-B is set to 0.05.

Area. STREAMINGGS design has a total of 5.37 mm^2 in Tbl. I. The major area contributors are 4 HFUs (0.79 mm^2), 64 rendering units (2.53 mm^2), and on-chip buffers (1.95 mm^2). Overall, our design has a similar area compared to GSCore (5.53 mm^2) which is scaled to 32 nm by DeepScaleTool [26].

Workloads. We evaluate both synthetic and real-world datasets: Synthetic-NSVF [27], Synthetic-NeRF [3], Tanks&Temples [28] and Deep Blending [29]. Each scene is initially trained with a default 3×10^4 iterations, followed by 3×10^3 iterations of boundary-aware fine-tuning (Sec. III-B) and 5×10^3 iterations of quantization-aware fine-tuning (Sec. III-C). We evaluated our method on three widely-adopted 3DGS algorithms, including original 3DGS [7], Mini-Splatting [8], and LightGaussian [10].

Baselines. We compare against two hardware baselines: Nvidia GPU Orin NX [16] and a dedicated accelerator, GSCore [18]. GPU performance and power consumption are

TABLE II
QUANTITATIVE COMPARISON OF RENDERING QUALITY (PSNR ↑) ACROSS DIFFERENT DATASETS AND ALGORITHMS.

Algorithm \ Dataset	Train	Truck	Playroom	Drjohnson	Lego	Palace
3DGS	22.54	26.65	30.18	29.21	36.11	38.56
Ours	22.52	26.61	30.27	29.07	36.02	38.52
Mini-Splatting	21.49	25.19	30.32	29.23	36.20	39.00
Ours	21.44	25.11	30.37	29.34	36.18	38.98
LightGaussian	22.29	26.02	28.58	25.87	35.18	37.76
Ours	22.32	25.89	28.47	25.79	35.15	37.68

obtained using built-in hardware measurements. We implement GSCore based on its published specifications and validate our results against reported data.

Variants. We evaluate three variants:

- w/o VQ+CGF performs our fully-streaming algorithm without vector quantization and coarse-grained filtering.
- w/o CGF performs our fully-streaming algorithm without coarse-grained filtering.
- STREAMINGGS is our full-fledged version including all optimization techniques.

B. Rendering Quality

Tbl. II shows a quantitative comparison of rendering quality between the original pipeline and our fully streaming pipeline across three 3DGS algorithms. Overall, our fully streaming pipeline shows no accuracy degradation in visual quality. On average, our pipeline drops the rendering quality by 0.04 dB. Our results sometimes achieve higher PSNR than the original approach. Our rendering results are also presented here: link.

C. Performance and Energy Efficiency

Fig. 11 illustrates the performance and energy savings of our design against three hardware baselines. All values are normalized against the corresponding Orin NX. Both speedup and energy numbers are averaged across four datasets.

On average, STREAMINGGS achieves 45.7× speedup against Orin NX. In comparison, GSCore can barely achieve 21.6× speedup. Our result also shows that coarse-grained filtering affects the performance of STREAMINGGS significantly. Without coarse-grained filtering, the speedup drops from 45.7× to 22.2×, while VQ has a minimal impact on the performance. This is because VQ only affects the voxel streaming which is overlapped by the subsequent stages.

On energy saving, STREAMINGGS achieves 62.9× energy reduction over Orin NX and 2.3× energy reduction over GSCore. The primary energy savings are from DRAM traffic reduction and efficient early-stage irrelevant Gaussian filtering. In addition, coarse-grained filtering and VQ contribute to 35.6× and 5.8× energy savings, respectively.

D. Sensitivity Study

Fig. 12 shows the sensitivity of rendering quality and energy savings to voxel size on one real-world scene, *train*. All variants are retrained according to our training procedure in Sec. V-A. Generally, smaller voxel sizes result in more cross-boundary Gaussians, leading to incorrect depth ordering. Our result shows that the rendering quality drops from 22.3 dB to 21.5 dB when the voxel size decreases from 2 to 0.5. However, further increasing voxel size shows minimal impacts on rendering quality. Meanwhile, larger voxels might potentially include more irrelevant Gaussians in every voxel, increasing the filtering workload and lowering energy efficiency. We empirically find a voxel size of 2 has a good balance between performance and rendering quality.

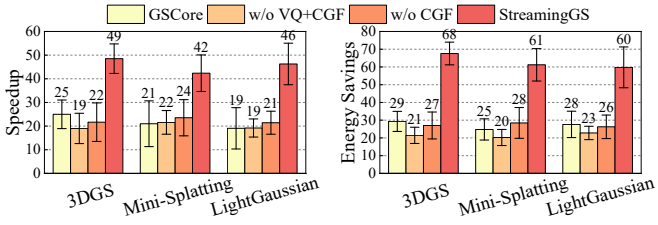


Fig. 11. End-to-end speedup and energy savings of our variants over the GPU baseline. The numbers are averaged over four datasets and normalized to the GPU baseline.

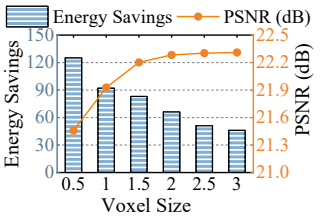


Fig. 12. The sensitivity of energy efficiency and rendering quality on the original 3DGS to voxel size. We only evaluate on *train* scene.

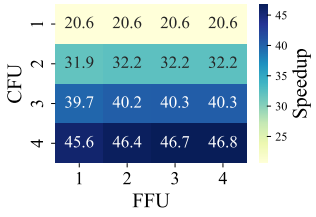


Fig. 13. The sensitivity of speedup on the original 3DGS to the numbers of CFUs and FFUs. Only *train* scene is evaluated.

Fig. 13 illustrates the sensitivity of performance to the numbers of coarse-grained filter units (CFUs) and fine-grained filter units (FFUs) in HFU (Sec. IV-C). We choose the same *train* scene and normalize all performance numbers to the GPU baseline. The results show that adding FFUs beyond the number of CFUs does not yield any speedup. In contrast, increasing the number of CFUs consistently boosts speedup, underscoring their importance in accelerating the filtering process. However, an excessive number of CFUs would lead to increased area overhead. Thus, we choose 4 CFUs with 1 FFU as our default configuration.

VI. CONCLUSION

This paper introduces STREAMINGGS that rethinks the 3DGS pipeline and proposes a new computing paradigm to process Gaussian points. Our fully streaming pipeline eliminates intermediate off-chip memory traffic and regularizes the off-chip accesses, all with a minimal but principle architecture augmentation. Overall, our algorithm-architecture co-design demonstrates 45.7× and 2.1× speedup compared to mobile GPUs and the state-of-the-art 3DGS accelerator, respectively.

VII. ACKNOWLEDGMENT

This work is sponsored by the National Natural Science Foundation of China (62472273, 62232015, 62402312).

REFERENCES

[1] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. MIT Press, 2023.

[2] Y. Deng, Y. Ni, Z. Li, S. Mu, and W. Zhang, "Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 1–41, 2017.

[3] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[4] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5855–5864, 2021.

[5] T. Wu, Y.-J. Yuan, L.-X. Zhang, J. Yang, Y.-P. Cao, L.-Q. Yan, and L. Gao, "Recent advances in 3d gaussian splatting," *arXiv preprint arXiv:2403.11134*, 2024.

[6] G. Chen and W. Wang, "A survey on 3d gaussian splatting," *arXiv preprint arXiv:2401.03890*, 2024.

[7] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023.

[8] G. Fang and B. Wang, "Mini-splatting: Representing scenes with a constrained number of gaussians," *arXiv preprint arXiv:2403.14166*, 2024.

[9] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3d gaussian representation for radiance field," *arXiv preprint arXiv:2311.13681*, 2023.

[10] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," *arXiv preprint arXiv:2311.17245*, 2023.

[11] W. Lin, Y. Feng, and Y. Zhu, "Metasapiens: Real-time neural rendering with efficiency-aware pruning and accelerated foveated rendering," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 669–682, 2025.

[12] X. Huang, H. Zhu, Z. Liu, W. Lin, X. Liu, Z. He, J. Leng, M. Guo, and Y. Feng, "Seele: A unified acceleration framework for real-time gaussian splatting," *arXiv preprint arXiv:2503.05168*, 2025.

[13] "Meta Quest Pro specs."

[14] "Apple Vision Pro screen refresh rate is up to 100Hz."

[15] J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, and H.-N. Liang, "Effect of frame rate on user experience, performance, and simulator sickness in virtual reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 5, pp. 2478–2488, 2023.

[16] "Nvidia jetson orin nx."

[17] "Qualcomm QCS8550/QCM8550 Processors."

[18] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 497–511, 2024.

[19] Y. Feng, W. Lin, Z. Liu, J. Leng, M. Guo, H. Zhao, X. Hou, J. Zhao, and Y. Zhu, "Potamoi: Accelerating neural rendering via a unified streaming architecture," *ACM Transactions on Architecture and Code Optimization*, 2024.

[20] H. He, G. Li, F. Liu, L. Jiang, X. Liang, and Z. Song, "Gsarch: Breaking memory barriers in 3d guassian splatting training via architectural support," in *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2025.

[21] Y. Feng, Z. Liu, J. Leng, M. Guo, and Y. Zhu, "Cicero: Addressing algorithmic and architectural bottlenecks in neural rendering by radiance warping and memory optimizations," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1293–1308, IEEE, 2024.

[22] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.

[23] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[24] "Micron 178-Ball, Single-Channel Mobile LPDDR3 SDRAM Features."

[25] "Micron System Power Calculators."

[26] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.

[27] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15651–15663, 2020.

[28] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.

[29] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (ToG)*, vol. 37, no. 6, pp. 1–15, 2018.

MILLION: MasterIng Long-Context LLM Inference Via Outlier-Immunized KV Product QuaNtization

Zongwu Wang[1,2,†], Peng Xu[1,2,†], Fangxin Liu[1,2,*], Yiwei Hu[1], Qingxiao Sun[4],
Gezi Li[3], Cheng Li[3], Xuan Wang[3], Li Jiang[1,2,*], and Haibing Guan[1]
1. Shanghai Jiao Tong University, 2. Shanghai Qi Zhi Institute,
3. Huawei Technologies Co., Ltd, 4.China University of Petroleum-Beijing
*Corresponding Author {wangzongwu, liufangxin, ljiang_cs}@sjtu.edu.cn

***Abstract*—Large language models (LLMs) are increasingly utilized for complex tasks requiring longer context lengths, with some models supporting up to 128K or 1M tokens. This trend, however, presents significant challenges in inference speed and memory management. The primary bottleneck in long-context LLM inference is the quadratic computational complexity of attention mechanisms, causing substantial slowdowns as sequence length increases. KV cache mechanism alleviates this issue by storing pre-computed data, but introduces memory requirements that scale linearly with context length, hindering efficient LLM deployment. Quantization emerges as a promising approach to address the widening gap between LLM size and memory capacity. However, traditional quantization schemes often yield suboptimal compression results for KV caches due to two key factors: i) On-the-fly quantization and de-quantization, causing significant performance overhead; ii) Prevalence of outliers in KV values, challenging low-bitwidth uniform quantization. To this end, we propose MILLION, a novel quantization framework achieving low-bitwidth KV cache through product quantization. First, we conduct a thorough analysis of KV cache distribution, revealing the limitations of existing quantization schemes. Second, we introduce a non-uniform quantization algorithm based on product quantization, which efficiently compresses data while preserving accuracy. Third, we develop a high-performance GPU inference framework with efficient attention kernel and pipeline design for MILLION that leverages sparse computation and asynchronous quantization, significantly enhancing inference speed. Comprehensive evaluation results demonstrate that MILLION can achieve 4 bits quantization with trivial perplexity and accuracy loss, and achieve 2.09x end-to-end performance gains at 32K context length. Code is released at https://github.com/ZongwuWang/MILLION.**

I. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across various tasks, with recent efforts focusing on expanding their context lengths to further enhance performance. Longer context lengths enable new applications such as long document summarization, retrieval-based question answering for extensive documents, extended multi-turn interactions, and code analysiscite [1]. This demand has driven significant advancements in long-context models in both industry [2], [3] and academiacite [1]. While increased contextual capability enhances model intelligence, it also introduces substantial computational demands, scaling quadratically ($O(n^2)$) with context length. To address this challenge, modern LLM frameworks have adopted the key-value (KV) cache, which reduces the computational complexity to $O(n)$ at the cost of increased memory usage. However, the practice of batch processing requests, while improving tensor processing efficiency, further exacerbates storage overhead as KV caches cannot be shared between multiple requests [4], [5].

KV cache storage scales linearly with context length, often surpassing device memory limits. For example, in the 540B PaLM model, a batch size of 512 and context length of 2048 can result in a 3TB KV cache—three times the model's parameter size. Additionally, each token generation in the decoding phase requires full KV cache access, creating a major memory bottleneck in long-context scenarios. Recent research has focused on addressing these memory-related challenges in LLMs through three primary approaches: **1) Optimized attention mechanisms:** Techniques such as MQA [6], GQA [7], MLA [8], YOCO [9], and CHAI [10] have been proposed to reduce computational and storage overheads by sharing KVs among attention heads. However, these optimizations often come at the cost of reduced accuracy, with only GQA currently seeing widespread adoption in state-of-the-art LLMs. **2) Sparse attention mechanisms:** These approaches aim to reduce the complexity of attention computations. For example, given the importance of local context, sliding window attention maintains a fixed-size sliding window of recent token states to limit the number of tokens involved in computations [11]. Streaming-LLM further improves on this by introducing the "attention sink" tokens [12], demonstrating that retaining initial token KVs can recover much of the lost performance. More advanced KV sparsity techniques [13]–[18] attempt to identify and remove less important token KVs based on historical attention patterns. However, recent studies [19], [20] suggest that these approaches may suffer from accuracy degradation, as past attention distributions may not reliably predict future attention needs in long-context generation. **3) Key-value quantization:** Quantization, successful in model weight compression [21]–[24], has emerged as a promising technique for KV cache optimization. However, applying weight quantization methods directly to KV caches presents unique challenges: First, the presence of outliers in the KV states of LLMs expands the quantization range, making low-bit quantization

† These authors contributed equally. This work was supported by the National Key Research and Development Program of China (2024YFE0204300), National Natural Science Foundation of China (Grant No.62402311), and Natural Science Foundation of Shanghai (Grant No.24ZR1433700).

difficult. Second, since KVs are dynamically generated during inference, quantization or de-quantization delays can negatively impact real-time performance. According to [25], 4 bit integer dequantization contributes 20% to 90% of the inference latency. Additionally, linear attention mechanisms like Manba [26], RWKV [27] and InfiniTransformer [28] have been developed to further reduce the complexity of attention computations. However, these approaches also tend to trade off some degree of model accuracy for efficiency gains.

Recent works have attempted to address these challenges: QServer [25] proposed SmoothAttention, which implements 4-bit KV cache quantization with the SmoothAttention scale Λ into previous layers' weights to reduce de-quantization overhead, however the accuracy is limited by the hard constraint introduced by the rotary positional embedding. KIVI employs group-wise quantization, adapting to outlier distribution through channel-wise and token-wise quantization of keys and values. KVQuant further refines this approach with outlier isolation and non-uniform quantization, improving low-bitwidth accuracy at the cost of inference performance. Unfortunately, these techniques often degrade inference speed due to the overhead introduced by outlier isolation and non-uniform quantization. The ideal solution would balance these factors without significantly compromising any single aspect.

This work introduces MILLION, a novel quantization scheme for efficient dynamic KV compression in LLMs. MILLION utilizes non-uniform quantization and eliminates de-quantization overhead, integrating seamlessly with a high-performance system optimized for GPU-based inference. First, we thoroughly analyze the distribution patterns of KVs in LLMs, uncovering the limitations of existing quantization methods. Based on these distribution characteristics, we propose a non-uniform quantization scheme using Product Quantization (PQ). By clustering vectors, PQ distributes quantization power unevenly across channels, which helps handle outliers in the KV states. PQ also naturally supports mixed-precision quantization, offering flexibility in balancing performance and accuracy. Finally, we design an efficient GPU inference framework for MILLION, which accelerates inference by leveraging sparse computation and asynchronous quantization, boosting processing speed while reducing memory overhead.

In summary, we make the following contributions:

- **KV distribution analysis**: We perform an in-depth analysis of the distribution characteristics of key-value pairs in LLMs, identifying the limitations of existing quantization methods and highlighting the unique challenges posed by outliers and dynamic KV generation.
- **PQ-based non-uniform quantization**: Building on the insights from the KV distribution analysis, we propose a novel non-uniform quantization scheme based on PQ. This approach leverages clustering to distribute quantization power unevenly across channels, enabling efficient compression while maintaining high accuracy.
- **Efficient GPU inference system**: We design and implement a high-performance GPU-based inference framework optimized for MILLION. This system accelerates inference through sparse computation and asynchronous quantization, effectively eliminating de-quantization overhead while ensuring fast, scalable, and memory-efficient processing of long-context inputs.

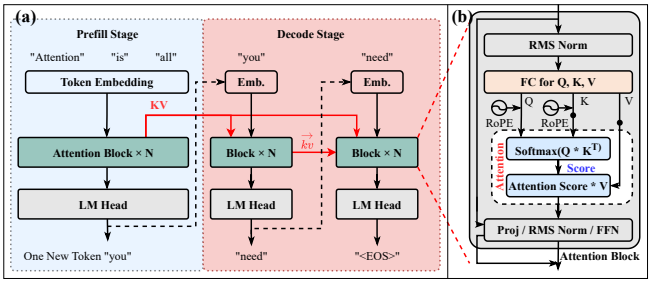


Fig. 1. Overview of transformer model structure. (a) Transformer model consists of two stage, the prefill stage processes the prompt prefilling of all tokens in batches, and the decode stage generates tokens one by one; (2) Details of the attention block.

II. PRELIMINARY

A. Attention-based LLM model.

The Transformer model architecture consists of two main stages: the Prefill stage and the Decode stage, as depicted in Fig. 1(a). In the Prefill stage, the model processes the entire input sequence in batches, and generate the key and value vectors for each prompt tokens simultaneously. The Decode stage operates auto-regressively, generating the output sequence token by token. The auto-regressively interaction is handled by the attention blocks, which is the most important modules in the transformer models. The detailed algorithm of the attention block is depicted in Fig. 1(b). The unique attention mechanism, highlighted with a white background, enables token interactions within the context. This process is formalized as follows:

$$A_n = \text{Softmax}(\frac{Q \times K^T}{\sqrt{d_k}}) \times V \tag{1}$$

Where $Q \in \mathbb{R}^{n_q \times d_k}$, K and $V \in \mathbb{R}^{n_k \times d_k}$ represent the query, key, and value vectors corresponding to each token, d_k denotes the head dimension, n_q and n_k denote the token number of query and key/value for the attention computation, respectively. Due to the causality of the language model, the query vector of each token needs to be feature fusion with all previous key/value vectors. Then the attention scores are utilized to calculate the weighted sum of the value vectors for each token, resulting in the current context state vector. This is followed by feature extraction using additional FFN layers.

B. Integer Quantization

Existing integer quantization maps high-precision numbers to discrete levels, the process can be formulated as:

$$\begin{cases} \boldsymbol{Q_X} = \lceil \frac{\boldsymbol{X}}{s} + z \rfloor \\ s = \frac{\boldsymbol{X}_{max} - \boldsymbol{X}_{min}}{q_{max} - q_{min}} \\ z = \lceil q_{min} - \frac{\boldsymbol{X}_{min}}{s} \rfloor \end{cases} \tag{2}$$

where $\boldsymbol{X}$ is the floating point tensor, $\boldsymbol{Q_X}$ is its n-bit quantized counterpart, s is the scaling factor and z is the zero point. Thus, the dequantized tensor can be represented as,

$$\hat{\boldsymbol{X}} = Q(\boldsymbol{X}) = (\boldsymbol{Q_X} - z) \cdot s \tag{3}$$

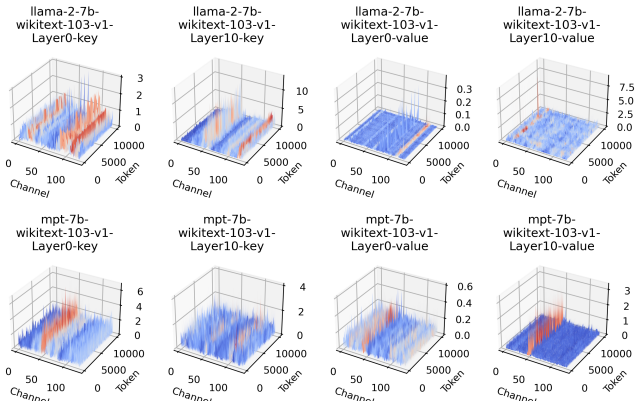


Fig. 2. Magnitude distribution of key and value cache for Llama-2-13B and Falcon-7B.

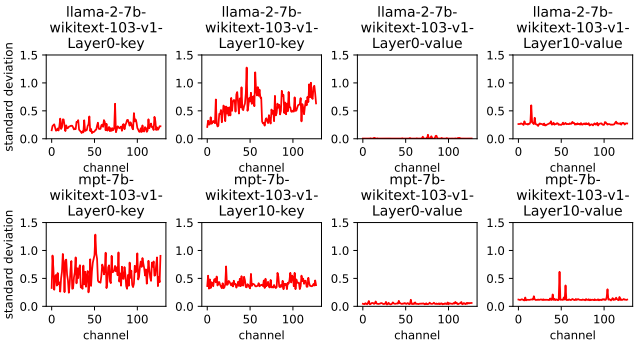


Fig. 3. channel-wise standard deviation distribution of key and value cache for Llama-2-13B and Falcon-7B.

This is known as asymmetric quantization, where $\boldsymbol{X}max = max(\boldsymbol{X})$, $\boldsymbol{X}min = min(\boldsymbol{X})$, and $q_{max} - q_{min} = 2^n - 1$ for n-bit integer quantization. Fig. 2 can be further simplified to symmetric quantization, where the zero point z is set to 0, and the range is centered around zero with $\boldsymbol{X}max = -\boldsymbol{X}min = max|\boldsymbol{X}|$. In this case, $q_{max} - q_{min} = 2^n - 2$, as one bit is reserved for the sign.

The de-quantization process of integer quantization needs to be implemented using general-purpose CUDA cores, which are less efficient for this type of operation. This implementation introduces a significant performance bottleneck, resulting in 20% to 90% overhead [25].

C. Product Quantization

Product Quantization (PQ) is an innovative algorithm that revolutionizes high-dimensional vector retrieval. It offers an efficient and scalable solution for large-scale approximate nearest neighbor searches (ANNS) in high-dimensional spaces [29]. The key concept of PQ is the partitioning of high-dimensional vectors into several lower-dimensional subspaces, each independently quantized. This approach significantly reduces the complexity and storage cost of high-dimensional vector similarity operations. According to [30], PQ can achieve 90% memory compression without significant accuracy loss.

The PQ process begins with a training phase, where a codebook is created for each subspace using k-means clustering. Each vector is partitioned and assigned to the nearest centroid in its subspace, with the centroid index stored as a compact code, effectively compressing the data and reducing storage. During the retrieval phase, query vectors are similarly partitioned and quantized. The algorithm approximates the distance by summing precomputed distances between the query subvectors and the centroids of each corresponding subspace in the database. This asymmetric distance computation enables fast ANNS without decoding, achieving both storage efficiency and quick retrieval.

D. Motivation

In this section, we analyze the distribution of KV cache to reveal the limitations of existing low-precision quantization methods. This insight informs our proposed product quantization-based scheme for efficient KV cache compression.

1) Outliers hinder low-precision quantization. To explore the root cause of poor performance in existing KV quantization algorithms, we analyzed the KV distributions across different models, as shown in Fig. 2. Our findings reveal that in various models, outliers in the key cache are concentrated in several channels, while the value cache vectors also contain outlier points but without obvious anisotropy in their distribution. As shown in Fig. 2, quantization accuracy depends on the distribution range of the tensor for a given bit width. Concentrated tensor distributions yield higher quantization accuracy. However, outliers significantly expand the numerical distribution range, complicating low-precision quantization. While group-wise and mixed-precision quantization can mitigate outliers, they introduce additional overheads for quantization/de-quantization.

2) Exploring Opportunities for Non-uniform Quantization. We further explore the standard deviation distribution of KV across different channels in Fig. 3. Key standard deviations are notably prominent in certain channels, which we term as standard deviation outliers. In contrast, value standard deviations remain relatively small across channels. The channel-wise standard deviation indicates the range of numerical distribution across channels, corresponding to $\boldsymbol{X}max - \boldsymbol{X}min$ in Fig. 2. This suggests that integer quantization of keys is more challenging than values, necessitating distinct quantization precisions for different channels.

Given the presence of amplitude and standard deviation outliers in KV cache, we propose using PQ for KV cache compression. PQ splits vectors into multiple subspaces and clusters multiple channels within each subspace for quantization. Channels in the same subspace are uniformly quantized to 2^{nbits} states, where $nbits$ denotes the quantization bits of each subspace vector. Following PQ centroid training principles, channels with higher quantization difficulty will occupy more states in the PQ centroid, inherently supporting mixed precision quantization between channels.

III. Efficient KV Cache Compression via PQ

A. MILLION Framework Overview

This section illustrates the algorithm framework of the proposed MILLION, which enhances dynamic KV compression using the PQ algorithm. As shown in Figure 4, MILLION consists of three main phases: (1) offline PQ codebook training, (2) online prefill with KV cache quantization, and (3) online decode with KV cache quantization.

Offline PQ Codebook Training. The process begins with offline PQ codebook training of the KV cache, similar to obtaining activation distribution in activation quantization scenarios. During baseline inference, KV cache is sampled, and then

sampled keys and values are used for PQ training to determine the codebooks. Figure 4 a) illustrates this process: sampled key/value vectors are first partitioned into multiple sub-vectors ①, then the k-means algorithm is applied to find centroids for each sub-space ②. After that, the trained centroids are stored in GPU memory as codebooks for future quantization.

Online Prefill with KV Cache Quantization. During the prefill phase, the generated full-precision keys and values are used for attention computation ③, and quantized with the trained codebooks ④. The compressed KV (centroid indices) are then saved into the KV cache. The quantization and de-quantization process can be expressed as Eq. (4) and Eq. (5), respectively.

$$\begin{cases} \Delta_K = \|_{n=1}^{M} argmin(\|(K^i \times C_K^i)\|) \\ \Delta_V = \|_{n=1}^{M} argmin(\|(V^i \times C_V^i)\|) \end{cases} \tag{4}$$

$$\begin{cases} \hat{K} = \|_{n=1}^{M} \Delta_K^i \otimes C_K^i \\ \hat{V} = \|_{n=1}^{M} \Delta_V^i \otimes C_V^i \end{cases} \tag{5}$$

where M indicates the subspace number of PQ, $\|$ denotes concatenation of M sub-vectors, C_K^i and $C_V^i \in \mathbb{R}^{2^{nbits} \times d_k}$ are the i-th PQ codebooks of K and V. Δ_K^i and $\Delta_V^i \in \mathbb{R}^{n_k}$ are the quantized K and V of the i-th subspace, and $nbits$ denotes the the quantization bits for each subspace of PQ, so the 2^{nbits} is the codebook size of each subspace, and the symbol $\otimes$ means column-wise sampling C^i according to the indices of Δ_i. $\hat{K}$ and $\hat{V}$ represent the de-quantized K and V.

Online Decode with KV Cache Quantization. In the decode phase, the current token generates pairwise query, key, and value vectors. Based on the auto-regressive mechanism, the current token needs to compute attention not only with itself but also with all historical tokens ⑥. This phase requires traversing all codes in the KV cache and restoring them to float vectors by looking up the codebooks. The attention computation can be represented as:

$$A_n = \text{Softmax}(\frac{q_n \times [\hat{K}_{n-1}^T, k_n^T]}{\sqrt{d_k}}) \times [\hat{V}_{n-1}, v_n] \tag{6}$$

Where A_n denotes the attention output of the n-th token. We obtain $\hat{K}$ and $\hat{V}$ by Eq. (5). Lastly, the current key and value are quantized using codebooks and appended to the KV cache⑦.

B. Efficient Quantization Transformation

While PQ compression reduces KV cache memory demands, quantization and de-quantization overheads (as shown in Eqs. (4) to (6)) still persist during the decoding phase. This section introduces an efficient design to address these inefficiencies.

Attention Computation Transformation. We begin by transforming the attention computation as follows:

$$\begin{aligned} A_n =& \text{OlSoftmax}(\frac{q_n \times \|_{i=1}^{M} {C_K^i}^T \otimes {\Delta_K^i}^T}{\sqrt{d_k}}) \times \Big\|_{i=1}^{M} \Delta_V^i \otimes C_V^i \\ &+ \text{OlSoftmax}(\frac{q_n \times k_n}{\sqrt{d_k}}) \times v_n \end{aligned} \tag{7}$$

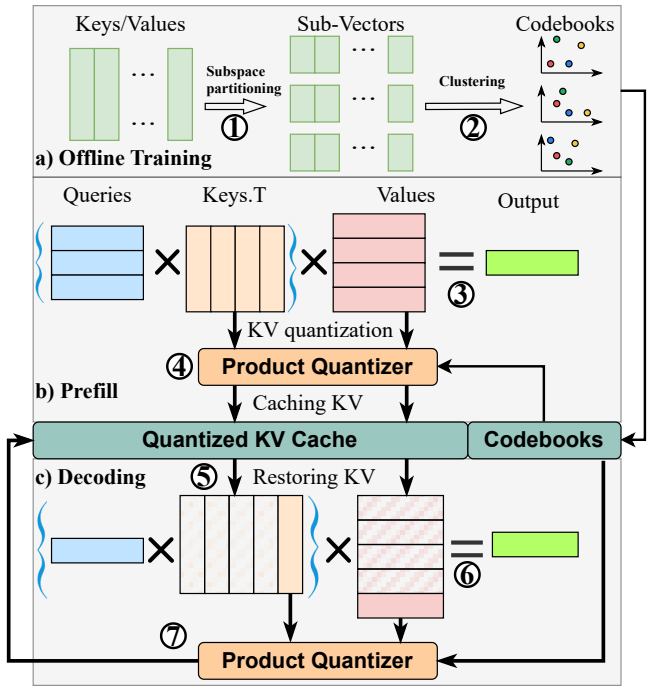


Fig. 4. An overview of MILLION algorithm framework. MILLION's algorithm framework consists of three parts: 1) offline PQ centroid training; 2) Prefill stage with KV quantization; 3) Decode stage with KV quantization.

Here, *OlSoftmax(·)* denotes an online softmax operation that splits the attention computation into two parts:

1) The current token's attention output to past tokens (first term $\text{OlSoftmax}(\frac{q_n \times \|_{i=1}^{M} {C_K^i}^T \otimes {\Delta_K^i}^T}{\sqrt{d_k}}) \times \|_{i=1}^{M} \Delta_V^i \otimes C_V^i$)
2) The current token's self-attention output (second term $\text{OlSoftmax}(\frac{q_n \times k_n}{\sqrt{d_k}}) \times v_n$)

The second part of the Eq. (7), dealing with the current token's self-attention, thus uses the full-precision key and value. It has $\mathcal{O}(1)$ computational complexity and requires no quantization or de-quantization for subsequent token generation. The first part, on the other hand, addressing attention to past tokens, is computed through two steps: 1) Calculate $q_n \times \|_{i=1}^{M} {C_K^i}^T$ with $\mathcal{O}(1)$ complexity; 2) Sampling the resulting vector with Δ_K, having $\mathcal{O}(n)$ complexity.

Efficient Attention Implementation. Given that Δ_K matrix columns are index vectors, we developed an efficient CUDA kernel. This kernel loads $q_n \times \|_{i=1}^{M} {C_K^i}^T$ as a look-up table into GPU L1 cache for fast random accessing during sampling and obtains the final result through sampling, thus the quantized Δ_K can be computed directly without de-quantization. Besides, the Δ_K matrix is prefetched with float4 data format, which further enhances the efficiency of continuous memory access. For instance, with 8-bit subspace product quantization ($nbits$ = 8), each thread in the CUDA kernel can simultaneously load 16 indexes into registers in float4 format, and then use these to perform 16 shared memory address lookups in int8 format. Furthermore, the kernel effectively paralleliz table lookup of Δ_K and de-quantization of Δ_v over CUDA thread blocks inspired by flash decoding.

Asynchronous Quantization Strategy. As of this point

TABLE I
DETAIL MODEL INFORMATION FOR EVALUATION.

	Parameters	positional embedding	sequence length
GPT2-xl	1.5 B	Absolute	1024
LLaMA-2-7B	7B	RoPE	4096
MPT-7B	7B	ALiBi	2048
Longchat-7B	7B	RoPE	32K
Yarn-LlaMA-2-7B	7B	RoPE	128K

TABLE II
PERPLEXITY EVALUATION RESULTS ON VARIOUS MODELS ACROSS DIFFERENT DATASETS.

Method	GPT2-xl		LLaMA-2-7B		MPT-7B	
	Wikitext-2	PTB	Wikitext-2	PTB	Wikitext-2	PTB
baseline	17.41	21	5.12	28.31	7.68	9.98
KVQuant-3b	18.59	23	11.21	12323.75	27681432	3E+07
KVQuant-3b-1%	18.24	22.35	5.22	24.34	7.826	10.14
MILLION-3b	17.6	21.14	5.2	29.55	7.79	10.08
KVQuant-4b	18.156	22.32	6.99	102.21	24043348	3E+07
KVQuant-4b-1%	18.108	22.15	5.14	25.86	7.71	10.02
MILLION-4b	17.57	21.12	5.21	29.56	7.75	10.03

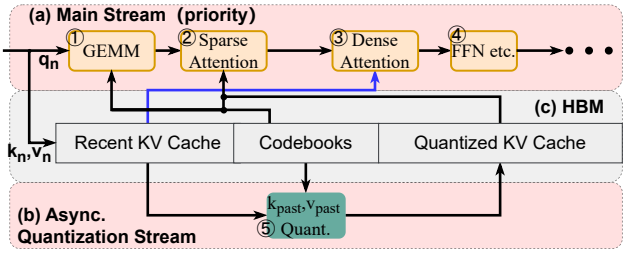


Fig. 5. Efficient Attention Implementation of MILLION with Asynchronous Quantization.

of introducing, quantization of k_n and v_n is still performed sequentially before the attention mechanism. However, cached codes are not needed until next token generation, indicating the computational independence of these two steps. Given the fact that decoding phase is memory-bounded, we assign quantization to a lower-priority CUDA stream to enable asynchronous execution of quantization and the subsequent processing, aiming at maximizing GPU utilization.

C. MILLION Dataflow

The dataflow of MILLION for LLM decoding is illustrated in Fig. 5. To minimize the impact of (de-)quantization on inference performance, MILLION employs two CUDA streams: 1) A high-priority main (CUDA) stream for attention computation; 2) A low-priority auxiliary (CUDA) stream is responsible for key and value quantization.

The process begins with the hidden state undergoing linear transformation to generate the query, key, and value for the current token, which serve as inputs to the attention layer. As analyzed in the previous section, the query performs a GEMM operation with the codebooks to calculate distances from each query to all centroids in the codebooks ①. The resulting distance vector is then used for sparse attention calculation with the quantized keys and values ②. Concurrently, the query undergoes dense attention computation with the recent KV cache (full precision) ③. Sparse and dense attention outputs are combined through online softmax and fed into subsequent operators for further inference ④. To avoid blocking the the main stream, keys, and values generated during decoding are temporarily stored in the recent KV cache. When GPU utilization of the main stream's CUDA kernel is low, the asynchronous quantization stream initiates the quantization kernel. It retrieves full-precision KV from the static recent KV cache for batch quantization ⑤. This design efficiently balances the computational demands of attention processing and quantization, optimizing overall inference performance.

IV. EXPERIMENTS

A. Experimental Setup

Models. Given the critical role of positional embeddings in the query and key components of LLMs, we conduct extensive experiments across various models employing different positional encoding techniques. Detailed specifications of these models are presented in Table I.

Tasks. We evaluate the quantization performance of MILLION across two distinct task categories: First, to assess model accuracy, we employ Perplexity (PPL), widely regarded as the most stringent metric, to compare the accuracy of various quantization schemes during the prefill stage. We conduct the PPL experiments on both Wikitext-2 and PTB datasets. Second, we conduct generation tasks using LongBench [31] for long context scenarios. This comprehensive assessment covered a range of linguistic challenges across various context lengths.

Baselines. We use the LLM model in float16 format as the baseline. We also compare our method with two state-of-the-art KV cache quantization works: KIVI [32] and KVQuant [33]. For these comparisons, we use the quantization bit widths recommended in their original papers. KIVI employs 2-bit and 4-bit quantization, while KVQuant adopts 3-bit and 4-bit quantization.[1]

B. Accuracy Results

PPL Evaluation. Table II compares the perplexity between KVQuant and MILLION. We reproduce KVQuant's evaluation on the Llama-2 model and extended the comparison to various models and datasets. KVQuant uses non-uniform quantization for all configurations in Table II. The results show that KVQuant, despite combining key/value quantization in different dimensions with non-uniform quantization, still incurs significant PPL loss. In contrast, full-precision sparse storage of 1% outliers achieves near-lossless performance at 3 bits. This demonstrates the impact of outliers on low-bit-width quantization. While isolating outliers for mixed-precision quantization effectively reduces KV cache quantization bit-width, it has drawbacks. This approach leads to unaligned memory access and introduces performance bottlenecks due to online sparse encoding, decoding, and computation. MILLION, employing uniform quantization, outperforms KVQuant in both 4-bit and 3-bit quantization[2]. Its accuracy is slightly lower than KVQuant's 1% outlier mixed-precision scheme. This indicates

[1]We conclude that the PPL of KIVI is huge and therefore omit it from the PPL experiment.

[2]We scanned different combinations of M and $nbits$ of PQ, and in this paper, the experiments adopted the best accuracy conditions of 3b and 4b, which were (64, 8) and (32, 12), respectively.

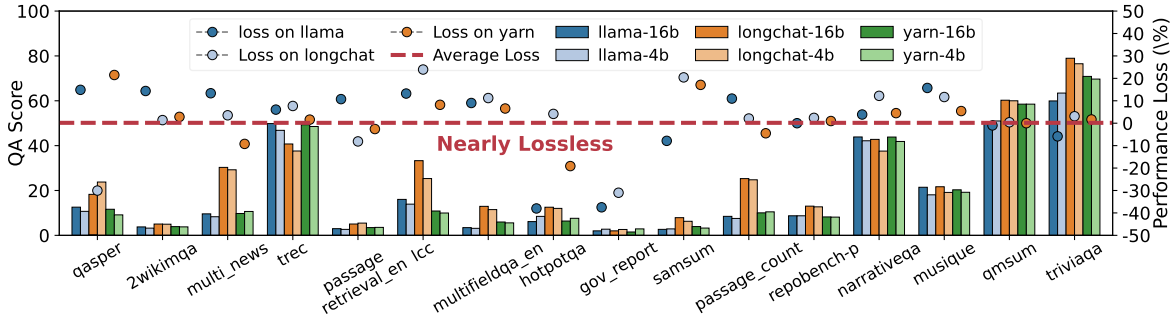


Fig. 6. Performance evaluation of MILLION on various models across a range of benchmarks in LongBench. While 16b denotes the baseline performance with fp16 KV cache, and the 4b depicts the MILLION performance of 4 bits KV quantization.

TABLE III
THE INFLUENCE OF 1% OUTLIERS ON PPL ON KVQUANT AND MILLION

	kv-3b	kv-3b -1%	sensitivity -3b	kv-4b	kv-4b -1%	sensitivity -4b
KVQuant	11.21	5.22	**53.4%**	6.99	5.14	**26.5%**
MILLION	5.2	5.22	**-0.38%**	5.21	5.18	**0.58%**

TABLE IV
TPOT (MS) COMPARISON OF KV QUANTIZATION METHODS.

Prefill Length	1K	2K	4K	8K	16K	32K
Baseline(fp16)	32.53	35.64	42.04	54.83	80.49	132.97
KIVI(4b)	46.69	46.88	46.92	47.86	OOM	OOM
KVQuant(4b)	75.73	73.92	75.34	74.90	78.17	90.16
MILLION(4b)	30.36	31.57	34.05	38.34	46.53	63.41

that MILLION effectively handles low-bit-width quantization with outliers while maintaining scheme efficiency.

LongBench Evaluation. Figure 6 shows MILLION performance on various models across LongBench benchmarks, the residual block size is set to 0 for stress evaluation. While 4-bit quantization shows some performance degradation compared to full-precision 16-bit models, it maintains sufficient accuracy for many applications. Specifically, MILLION with 4-bit KV cache quantization resulted in a score drop of 0.95 for llama-2-7b and 0.93 for longchat-7b. Notably, MILLION improved the score of yarn-llama-2-7b (with a $128K$ context) by 0.45, demonstrating its effectiveness for models with the longest contexts.

Outlier-immune Study. To verify MILLION's sensitivity[3] to outliers, we conduct additional experiments, as shown in Table III. We identified the top 1% of outliers in the KV cache based on data distribution and stored them in a sparse full-precision format. After applying MILLION to the filtered KV cache on LLMs, the PPL improved by only -0.38% and 0.58% in the 3-bit and 4-bit quantization scenarios, respectively. This marginal improvement suggests that MILLION is naturally robust to outliers, and incorporating sparse outlier handling offers little benefit, *while significantly degrading end-to-end system performance.*

C. System Performance

Generation Speed: We compare the performance of different KV quantization schemes by running Llama2-7b inference tasks on an A40 GPU. We assessment the average time per output token (TPOT) across different prefill lengths with 100 tokens generation. All schemes use 4-bit KV quantization,

[3]Sensitivity represents the proportion of PPL reduction caused by considering 1% sparse outliers.

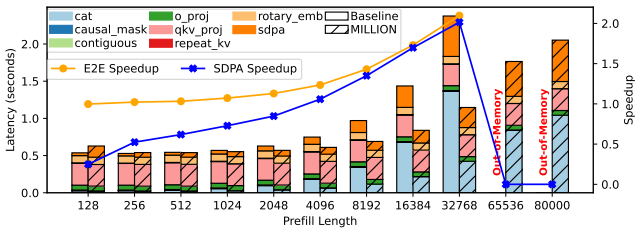


Fig. 7. Latency breakdown analysis. MILLION significantly outperforms the baseline on two KV access operators, the performance margins increasing with the severity of the memory bottleneck.

with results shown in Table IV. KIVI, which uses a simpler quantization scheme, outperforms the baseline at 8K context lengths. In contrast, KVQuant's non-uniform quantization significantly slows inference, and the outlier sparsity scheme further reduces performance. MILLION reduces computational complexity while maintaining high compression, delivering superior performance even at 1K context length.

Latency Breakdown: Fig. 7 shows the latency breakdown of each operator to highlight the performance gains in the MILLION system. Compared to the baseline, MILLION achieves significant performance improvements in two operators: Scaled Dot Product Attention (SDPA), which performs the attention operation, and Concatenate (cat), which manages the KV cache. This demonstrates that MILLION suppresses the access bottleneck in the generation phase by compressing the KV bitwidth, while reducing the computational complexity of attention further enhances performance. In contrast, other compression schemes incur higher computational overhead due to de-quantization. Besides, Fig. 7 shows the SDPA and end-to-end performance gains at different context lengths. The speedup of MILLION increases with context length, reaching 2.01x and 2.09x in attention and end-to-end acceleration ratios, respectively, at 32K context length.

V. CONCLUSION

We present MILLION, a novel and efficient PQ-based quantization framework for LLM inference. MILLION efficiently compresses data while preserving accuracy. It incorporates asynchronous quantization, enhancing inference speed and reducing additional operations caused by hardware-inefficient quantization/dequantization for KV cache compression. Experiments demonstrate that MILLION achieves better model performance while maintaining low latency.

REFERENCES

[1] Y. Chen, S. Qian, H. Tang, X. Lai, Z. Liu, S. Han, and J. Jia, "Longlora: Efficient fine-tuning of long-context large language models," *arXiv preprint arXiv:2309.12307*, 2023.

[2] anthropic, "Introducing claude 3.5 sonnet \ anthropic," https://www.anthropic.com/news/claude-3-5-sonnet, (Accessed on 07/09/2024).

[3] openai, "Models - openai api," https://platform.openai.com/docs/models, (Accessed on 07/09/2024).

[4] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[5] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee, "Taming throughput-latency tradeoff in llm inference with sarathi-serve," *arXiv preprint arXiv:2403.02310*, 2024.

[6] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *arXiv preprint arXiv:1911.02150*, 2019.

[7] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.

[8] A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[9] Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei, "You only cache once: Decoder-decoder architectures for language models," *arXiv preprint arXiv:2405.05254*, 2024.

[10] S. Agarwal, B. Acun, B. Homer, M. Elhoushi, Y. Lee, S. Venkataraman, D. Papailiopoulos, and C.-J. Wu, "Chai: Clustered head attention for efficient llm inference," *arXiv preprint arXiv:2403.08058*, 2024.

[11] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[12] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[13] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[14] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao, "Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference," *arXiv preprint arXiv:2405.12532*, 2024.

[15] J. Dai, Z. Huang, H. Jiang, C. Chen, D. Cai, W. Bi, and S. Shi, "Corm: Cache optimization with recent message for large language model inference," 2024. [Online]. Available: https://arxiv.org/abs/2404.15949

[16] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "Snapkv: Llm knows what you are looking for before generation," *arXiv preprint arXiv:2404.14469*, 2024.

[17] M. Adnan, A. Arunkumar, G. Jain, P. Nair, I. Soloveychik, and P. Kamath, "Keyformer: Kv cache reduction through key tokens selection for efficient generative inference," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 114–127, 2024.

[18] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 692–705.

[19] J. Y. Yang, B. Kim, J. Bae, B. Kwon, G. Park, E. Yang, S. J. Kwon, and D. Lee, "No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization," *arXiv preprint arXiv:2402.18096*, 2024.

[20] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," *arXiv preprint arXiv:2406.10774*, 2024.

[21] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh, "Extreme compression of large language models via additive quantization," *arXiv preprint arXiv:2401.06118*, 2024.

[22] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," *arXiv preprint arXiv:2404.00456*, 2024.

[23] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[24] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," 2024. [Online]. Available: https://arxiv.org/abs/2306.00978

[25] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," *arXiv preprint arXiv:2405.04532*, 2024.

[26] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[27] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella *et al.*, "Rwkv: Reinventing rnns for the transformer era," *arXiv preprint arXiv:2305.13048*, 2023.

[28] T. Munkhdalai, M. Faruqui, and S. Gopal, "Leave no context behind: Efficient infinite context transformers with infini-attention," *arXiv preprint arXiv:2404.07143*, 2024.

[29] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2010.

[30] H. V. Simhadri, G. Williams, M. Aumüller, M. Douze, A. Babenko, D. Baranchuk, Q. Chen, L. Hosseini, R. Krishnaswamny, G. Srinivasa *et al.*, "Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search," in *NeurIPS 2021 Competitions and Demonstrations Track*. PMLR, 2022, pp. 177–189.

[31] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou *et al.*, "Longbench: A bilingual, multitask benchmark for long context understanding," *arXiv preprint arXiv:2308.14508*, 2023.

[32] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "Kivi: A tuning-free asymmetric 2bit quantization for kv cache," *arXiv preprint arXiv:2402.02750*, 2024.

[33] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "Kvquant: Towards 10 million context length llm inference with kv cache quantization," 2024. [Online]. Available: https://arxiv.org/abs/2401.18079

RAGNAR: Exploring Volatile-Channel Vulnerabilities on RDMA NIC

Yunpeng Xu*, Yuchen Fan*, Teng Ma†, Shuwen Deng*‡§
*Department of Electronic Engineering, Tsinghua University, †Alibaba Group, ‡Zhongguancun Laboratory
*{xyp24, fanyc22}@mails.tsinghua.edu.cn, shuwend@tsinghua.edu.cn, †sima.mt@alibaba-inc.com

***Abstract*—With the surge in data computation, Remote Direct Memory Access (RDMA) becomes crucial to offering low-latency and high-throughput communication for data centers, but it faces new security threats. This paper presents RAGNAR, a comprehensive suite of hardware-contention-based volatile-channel attacks leveraging the under-explored security vulnerabilities in RDMA hardware. Through comprehensive microbenchmark reverse engineering, we analyze RDMA NICs at multiple granularity levels and then construct covert-channel attacks, achieving 3.2x the bandwidth of state-of-the-art RDMA-targeted attacks on CX-5. We apply side-channel attacks on real-world distributed databases and disaggregated memory, where we successfully fingerprint operations and recover sensitive address data with 95.6% accuracy.**

I. INTRODUCTION

With the boom of data computation, the data center has prevailingly used Remote Direct Memory Access (RDMA) [31] for low latency and high throughput [41]. RDMA allows kernel bypass and zero-copy, crucial for large-scale and real-time data processing [8], [20].

However, recent studies have shown that RDMA also opens up new attack vectors [13], [17], [34], [37]. These security threats can significantly compromise the data security of RDMA-related systems.

Given the extensive attack surface, we ❶ *provide a systematic abstraction and analysis of RDMA security*, categorizing them from the perspectives of HW/SW, RDMA-targeted/related, granularity, and attack types. While several studies [17], [18], [22], [33], [34], [37] cover various aspects of RDMA security, significant gaps remain. We propose RAGNAR[1], which addresses these uncharted security concerns by exploring the potential of RDMA-targeted contention-based volatile-channel[2] covert-channel and side-channel attacks.

To better understand RDMA hardware, we ❷ *comprehensively reverse engineer RDMA NICs (RNICs) with four different levels of traffic granularity*. We analyze hardware resource contention between flows across traffic classes, memory regions (MRs), and address offsets within the same MR, and spot new hardware vulnerabilities.

We then ❸ *build covert-channel attacks given the reverse-engineered results, achieving 3.2× bandwidth compared with state-of-the-art RDMA covert channels on the same setup*. Our attacks achieve a data rate of 84.3 Kbps on CX-6 and 63.6 Kbps on CX-5, outperforming PYTHIA's covert channel (20 Kbps on CX-5) [37] and not being mitigated by state-of-the-art RDMA isolation defenses [22].

Furthermore, we ❹ *perform side-channel attacks on distributed database [23] and disaggregated memory [39]* to demonstrate the real scenario impact of RAGNAR. We fingerprint different shuffle/join application patterns and target a key-value store application, obtaining information about the victim's access address with 95.6% accuracy.

§Shuwen Deng is the corresponding author.

[1]Ragnar Lodbrok is a legendary Norse hero known for his multiple conflicts and raids, symbolizing intense warfare and struggle. His many battles represent significant interference and opposition, mirroring the volatile contention that happened in RDMA attacks.

[2]Volatile channels refer to the cases that the sender and receiver share the resources on the fly, which maps to the hardware contention scenario previous RDMA attacks do not explore and we will introduce more in Section II.

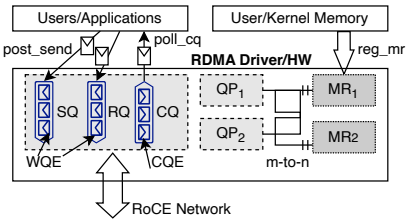


Fig. 1: Data Structures and Architecture of RDMA

Our main contributions are as follows:

- Propose granularity levels and analyze the stealthiness of RDMA-targeted HW attacks and mitigation with such metrics.
- Comprehensively reverse-engineer RDMA hardware from four different traffic granularity, especially reconstructing information on traffic priority and address offset.
- Construct 3 different granularity-level covert-channel attacks achieving 3.2× bandwidth compared with state-of-the-art attack.
- Build 2 side-channel attacks on real-world applications including distributed database and disaggregated memory, achieving up to accuracy of 95.6% on recovering victim address.

The code used in this paper will be released under an open-source license at https://github.com/THU-HAS/Ragnar.

II. BACKGROUND, RELATED WORK AND ATTACK ANALYSIS

A. Remote Direct Memory Access (RDMA)

RDMA is a network technology enabling direct memory access between physical servers, bypassing the OS [4], [7], [14], [30], [31], [35]. For a connection, each side generates queue pairs (QPs) for request queuing, registers memory region (MR) for access control, and exchanges MR addresses, as shown in Figure 1. Once connected, both sides post work queue entries (WQEs) and poll completion queue entries (CQEs) for message exchange.

B. Side-Channel and Covert-Channel Attacks

Side-channel attacks leverage indirect information of the architecture and hardware leakage to extract secrets [15], [16], [28]. *Covert-channel attacks* are where the sender and receiver establish covert communications through indirect information channels.

These channels can be categorized into *persistent channels* and *volatile channels* [42]. Persistent channels utilize state changes, e.g., the cache state attacks [1], [3], [5], [11], [28]; volatile channels utilize resource sharing on the fly, e.g., exploiting port contention in the execution engine [2], or contend network-on-chip (NoC) [9], [27].

C. Security Issues on RDMA

RDMA introduces variant security issues. We categorize these issues from two perspectives: SW/HW, RDMA-targeted/related.

- **RDMA-targeted/related SW issues.** Software implementation issues by REDMARK [33], and one-sided non-auditability [32].

TABLE I: Comparisons with prior works on RDMA-targeted HW attacks. Attack Types and Granularity Levels are discussed in II-D.

	Type	Grain	Defended	Channel	Steal.
Zhang [43]	P	II	[22]	-	Medium
Kong [18]	P	II	[22]	-	Medium
HUSKY [17]	P	II	[22]	-	Medium
Kim [13]	S	I	-	Volatile	Low
PYTHIA [37]	C+S	IV	[3]	Persistent	High
RAGNAR	C+S	I/II/III/IV	-	Volatile	High

- **RDMA-targeted HW issues.** Hardware vulnerabilities of RNICs on performance isolation [43], [18], [17], and persistent covert-channel issues with onboard cache by PYTHIA [37].
- **RDMA-related HW issues.** RNIC exacerbates vulnerabilities together with other components in RDMA-enabled systems, e.g., DRAM, DDIO, by THROWHAMMER [36] and NETCAT [19].

When we focus on RDMA-targeted vulnerabilities, we find limitations of the existing works on the hardware side, where we propose new hardware vulnerabilities and build new volatile-channel attacks.

D. Analysis and Comparison on RDMA-Targeted HW Vulnerabilities

We sort out works on RDMA-targeted HW security issues and compare them with our work, RAGNAR, in TABLE I.

Attack Types. RDMA-Targeted HW Vulnerabilities are categorized into 3 types: 1) *Performance (P)* performs availability attacks on bandwidth or PFC; 2) *Covert (C)* performs covert-channel attacks; and 3) *Side (S)* performs side-channel attacks.

Granularity Levels. For attacks, the levels refer to the granularity of resources or parameters the attacker manipulates; for defenses, they refer to counters or metrics the defender monitors. We divide the granularity into the coarsest Grain-I to the finest Grain-IV.

- **Grain-I**. Traffic pressure. Attackers can manipulate traffic pressure to exert influence on bandwidth. For defenses, modern RNIC provides native Grain-I per-traffic-class counters and flow control to detect and defend Grain-I attacks easily.
- **Grain-II**. Main traffic patterns, e.g., traffic classes, opcodes, and message sizes. Several studies [17], [18], [43] manipulate Grain-II parameters to apply availability attacks. For defenses, HARMONIC [22] provides Grain-II counters on RDMA opcodes for performance isolation to mitigate these Grain-II attacks.
- **Grain-III**. RDMA-specific resources engaged, e.g., PDs, QPs, MRs. For attacks, variant RDMA resources can be utilized to exert influence. For defenses, HARMONIC also provides RDMA-specific resource utilization counters to detect Grain-III attacks.
- **Grain-IV**. Addresses and other detailed parameters of traffic patterns. If an attacker keeps accessing different addresses with such parameters, as done by PYTHIA [37], typical Grain-I-to-III defenses cannot detect the existence of such an attacker.

As discussed above, Modern RNIC provides native Grain-I counters for resource isolation and detection to mitigate Grain-I attacks. Several studies [17], [18], [43] exploits Grain-II performance issues that bypass native Grain-I PFC mitigation but are defended by the consequent work HARMONIC [22] providing Grain-II/III counters.

Kim [13] implements side-channel attack [13] utilizing PCIe contention, but is not fine-grained enough[4]. PYTHIA [37] utilizes Grain-IV address manipulation to build side-channel attacks with onboard caches. Due to fine granularity, PYTHIA can bypass Grain-I-to-III counters and mitigations provided by RNIC and HARMONIC. However, PYTHIA is a cache-based attack, so general cache-attack detection and mitigation [38] can be applied to mitigate PYTHIA. Also, the PYTHIA PTE attack can be mitigated by widely-used huge pages [37].

[3]Defenses are discussed in Tsai's work [37]. Using huge pages or physical addresses can mitigate PTE-based attacks, and sniffers can easily detect evicting operations.

[4]It can only steal coarse information if the GPU is running rather than reveal detailed data.

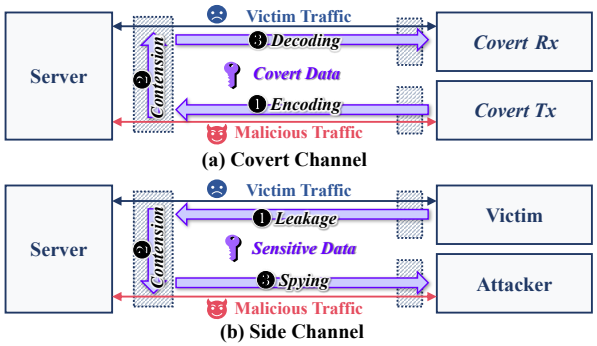


Fig. 2: Threat models of (a) covert channel and (b) side channel. Light blue boxes are hosts, and dotted boxes are RNICs. Blue and red arrows show the victim and malicious traffic, respectively. Purple arrows show the covert/sensitive data flow in contention channels.

TABLE II: Specifications of Test Environment

Host	Processor	RNIC	OS	RAM
H1	AMD EPYC 9554	CX6	Ubuntu 20.04	755GB
H2	Intel Xeon S4314	CX4,5	Ubuntu 18.04	256GB
H3	Intel Xeon P8480+	CX4-6	Ubuntu 22.04	1TB

TABLE III: Parameter Sheet of CX-4 to CX-6 Network Adapters

Feature	ConnectX-4	ConnectX-5	ConnectX-6
Speed	25Gbps	100Gbps	200Gbps
PCIe Interface	PCIe 3.0 x8	PCIe 3.0 x8	PCIe 4.0 x16

Comparison with related work. Compared with prior works, RAGNAR utilizes up to Gran-IV to achieve high stealthiness and bypass state-of-the-art defense HARMONIC. Compared with PYTHIA, RAGNAR is based on the processing path in RNICs. It does not rely on cache mechanism. Volatile channel brings our attack higher stealthiness and more general applicability.

III. THREAT MODEL AND EXPERIMENTAL SETUP

A. Threat Model

In our attack scenario, three parties are involved, as illustrated in Figure 2. The server hosts data, such as an in-memory database or key-value store. One client keeps regular traffic to access the server's in-memory data via RDMA, while the other client controls malicious traffic to perform the attack.

For (a) covert channels, the covert *Tx* ❶ sets up malicious traffic to encode covert data, and ❷ transfers the encoded data to observable effects in the victim traffic, which ❸ is decoded by covert *Rx*.

For (b) side channels, the victim ❶ leaks sensitive data in its traffic and ❷ influences the malicious traffic through the contention channel. ❸ The attacker sets up and observe variant malicious traffic at the same time to recover sensitive data and the access patterns of the victim.

B. Experimental Setup

We run all the tests and attacks across different hosts and environments, including various processors from Intel and AMD listed in TABLE II, as well as various network cards such as CX-4, CX-5, and CX-6 listed in TABLE III.

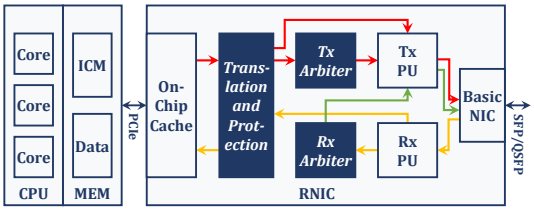


Fig. 3: Framework of RDMA System. Red arrows are the Tx flow, yellow arrows are the Rx flow, and green arrows are the replying flow. Lightboxes are microarchitectures whose existence and basic processing logic are known. Dark boxes are exponents that are black boxes to us, where we find new features by reverse engineering. [6]

IV. Reverse-Engineer RDMA NICs

The deployment of RDMA networks is contingent upon the support of RNICs. To develop RDMA-specific covert-channel and side-channel attacks, we provide some reverse-engineering results for a better understanding of RNICs in this section.

A. Architectural and Microarchitectural Analysis

Due to the black-box nature of detailed structure, we provide an architectural and microarchitectural picture of the RNIC, which are recovered from ETHTOOL [25] bps and pps counters and description of existing literature[5] and Nvidia manual [24], [26], as shown in Figure 3.

The RNIC is usually connected to the network with a Fiber Optic Interface and to the CPU/MEM with PCIe. MRs are pinned to physical memory to avoid page faults in the data path. The RNIC typically has the same functional parts as general network cards (Basic NIC) with typical L1 to L3 layer operations. Main RDMA-specific microarchitectures on the RNIC are shown in Figure 3.

In Figure 3, the red arrow indicates the Tx flow of outbound traffic. The RNIC reads the SQE from the host and arbitrates among all the SQEs. Requests are sent to the processing part according to RDMA opcodes and then handed over to the basic NIC for non-RDMA-specific procedures.

The yellow arrow indicates the Rx flow of inbound traffic. The basic NIC hands over the received and parsed message to the Rx processing pipeline.

RDMA Reads, as well as reliable service, need ACK packets, forming a reverse flow (green arrows). The reverse flowis arbitrated and processed as the outbound flow.

B. Grain-I/II Contention on Different-Priority Traffics

We identify *arbiters* that arbitrate packets of different flows within the RDMA hardware pathway. This arbitration impacts RDMA traffics in very different priorities in contention scenarios depending on parameters like opcodes, QP count, and message size.

Setup and Priority Indication. We systematically test the priority behavior of traffic during contention to evaluate the impact of different RDMA operations on competing flows. Using the `mlnx_qos` tool [26], we configure two traffic flows in ETS mode, each allocated 50% of the bandwidth. *However, we observe unbalanced bandwidth affected by priority mechanisms and hardware arbiters.* We configure traffic with varying QP numbers and message sizes and monitor changes in bits per second (bps) and packets per second (pps).

Comprehensive study on priority effects between traffics. We run a comprehensive benchmark of over 6000 parameter combinations to describe traffic performance in competitive scenarios. Analysis of the results yields a conceptual diagram (Figure 4) illustrating competition patterns under typical parameters. From Figure 4, we have the following key observations:

- The competition between RDMA Read and Write operations is complex and non-monotonic. As shown in the blue-outlined section ❶ of Figure 4, when message sizes are small, RDMA Write traffic loses over 50% of its bandwidth due to competition. However, once the write message size reaches around 512 Bytes, its bandwidth increases significantly, causing RDMA Read traffic bandwidth to drop by 30% to 80%.
- The RDMA Atomic operation exhibits a similar trend when competing with RDMA Read/Write operations, as illustrated in the orange-outlined section ❷ of Figure 4.
- In some cases, competition increases both traffic flows, resulting in total traffic exceeding 200% of the original single flow, as demonstrated in the green-outlined section ❸ of Figure 4.
- RDMA Write and reverse RDMA Read traffic with identical parameters show different competition dynamics when competing with RDMA Write traffic, highlighted in the yellow-outlined section ❹ of Figure 4.

From the analysis of the above observations, we can derive the following findings:

> **Key Finding 1: Non-monotonic bandwidth contention.** When the competing write flow is small, only the medium read flow experiences a significant drop, while the bandwidth of both the small and large read flows remains unaffected. However, as the competing write flow increases, the situation is reversed.
> **Key Finding 2: Abnormal bandwidth increment.** Contention of small RDMA Writes traffic can lead to abnormal BW increment in both traffic, which can be related to NoC activation.
> **Key Finding 3: Arbiter priority.** The logical Tx arbiter has a higher priority than the logical Rx arbiter.

C. Grain-III/IV Contention within Memory Region

We then post the fixed pattern of requests in a single-threaded manner and involve a new metric to explore finer-grained features.

Unit latency increase. The latency Lat_{total} from completing the `ibv_post_send` to polling a completion with `ibv_poll_cq` is a more precise and stable per-message observable than throughput statistics. Since Lat_{total} includes the queuing delay in the WQ before actual transmission, we assume that Lat_{total} should have a linear relationship with the queue length as $Lat_{total} = k \cdot (len_{sq} + 1) + C$. len_{sq} is the number of WQEs in front of the measured WQE. k stands for the latency increase caused by each queuing request, namely the *Unit Latency Increase (ULI)*. C is some len_{sq}-independent constant term caused by other overhead. Both theoretical analysis [7] and experiments [8] support the linearity and $C \approx 0$. So we use $ULI \approx Lat_{total}/(len_{sq} + 1)$ to characterize the contention.

[5]Only a few studies provide their original description. PYTHIA [37] proved the existence of multiple on-chip caches and reverse-engineered their parameters, Kalia et al. [12] illuminates the software-hardware interface data flows of various RDMA operations.

[6]The TxPU/RxPU and TxArbiter/RxArbiter in the figure are logically separated, but this does not mean that they are deterministically separated parts in hardware. For the convenience of discussion, we will still consider them separately later.

[7]If we consider an SQ reaching the maximum send queue size $len_{sq,max}$ in the stable traffic case, all these $len_{sq,max}$ work requests on the fly should be uniformly distributed in a Lat_{total} round trip time, where ULI should be $k = Lat_{total}/len_{sq,max} = Lat_{total}/(len_{sq} + 1)$.

[8]The linear relationship fits well with Pearson correlation coefficient = 0.9998, and C can be neglected.

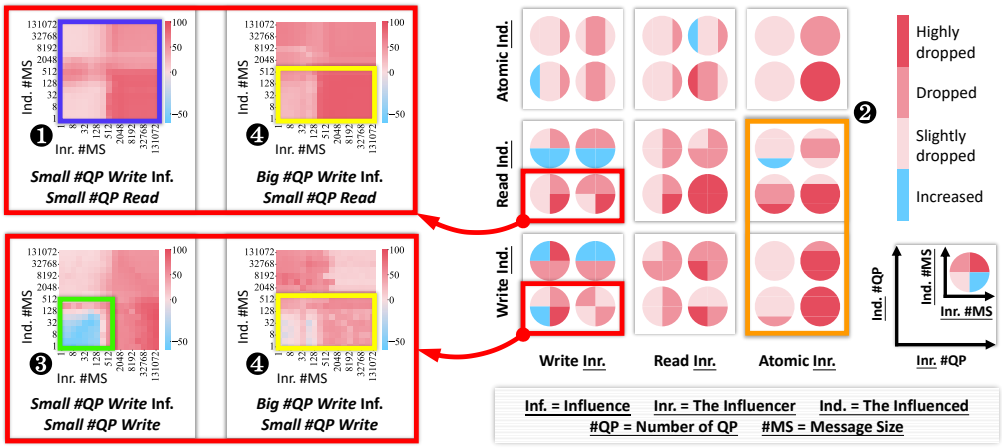


Fig. 4: Conceptual traffic priority diagram. This figure illustrates the competition-caused reduction in one party's traffic when competing with another under various conditions. For instance, the subplot of Write Inr. and Read Ind. shows a decrease in Read traffic when competing with Write traffic. The axes of each subplot represent qp number (qp_num), and the pie chart sections show results for different message sizes, as depicted in the legend. Dark red indicates no significant decrease, medium red a 50% decrease, light red a slight decrease, and blue an abnormal increase. These pie charts summarize measured data, with details expanded on the left side and key phenomena highlighted.

TABLE IV: Parameters of Grain-III/IV microbenchmark. *a*@MR#*b* means the address with offset *a* to the base address of the *b*th MR.

Operation	Msg Size	Remote Address 1	Remote Address 2
RDMA Read	1 - 1MB	0 @ MR#0	0 - 4KB @ MR#0
RDMA Read	1 - 1MB	0 @ MR#0	0 @ MR#1
RDMA Read	1 - 1MB	0 - 4KB @ MR#0	same as RA1

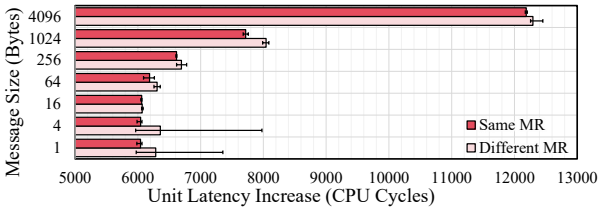


Fig. 5: ULI vs. same/different remote MRs vs. message size. Alternately accessing two addresses in the same/different remote MRs on CX-4 with RDMA Reads. Average and 10/90-percentile are shown.

Setup. We establish MRs on 2 MB huge pages, use 2 QPs, and set all resources within the same PD. We disable DDIO and bind the benchmark process to the same CPU core in the same NUMA node with the RNICs to eliminate the differences in address translation and context switching. Detailed configurations are shown in TABLE IV. [9] We perform the tests on CX-4, CX-5, and CX-6 RNICs.

With the Grain-II parameters unchanged, we focus on a more covert parameter, *remote address*. We find that the remote address of RDMA Reads can have a significant impact on the datapath contention. This impact can be reflected in ULI as the following:

- Latency distinction does not only occur when accessing different/same MR for RDMA Reads (Figure 5), but also when accessing variant addresses, with huge pages enabled, DDIO disabled, and affects of NUMA and cache excluded (Figure 6).
- The latency pattern caused by accessing different remote address offsets varies with message sizes but shows similar 2's power periodicity. As shown in Figure 6, 7 and 8, stable latency drops occur at addresses aligned with 8 Bytes under RDMA Reads. More significant drops appear at addresses multiples of 64 Bytes. An apparent periodicity at 2048 Byte intervals occurs as well.
- The absolute address offset (to the base address of the MR) and the relative address offset (between consequent RDMA Reads) have different impacts on latency, shown in Figure 6 and 8. The relative address offset effect indicates the mutual interaction among different packets due to complex mechanisms in the Translation and Protection module in Figure 3.

[9]Besides, we have tried RDMA Write, different MR sizes, and different local addresses. Yet, these variations do not bring up stable and observable effects.

Key Finding 4: Offset effect. Remote address offsets affect ULI in variant 2's power periodic manners for RDMA Reads.

V. Covert-Channel Ragnar Attacks

A. Covert-Channel Attack Setup

In this section, we present three covert-channel attacks. In our threat model (Figure 2(a)), the covert Tx/Rx cannot communicate mutually but share RDMA-based service with the same server. In V-B, the clients do not need to share a common address space, while in V-C and V-D, the clients are only permitted to perform certain RDMA Reads. In both cases, they cannot communicate directly.

The approach for these covert channel attacks is illustrated as follows: The sender modifies the mode of resource X based on the covert bits, which influences the receiver's observed metric of resource Y, allowing the receiver to infer the transmitted information.

B. Inter-Traffic-Class Priority-Based Channel

In this attack, the covert Rx maintains a flow with a small bandwidth, continuously monitoring its bandwidth (resource Y). Meanwhile, the covert Tx encodes bits 1/0 using the bandwidth of another flow (resource X). Covert Rx can infer the transmitted bits by observing Y.

Figure 9 shows our results on three RNICs, using the covert channel to transmit the bitstream `1101111101010010`. The TX performs RDMA Writes with 128 Bytes (bit `1`) and 2048 Bytes bit `0`). The results clearly show distinct bandwidth for bits `1` and `0`. This attack has a meager error rate because priority-based covert channel attacks rely on traffic bandwidth.

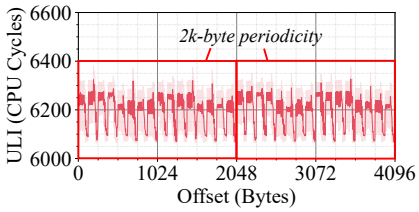


Fig. 6: ULI vs. *absolute* address offset. Alternately accessing two addresses with *64 B* RDMA Reads with the same remote MR on CX-4. Red lines and zone show the average and 10/90-percentile.

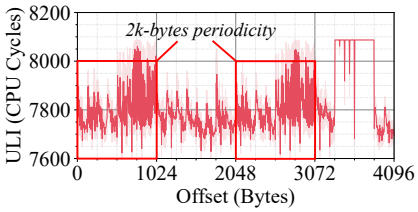


Fig. 7: ULI vs. *absolute* address offset. Alternately accessing two addresses with *1024 B* RDMA Reads with the same remote MR on CX-4. Red lines and zone show the average and 10/90-percentile.

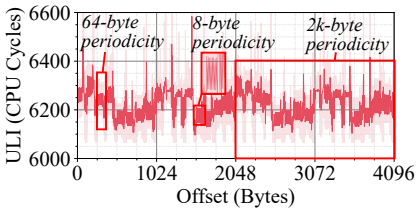


Fig. 8: ULI vs. *relative* address offset. Alternately accessing two addresses with *64 B* RDMA Reads with the same remote MR on CX-4.Red lines and zone show the average and 10/90-percentile.

TABLE V: Design features and evaluations of different granularity level covert-channels attacks on CX-4, CX-5, and CX-6.

Covert Channel	**Inter Traffic-Class**			**Intra Traffic-Class**					
				Inter MR			**Intra MR**		
Granularity Level	I+II			III			IV		
Base	Priority			RDMA resources			Offset effect		
RNIC (ConnectX)	CX-4	CX-5	CX-6	CX-4	CX-5	CX-6	CX-4	CX-5	CX-6
Bandwidth	1.0 bps	1.1 bps	1.1 bps	31.8 Kbps	63.6 Kbps	84.3 Kbps	32.2 Kbps	31.5 Kbps	81.3 Kbps
Error Rate	0.00%	0.00%	0.00%	5.92 %	3.98 %	7.59 %	6.95%	4.84%	4.08%
Effective Bandwidth	1.0 bps	1.1 bps	1.1 bps	21.5 Kbps	48.3 Kbps	51.6 Kbps	20.5 Kbps	22.7 Kbps	61.3 Kbps

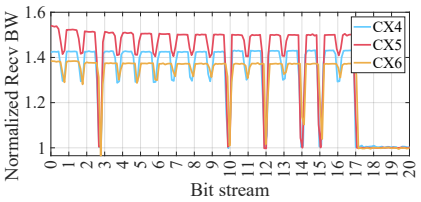


Fig. 9: Priority-based covert-channel attacks on CX-4/5/6. The significant drop means bit 0, and the slight drop means bit 1.

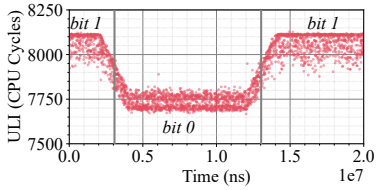


Fig. 10: Covert bits decoded from different unit latency increase, 1024 B RDMA Read, Max Send Queue Length = 256, on CX-4.

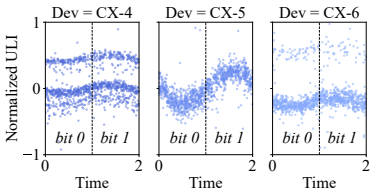


Fig. 11: Inter-MR resources-based channel on CX-4/5/6. X-axes indicate a folded period of two covert bits; Y-axes are normalized ULI.

C. Inter-MR Resource-Based Channel

We further employ Grain-III parameters, accessing the same or the different MRs (resource X) to encode the covert bits and measure the ULI (resource Y) of the background traffic.

Figure 10 demonstrates the folding ULI pattern with periodically switching covert bitstream. This ULI distinction remains stable over tens of seconds. More tests are done on CX-4, CX-5, and CX-6, utilizing 2 MB MR and 2 QPs. Under the best parameter combinations[10], the results are shown in Figure 11. As shown in TABLE V, RAGNAR's covert-channel attack on CX-6 reaches 84.3 Kbps bandwidth with 51.6 Kbps effective bandwidth.

D. Intra-MR Address-Based Channel

For better stealthiness, we manipulate Grain-IV parameters to construct an intra-MR channel. We switch address offsets (resource X) and observe ULI (resource Y) with 512 B RDMA Reads.

Under the best parameter combinations[11], the attack achieves 78.0 Kbps bandwidth on CX-6 with an error rate of 4.08 %, shown in TABLE V. Compared to the inter-MR channel, it offers higher stealthiness. This is because that encoding the covert bits brings nothing more than normal variation of access address offsets.

[10]Operations are 512 B Read, 64 B Read, 512 B Read respectively; max send queue sizes are 10, 6, 6 respectively.

[11]Max send queue size is set to 8. Covert bits are encoded to 0 B/255 B address offsets for CX-4 and CX-5, and 0 B/257 B address offsets for CX-6.

VI. SIDE-CHANNEL RAGNAR ATTACKS

In this section, we establish side-channel RAGNAR attacks on real-world applications.

We illustrate the threat model in Figure 2(b), where both the attacker and the victim are clients sharing an RDMA-based real-world service with the same server. We focus on privacy leakage of the access pattern rather than the data. In VI-A, we snoop database operation workloads in a distributed database. In VI-B, we reconstruct which data the victim accesses. In this model, the attacker and victim can read from a shared memory, like an open library or key-value store. Workload identification and access pattern leakage cause privacy breaches. An attacker can infer individual usage habits and expose system access hotspots in key-value stores. This leakage exacerbates the system's vulnerability. For instance, NVLEAK [40] compromise the privacy of a SQL database or key-value storage by maliciously spying on the victim's queries. Researches on Oblivious RAM indicate that curious attackers can concentrate resources on decrypting only hotspot data, analyze inter-query relations, and eventually break data confidentiality through access pattern snooping [29].

A. Fingerprint Distributed Database with Grain-II Attack

We exploit our reverse-engineering findings to fingerprint and extract operation workloads from the victim.

Specifically, we target an RDMA-based database shuffle/join application [23]. The shuffle/join method is a network-intensive operation and is widely used in distributed databases, especially suitable for

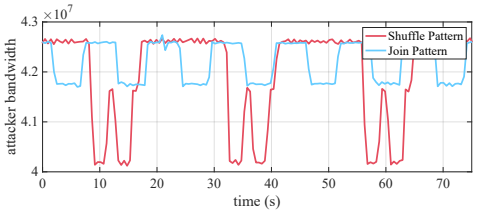


Fig. 12: Fingerprint patterns of shuffle/join.

Algorithm 1 Side-channel attack on SHUFFLE-JOIN

Input: $P_{Shuf/Join}$. Pattern of the Shuffle/Join operation
Output: Detected pattern P

$BW_{History} \leftarrow [\,]$
Establish RDMA connection and initialize monitoring traffic
while RDMA connection is active **do**
  Current time $t \leftarrow GetCurrentTime()$
  Current traffic bandwidth $BW \leftarrow GetTrafficBW()$
  Append (t, BW) to $BW_{History}$
  Maintain Time Window:
  for each entry $(time, _)$ in $BW_{History}$ **do**
    if $time < t - T_{window}$ **then**
      Remove $(time, _)$ from $BW_{History}$
    end if
  end for
  Traffic Pattern Detection:
  P = CorrelationDetect ($BW_{History}$, P_{Shuf}, P_{Join})
  Pattern P detected **if** P != P_{Null}
  Wait for the next monitoring cycle
end while
return P

large datasets. We establish a side channel based on priority-based contention, and perform fingerprinting on each shuffle/join operation.

In this experiment, we monitor the behavior of a small flow maintained by the attacker (as shown in Algorithm 1). As illustrated in Figure 12, the attacker's bandwidth decreases plateau-like during the shuffle and tooth-like during the join. The observed pattern slightly deviates from the baseline under different round times and configurations, demonstrating that this side-channel attack can extract clear information about shuffle/join operations.

B. Snoop on Disaggregated Memory with Grain-IV Attack

Disaggregated memory decouples CPU and memory into independent and network-attached components, computing, and memory servers (CS/MS) [6], [10], [21]. We choose SHERMAN [39], a disaggregated memory setup with write-optimized distributed B$^+$ Tree index, to perform our side-channel attack. SHERMAN is currently implemented as a 64 B KV store. We regard it as a file index in the MS cluster. We set the shared file to be 1 KB in the remote memory, assuming the ratio of numbers of file index access to file access to be 0.01 and the file access size to be 64 B.

In this setting, the attacker and the victim are procedures on CSs to read data from shared memory in an MS. Typically, one client cannot know which address the other client accesses, while the attacker aims to do so by posting certain RDMA Reads to trigger offset effect.

Figure 13 illustrates the attack's three steps. The victim repeatedly accesses an address from the *Candidate Set* (17 candidates, 0 B to 1024 B address offset) using 64 B RDMA Reads. During ❶, the attacker performs 64 B RDMA Reads on each address in the *Observation Set* (257 samples, 0 B to 1024 B address offset) N times

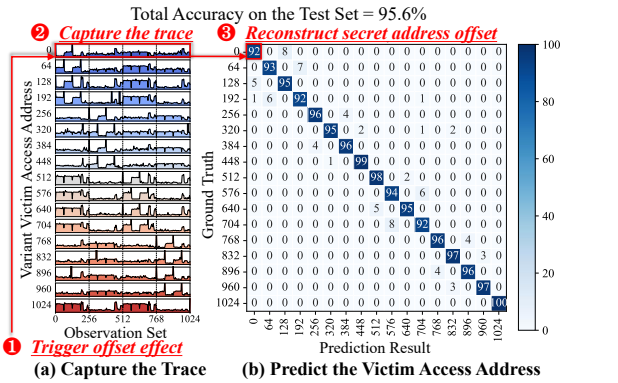


Fig. 13: Evaluation on the RAGNAR side-channel attack. (a) Differentiated traces were captured by the attacker under 17 variant victim access addresses. (b) ResNet18-based 17-Classifier achieves an overall 95.6% accuracy on recovering the address from a trace.

to measure ULI. In ❷, the average ULIs form a trace revealing the victim's access. For instance, if the victim accesses 0 B address offset, the trace will match the pattern in the red box shown in Figure 13(a).

Furthermore, ❸ we involve a ResNet18 classifier to recover the victim's access address from the captured trace. We train the classifier with 6720x 257-dimensional traces. It performs 17-classification on the traces and achieves an overall 95.6% accuracy on the test set, shown in Figure 13 (b). It shows that RAGNAR can easily steal the address accessed by the victim with high accuracy.

VII. DISCUSSION ON POTENTIAL MITIGATION

In this section, we discuss potential defenses.

Existing defenses. As discussed in II-D, the state-of-the-art performance isolation implementation HARMONIC [22] is not sufficient to mitigate RAGNAR's attack, since it does not take Grain-IV metrics into account and cannot eliminate the bandwidth differences completely. To the best of our knowledge, there is no direct existing mitigation to our attacks.

Hardware partitioning or adding noise. Direct mitigation involves fixing hardware features like eliminating priority races and mitigating offset effects by partitioning traffic workloads fairly according to counters from Grain-I-to-IV, which is costly and degrades performance. On the other hand, introducing sub-microsecond noise into packet latency can obscure ULI but may still leave detectable traces. Adding full noise for complete masking results in significant performance degradation.

VIII. CONCLUSION

This paper introduces RAGNAR, a comprehensive suite of RDMA-targeted volatile-channel attacks leveraging RNIC contention-based vulnerabilities at different granularity levels. Covert-channel RAGNAR attacks offer higher bandwidth and stealthiness than existing RDMA-targeted hardware attacks. Side-channel RAGNAR attacks can successfully reconstruct operations and achieve up to 95.6% accuracy in stealing secrets on real-world applications.

ACKNOWLEDGMENT

This work was generously supported by NSFC (U24A6009), National Key Research and Development Program of China under Grant (2024YFB4405402), Beijing Municipal Science and Technology Project (Nos.Z241100004224028), Beijing Natural Science Foundation (L247013), BNRist, the Disruptive Innovation Talent Cultivation Program of Tsinghua University.

References

[1] O. Acıiçmez and Ç. K. Koç, "Trace-driven cache attacks on aes (short paper)," in *Information and Communications Security: 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006. Proceedings 8*. Springer, 2006, pp. 112–121.

[2] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. García, and N. Tuveri, "Port contention for fun and profit," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 870–887.

[3] D. J. Bernstein, "Cache-timing attacks on aes," 2005.

[4] C. Bestler, B. Metzler, J. Garcia, and A. Romanow, "Direct Data Placement over Reliable Transports," RFC 5041, October 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5041

[5] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*. Springer, 2006, pp. 201–215.

[6] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 79–92.

[7] J. Carrier and B. Aboba, "Marker PDU Aligned Framing for TCP Specification," RFC 5044, October 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5044

[8] I. Corporation, "Remote direct memory access over the converged enhanced ethernet fabric: Ibm's perspective on iwarp," *IBM Systems Journal*, vol. 41, no. 4, pp. 726–745, 2002.

[9] M. Dai, R. Paccagnella, M. Gomez-Garcia, J. McCalpin, and M. Yan, "Don't mesh around:{Side-Channel} attacks and mitigations on mesh interconnects," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2857–2874.

[10] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 249–264.

[11] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 490–505.

[12] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance rdma systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 437–450.

[13] H. Kim and J. Hur, "Pcie side-channel attack on i/o device via rdma-enabled network card," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2022, pp. 1468–1470.

[14] M. Ko, J. Hufferd, J. Chu, and S. Shah, "Transport Mappings for Remote Direct Memory Access over IP Fabrics," RFC 5045, October 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5045

[15] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[16] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[17] X. Kong, J. Chen, W. Bai, Y. Xu, M. Elhaddad, S. Raindel, J. Padhye, A. R. Lebeck, and D. Zhuo, "Understanding {RDMA} microarchitecture resources for performance isolation," in *20th USENIX Symposium on Networked Systems Design and Implementation*, 2023, pp. 31–48.

[18] X. Kong, Y. Zhu, H. Zhou, Z. Jiang, J. Ye, C. Guo, and D. Zhuo, "Collie: Finding performance anomalies in {RDMA} subsystems," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 287–305.

[19] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "Netcat: Practical cache attacks from the network," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 20–38.

[20] Y. Li, Z. Wang, B. Han, P. Li, and H. Zhang, "Rdma over commodity ethernet at scale," in *Proceedings of the 2017 ACM SIGCOMM Conference*, 2017, pp. 385–398.

[21] L. Liu, W. Cao, S. Sahin, Q. Zhang, J. Bae, and Y. Wu, "Memory disaggregation: Research problems and opportunities," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1664–1673.

[22] J. Lou, X. Kong, J. Huang, W. Bai, N. S. Kim, and D. Zhuo, "Harmonic: Hardware-assisted {RDMA} performance isolation for public clouds," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1479–1496.

[23] S. Ma, T. Ma, K. Chen, and Y. Wu, "A survey of storage systems in the rdma era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4395–4409, 2022.

[24] Mellanox Technologies, *Mellanox Adapters Programmer's Reference Manual (PRM)*, https://network.nvidia.com/files/doc-2020/ethernet-adapters-programming-manual.pdf, 2020, accessed: 2024-08-02.

[25] NVIDIA Corporation, *Ethtool: A Standard Linux Utility for Network Drivers and Hardware*, NVIDIA, Santa Clara, CA, 2024, Accessed: 2024-11-19. [Online]. Available: https://docs.nvidia.com/networking/display/mlnxofedv461000/ethtool

[26] NVIDIA Corporation, *NVIDIA MLNX_OFED Documentation v24.04-0.6.6.0*, 24th ed., NVIDIA, Santa Clara, California, USA, May 2024, available at https://docs.nvidia.com/networking/display/mlnxofedv24040660.

[27] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring (s): Side channel attacks on the {CPU}{On-Chip} ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 645–662.

[28] C. Percival, "Cache missing for fun and profit," *BSDCan*, vol. 2005, 2005.

[29] B. Pinkas and T. Reinman, "Oblivious ram revisited," in *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer, 2010, pp. 502–519.

[30] J. Pinkerton and E. Deleganes, "DDP/RDMAP Security," RFC 5042, October 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5042

[31] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia, "Remote Direct Memory Access Protocol (RDMAP)," RFC 5040, October 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5040

[32] L. Rosa, L. Foschini, and A. Corradi, "Empowering cloud computing with network acceleration: A survey," *IEEE Communications Surveys & Tutorials*, 2024.

[33] B. Rothenberger, K. Taranov, A. Perrig, and T. Hoefler, "{ReDMArk}: Bypassing {RDMA} security mechanisms," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 4277–4292.

[34] A. K. Simpson, A. Szekeres, J. Nelson, and I. Zhang, "Securing {RDMA} for {High-Performance} datacenter storage systems," in *12th USENIX Workshop on Hot Topics in Cloud Computing*, 2020.

[35] R. Stewart and M. Kojo, "Stream Control Transmission Protocol (SCTP) Direct Data Placement (DDP) Adaptation," RFC 5043, October 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5043

[36] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 213–226.

[37] S.-Y. Tsai, M. Payer, and Y. Zhang, "Pythia: remote oracles for the masses," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 693–710.

[38] H. Wang, H. Sayadi, T. Mohsenin, L. Zhao, A. Sasan, S. Rafatirad, and H. Homayoun, "Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1414–1419.

[39] Q. Wang, Y. Lu, and J. Shu, "Sherman: A write-optimized distributed b+ tree index on disaggregated memory," in *Proceedings of the 2022 international conference on management of data*, 2022, pp. 1033–1048.

[40] Z. Wang, M. Taram, D. Moghimi, S. Swanson, D. Tullsen, and J. Zhao, "{NVLeak}:{Off-Chip}{Side-Channel} attacks via {Non-Volatile} memory systems," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6771–6788.

[41] T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance rdma protocols in hpc," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 13th European PVM/MPI User's Group Meeting Bonn, Germany, September 17-20, 2006 Proceedings 13*. Springer, 2006, pp. 76–85.

[42] W. Xiong and J. Szefer, "Survey of transient execution attacks and their mitigations," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–36, 2021.

[43] Y. Zhang, J. Gu, Y. Lee, M. Chowdhury, and K. G. Shin, "Performance isolation anomalies in rdma," in *Proceedings of the Workshop on Kernel-Bypass Networks*, 2017, pp. 43–48.

Buffer Prospector: Discovering and Exploiting Untapped Buffer Resources in Many-Core DNN Accelerators

Yuchen Wei[†*], Jingwei Cai[†*], Mingyu Gao[†], Sen Peng[¶], Zuotong Wu[¶], Guiming Shi[†], Kaisheng Ma[†§]
[†] Tsinghua Univeristy, [¶] Xi'an Jiaotong University
[*] *Co-first Author,* [§] *Corresponding Author*
{weiyc22, caijw21}@mails.tsinghua.edu.cn, kaisheng@tsinghua.edu.cn

***Abstract*—In large-scale DNN inference accelerators, the many-core architecture has emerged as a predominant design, with layer-pipeline (LP) mapping being a mainstream mapping approach. However, our experimental findings and theoretical justifications uncover a hardware-independent and prevalent flaw in employing layer-pipeline mapping on many-core accelerators: a significant underutilization of buffer space across numerous cores, indicating substantial potential for optimization. Building on this discovery, we develop a universal and efficient buffer allocation strategy, BufferProspector, which includes a Buffer Requirement Calculator and Buffer Allocator, to capitalize on these unused buffers, addressing the timing mismatch challenge inherent in LP mapping. Compared to the state-of-the-art (SOTA) open-source LP mapping framework Tangram, BufferProspector averages a simultaneous increase in energy efficiency and performance by 1.44× and 2.26×, respectively. Moreover, we conduct some case studies on architecture and mapping. *BufferProspector will be open-sourced*.**

***Index Terms*—DNN Accelerators, Many-core, Buffer Management, Scheduling**

I. Introduction

Deep neural networks (DNNs) are widely deployed to solve real-world problems, such as image recognition [1], [2], [3] and natural language processing [4]. To achieve higher accuracy and robustness in complex scenarios, the size and complexity of DNN are becoming larger and larger, which results in increasing compute and storage demands. Therefore, many large-scale accelerators have been proposed to meet these demands [5], [6], [7], [8]. Among them, many-core architecture stands out as a predominant type [5], [6], [7], [8]. Within such accelerators, each core can independently handle a specific computational task. Moreover, the cores are interconnected via a Network-on-chip (NoC), ensuring mutual access.

Despite the rich on-chip computing and storage resources of large-scale many-core accelerators, harnessing their full potential to boost computing performance and energy efficiency remains challenging [9], [10], [11]. This has led to the introduction of LP mapping [8], [10], [11], [12]. In LP mapping, distinct layers are assigned to specific core groups, and their computed output feature maps (ofmaps) can be directly relayed to the consuming layer via the NoC, avoiding the expensive DRAM access costs. Compared to the traditional layer-sequential (LS) mapping [13], [14], where all computing cores are employed to compute layers one by one, LP mapping notably elevates both performance and utilization [11], [8]. For example, simply implementing LP mapping in Simba resulted in a 2.3× throughput increase over LS [8]. Due to its effectiveness, LP mapping is now widely embraced by most contemporary many-core accelerators [5], [6], [7], [8], [12].

Nonetheless, the prevailing LP mapping strategy exhibits a fundamental shortcoming: numerous cores suffer from extremely low buffer utilization rates. For example, when mapping ResNet-50 [1] on the default hardware setting (See Sec. V-A1) by existing SOTA LP mapping strategies [11], the buffer utilization of a half of the cores will be less than 25.64%. We conduct extensive theoretical analyses combined with empirical experiments to discern the direct proportional relationship between the Data-Compute Ratio (DCR) of each layer under LP mapping and the buffer utilization of the core processing it. DCR is defined as the ratio of a layer's data amount to its computation amount. Consequently, the substantial variability in DCR across different layers in modern DNNs contributes to the low utilization of numerous cores.

Upon unearthing this deficiency, we identified a significant amount of unused buffer. The next question to answer is: how to fully utilize such unused buffer resources and improve the performance and energy efficiency of the accelerator? We focus on the **timing mismatch challenge** in LP: Contemporary DNN architectures typically manifest as complex Directed Acyclic Graphs (DAGs) [1], [2], [4], and when two paths from a source layer to a destination layer have different lengths (e.g., in a residue block [1]), the shorter one will produce its ofmaps much earlier than the other. If we sent these early-produced fmaps directly to the consuming layer, there will be severe pipeline stalls. Thus, SOTA scheduling frameworks like Tangram [11] simply forbid such reuse and send it directly to DRAM for future usage, missing significant reuse opportunities. However, with the observation above, we can store such early-produced fmaps into the untapped buffers, which previous frameworks are unaware of, until the latest ones are ready.

Realizing the objectives above in a universal and efficient manner poses significant challenges. Specifically, with an arbitrary complex DNN DAG structure, how do we ascertain the buffer needs for each edge? Furthermore, how do we optimally distribute buffer space to accommodate these requirements? To address these issues, we introduce a comprehensive and efficient buffer allocation strategy, BufferProspector, which encompasses a universal buffer requirement calculator capable of pinpointing all edges that lead to timing discrepancies in any DNN DAG and determining the buffer needs for each edge. Thus, based on these requirements and the remaining capacity of each on-chip buffer, our efficient buffer allocator judiciously allocates unused on-chip buffer space to each demanding edge in a NoC-friendly manner.

In conclusion, we make the following contributions:

(1) We observe a fundamental deficiency in the widely-used LP mapping: a low buffer utilization rate in many cores, and analyze its cause qualitatively and quantitatively.

(2) To fully leverage the discovered unused buffer, we propose a universal and efficient buffer allocation strategy, BufferProspector, to address the time mismatch challenge within LP mapping. BufferProspector is open-sourced at this link [15].

(3) We conduct experiments spanning diverse DNN architectures, accelerator scales, and batch sizes. These experiments conclusively show that BufferProspector enhances both performance and energy efficiency over Tangram [11].

(4) We conduct case studies to explore the impact of buffer size,

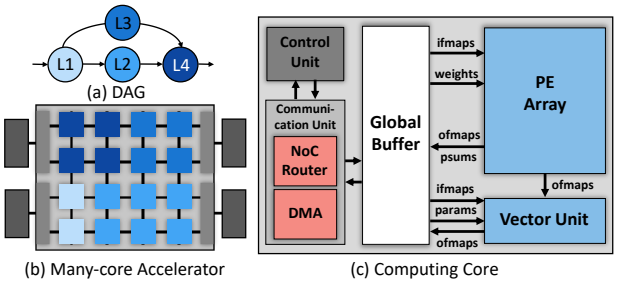


Fig. 1: Mapping Four Layers on A Many-core Accelerator.

a critical architectural factor, as well as the effect of pipeline depth, a key mapping factor, on performance and energy efficiency.

II. Background and Motivation

A. Many-core DNN Inference Accelerators

In recent years, many-core DNN inference accelerators have gained significant popularity in DNN acceleration. To facilitate a deeper investigation into their mapping strategies, we develop a universal and configurable hardware template (see Fig. 1) by extracting and synthesizing common features from a range of existing many-core inference accelerators [11], [5], [8], [10].

Overall, many-core accelerators comprise computing cores responsible for executing specific computing tasks, IO interfaces such as DRAM PHY & controllers (depicted as the light grey blocks on the edge of the chip in Fig. 1(b)), and a NoC, which manages data exchange between cores as well as between cores and DRAM.

The computing core (see Fig. 1(c)) is responsible for executing computing tasks. The following are its critical components. The communication unit, comprised of DMA and a router, enables seamless data exchange with other cores and DRAM. The control unit orchestrates computing tasks, drawing upon statically-compiled instructions and ongoing task progress data, and it also manages the inward and outward flow of data or messages from external cores or DRAMs. Each core is equipped with a Global Buffer (GLB), accessible throughout the accelerator, allowing for data reads and writes to and from the GLBs of other cores, given that the data is valid or the address is writable. The PE array and vector unit compute General Matrix Multiply (GEMM)/Convolution (Conv) operations and handle vector/scalar operations, respectively.

B. Layer-pipeline Mapping

In this section, we delve into the basis of LP Mapping and explore the challenges associated with timing mismatch that it faces.

1) Layer-pipeline Mapping: LP mapping is extensively employed in current large-scale many-core DNN accelerators [5], [8], [7], [6]. Under this mapping paradigm, each DNN graph is initially divided into various subgraphs, and each layer within a subgraph is assigned to a group of cores for computation. For instance, as illustrated in Fig. 1, L1-L4 are allocated 2, 6, 4, and 4 computing cores, respectively. Subsequently, a batch of data is divided into subbatches (SB), and each subbatch is processed in a pipelined manner by the core groups assigned to each layer. For example, once the first subbatch of data computed by L1 is sent to L2, L2 begins processing the first subbatch while L1 starts computing the second subbatch (see Fig. 2). This method can reduce the overhead of filling and draining while saving expensive DRAM access.

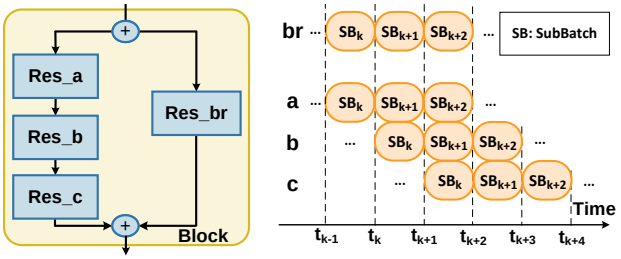


Fig. 2: An Example of Timimg Mismatch in LP Mapping.

2) Timimg Mismatch Challenge: With the development of Deep Learning, the DAG structures of DNN are becoming more and more complex [1], [16], [2], [4]. To balance each pipeline stage and reduce bubbles, the computing time of different Conv layers is kept roughly equal in LP mapping. This is achieved by controlling the number of cores allocated to each layer roughly proportional to the OP-num of that layer. However, when trying to pipeline map multiple branches with different lengths in a segment on the system, we will encounter timing mismatches. For example, as shown in Fig.2, both the shorter and longer branch receive the same input feature maps (ifmaps) at t_{k-1}, but the shorter one outputs data (at t_k) earlier than the longer branch output corresponding data (at t_{k+2}). However, the following element-wise add layer needs to get these ofmaps at the same time to calculate. This mismatch will lead to severe stalls in the pipeline, lowering the utilization of the system. Thus, existing works [11], [17], [18] give up mapping even one whole block containing multiple different branches, which limits the potential of LP mapping and increases expensive DRAM access costs.

III. Key Observation: Low Buffer Utilization with LP Mapping

As introduced in Sec. II-B, LP Mapping is widely employed by most many-core DNN accelerators due to its effectiveness in reducing expensive DRAM access and huge improvement on both performance and energy efficiency. However, in practice, we notice a key deficiency of current LP mapping on many-core accelerators: The buffer utilization of numerous cores is very low. For example, when mapping ResNet-50 [1] on the default hardware setting (See Sec. V-A1) by SOTA LP mapping framework [11], the buffer utilization of a half of the cores is less than 25.64%. Since buffers provide precious on-chip reuse opportunities, low buffer utilization implies that such opportunities are greatly wasted and the potential DRAM access reduction brought by on-chip data reuse in LP mapping is not fully explored.

Next, we aim to establish that this phenomenon is universally prevalent, solely related to the layers' DCR attribute, and independent of the hardware architecture. This proof also signifies the discovery of a substantial amount of underutilized buffer space, presenting numerous opportunities for optimization.

The Data-Compute-Ratio (DCR) of layer i is define as

$$DCR_i = \frac{data_i}{op_i} \tag{1}$$

where $data_i$ is the total data size of layer i, including ifmaps, weights, and ofmaps, and op_i is the OP-num of layer i. DCR depicts the ratio between the data size and computation of a layer. Typically, different layers in the same DNN will have different DCR values, and the variance of these values are relatively high. For example, the largest DCR among all convolution (Conv) layers in ResNet-50 [1] and DenseNet [2] are 11.25× and 6.14× larger than the smallest ones, respectively.

Then we conduct some theoretical analysis with some ideal assumptions to show the mathematical relation between DCR and buffer utilization of each layer. Let BU_i denote the buffer utilization of layer i, num_i denote the number of cores computing layer i, B denote the buffer capacity of each core, then we have

$$BU_i = \frac{data_i}{num_i \cdot B} \tag{2}$$

Furthermore, let C be the computing power of each core (the OP-num a core can compute in one time unit), t_i be the compute time of layer i, then by definition of C we have

$$t_i = \frac{op_i}{num_i \cdot C} \tag{3}$$

Combine equation 1, 2 and 3, we have

$$DCR_i = \frac{data_i}{op_i} = \frac{BU_i \cdot (num_i \cdot B)}{t_i \cdot (num_i \cdot C)} = \frac{B}{C} \cdot \frac{BU_i}{t_i} \tag{4}$$

which is a universal relation between DCR_i and BU_i: The DCR of a layer is proportional to the ratio between buffer utilization and compute time of that layer.

However, there is a fundamental property in LP mapping: To avoid bubbles and maintain high compute utilization, the compute time of each layer in the DNN must be roughly the same in LP mapping [8], [5], [10], [11]. In other words, there exists a constant T such that $t_i \approx T$ for different layers in the pipeline, then equation 4 becomes

$$DCR_i = \frac{B}{C} \cdot \frac{BU_i}{t_i} \approx \frac{B}{C \cdot T} \cdot BU_i = Const \cdot BU_i \tag{5}$$

This shows that the DCR of a layer is roughly proportional to its buffer utilization in LP mapping. Thus a high variance in DCR implies a high variance in the buffer utilization among different layers. Since the largest buffer utilization cannot exceed 100%, layers with low DCR will suffer from much lower buffer utilization.

Figure 3 demonstrates the DCR of four layers within a residual block from ResNet-50 and their corresponding buffer utilization in a practical LP mapping scenario. Among these layers, a relative proportionality is observed between their DCR and buffer utilization: Layer_a (L_a) has the highest DCR, which correlates with the largest buffer utilization among all layers. In contrast, layer L_b's DCR is only $0.089\times$ that of L_a, leading to a significantly lower buffer utilization of just 12.9%.

In summary, the analysis reveals a linear relationship between the DCR, a layer feature, and its buffer utilization. This, coupled with the fundamental requirement in existing layer-pipeline mapping strategies [11], [18], [17] for computational timing to be as identical as possible across different layers, leads to low buffer utilization under current mapping strategies. Thus, this deficiency is independent of hardware architecture and exhibits a universal presence.

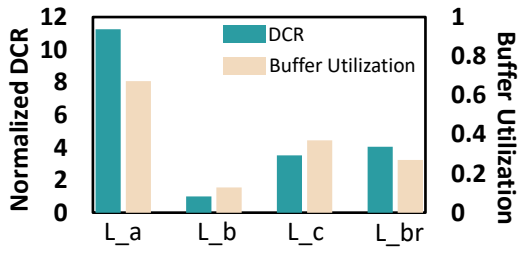


Fig. 3: DCR and Buffer Utilization of Mapping Four Layers in A Residual Block [1]. DCRs of all layers are normalized to the L_b layer with the lowest DCR.

IV. BufferPropector Strategy

After realizing the existence of such unused buffer on-chip, the next question to answer is: How can we turn such precious unused buffer resources into actual performance and energy efficiency gain? We focus on solving the timing mismatch challenge in LP mapping introduced in Sec. II-B2.

The high-level idea to solve the above challenge is straightforward: With the observation in Sec. III, we can store such early-produced fmaps (introduced in Sec. II-B2) into under-utilized buffers, which existing strategies are unaware of, and send buffered fmaps to the destination layer when required. As in the example in Sec. II-B2, the former ofmaps can be stored in unused buffers between t_k and t_{k+2}, and sent to the next layer at t_{k+2}. In this way, the ofmaps can be transmitted to the destination without storing into DRAM, which reduces a large amount of expensive DRAM accesses compared with existing works [11], [17], [18].

However, implementing this high-level concept universally and efficiently encounters two major challenges:

1. In an arbitrary DNN model with complex dependencies, how do we ascertain the buffer requirements for each timing mismatch?
2. With current buffer usage information, how do we efficiently allocate unused buffer space to accommodate these requirements?

Thus, we develop a universal and efficient buffer allocation strategy, BufferProspector, to solve these challenges. BufferProspector consists of a universal Buffer Requirement Calculator and an efficient Buffer Allocator. The Buffer Requirement Calculator detects all edges that lead to timing mismatch in the DNN dependency graph and calculates the buffer needed for each edge, solving the challenge 1. Then this calculated buffer requirement, along with the remaining capacity of each buffer, is sent to the Buffer Allocator, where the Buffer Allocator allocates unused on-chip buffer space to each edge in a NoC-friendly manner, which solves the challenge 2.

A. Buffer Requirement Calculator

Before calculating the buffer requirement of each edge in the DAG, we need to define the depth of each layer in the graph. For an arbitrary DNN with N layers denoted as $L_1, L_2, \ldots, L_N$, let $L_j \to L_i$ denote that there is an edge from L_j to L_i in the DAG. Then denote $P_i = \{L_j \mid L_j \to L_i\}$ be the preceding layers of L_i. Now the depth d_i of a layer L_i is defined as

$$d_i = \begin{cases} 0 & , P_i = \emptyset \\ \max_{L_j \in P_i} [d_j] + 1 & , P_i \neq \emptyset \end{cases} \tag{6}$$

where d_i can be calculated iteratively under a topological order of the DAG.

A layer has depth d means that there is a chain of d layers before that layer in the graph, and there does not exist any chain longer than d. Since when there is a dependency $L_j \to L_i$ in the graph, L_i must be processed at least one subbatch later than L_j (e.g. see Res_a $\to$ Res_b in Fig. 2), a layer of depth d will be processed d subbatches later than the first layer.

Then, for a dependency $L_j \to L_i$, the finishing of a subbatch at layer L_j is $d_i - (d_j + 1)$ subbatches before the starting of a subbatch at layer L_i. Thus if $d_i = d_j + 1$, then there is no need to buffer; if $d_i > (d_j + 1)$, a subbatch should be buffered during the processing of $d_i - (d_j + 1)$ subbatches, thus at most $d_i - (d_j + 1)$ subbatches will be buffered at the same time. However, to hide the latency of NoC transmission, each subbatch should be produced in a stream-processing manner and as long as some ofmaps are produced, NoC can start transmitting them. Similar to double buffering, such overlapping of NoC and processing requires the buffer capacity of

one additional subbatch, yielding a capacity of $d_i - d_j$ subbatches in total. Furthermore, notice that when the total number of subbatches NSB is smaller than $d_i - d_j$, then there is only NSB subbatches to be buffered in total. So if we denote c_j as the size of ofmaps of one subbatch in L_j, the capacity needed for $L_j \to L_i$ is

$$Cap_{j,i} = \min\left[d_i - d_j, NSB\right] \cdot c_j \tag{7}$$

Finally, when there are multiple edges starting from L_j that are needed to buffer, we only need to buffer the edge with the longest time interval. Since all buffering starts at the same time, the buffer needed for the ofmaps of L_j is:

$$Cap_j = \max_{\substack{L_j \to L_i \\ d_i > d_j + 1}} [Cap_{j,i}] = \min\left[\max_{\substack{L_j \to L_i \\ d_i > d_j + 1}} [d_i] - d_j, NSB\right] \cdot c_j \tag{8}$$

and if there are no such $L_j \to L_i$ with $d_i > d_j + 1$, let $Cap_j = 0$ and there is no need to buffer.

Taking Fig. 2 as an example: If the depth of the top Add layer is 0, then the depth of the bottom Add layer and Res_br is $d_{add} = 4$ and $d_{br} = 1$, respectively. Since SB_k is produced by Res_br before t_k and produced by the bottom add layer after t_{k+2}, it needed to be buffered between t_k and t_{k+2}. Then considering the transmission of NoC, we also need to allocate its buffer between t_{k-1} and t_k to receive the partial result from Res_br. Similarly, SB_{k+i} needs an allocation of buffer between t_{k-1+i} and t_{k+2+i}. In this way, the buffer needs to buffer SB_k, SB_{k+1}, SB_{k+2} at once, resulting with a capacity of 3 subbatches, which matches with $d_{add} - d_{br}$.

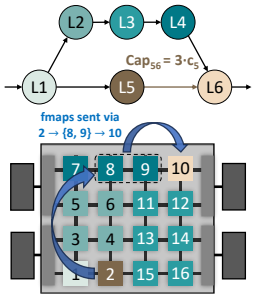
In summary, the Buffer Requirement Calculator firstly finds a topological order of the layers in the DNN, then calculate $d_1, d_2, \ldots, d_n$ using equation 6, at last outputs the buffer requirement of each layer by equation 8.

B. Buffer Allocator

After the buffer requirement of each edge is obtained, we need to allocate unused on-chip buffer space to satisfy these requirements. During the allocation, there are three factors to consider:

Number of Cores: Since the buffer requirement size usually contains multiple subbatches, we need to select multiple cores to buffer early-produced fmaps in many cases. However, allocating too many cores for a single requirement can lead to considerable concurrent synchronization overheads when the subsequent layer needs to fetch these fmaps.

Total NoC Distance: The NoC transmission of these fmaps will not only produce additional energy costs, but also occupy limited on-chip NoC bandwidth. Both factors are proportional to the total NoC distance from the producer and consumer layers to the buffer, so reducing the total NoC distance can improve performance and energy efficiency.



Buffer Allocation Example (3·c_5 =300 KB, L = 50 KB)

Core	Remaining Capa.	Distance	Selected
1	200 KB	7	
2	40 KB	5	
3, 5	10 KB	7	
4, 6	10 KB	5	
7	150 KB	7	
8, 9	150 KB	5	√
10	0 KB	5	
11~16	30 KB	5	

Fig. 4: An Example of Mapping Six Layers on A 16-core Accelerator. In the table "Capa." stands for Capacity and "Distance" represents the Total NoC Distance. Cores are labeled from 1 to 16, in which core 8 and 9 are selected to buffer the 300KB requirement of $L5 \to L6$. L is set to 50KB.

Algorithm 1: Buffer Allocator

Input: List of Requirements req_list, Remain Space of Each Buffer $rems$, Size Limit of Each Allocation L
Output: Buffer Allocated to Each Requirement $alloc$

```
// Iteratively allocate each requirement req
for each req in req_list do
    // Filter all buffers with at least L remaining capacity
    avail_bufs = [x for x in rems if rems[x] ≥ L]
    // Then sort with total NoC distance and allocate in this order
    sort_key = lambda i: dist(req.src, i) + dist(i, req.dst)
    avail_bufs.sort(key = sort_key)
    remain_req = req.size
    for each buf in avail_bufs do
        // Allocate all remaining capacity of buf to req
        alloc_size = min(remain_req, rems[buf])
        remain_req -= alloc_size
        rems[buf] -= alloc_size
        alloc[req][buf] = alloc_size
        if remain_req == 0 then
            break
        end
    end
    if remain_req > 0 then
        // All avail_bufs has used up, must use DDR
        alloc[req][DDR] = remain_req
    end
end
return alloc
```

Allocated Size of Each Buffer: If the allocated size on a buffer is too small (e.g., less than 1KB), the buffer will suffer fragmentation issues, and transmitting such a small amount of data is not NoC-friendly either.

Considering all three factors above, the Buffer Allocator is implemented with a greedy algorithm (Alg. 1): For each buffer requirement, we set a limit L, which is a configurable parameter, for the smallest allocated size on each buffer, and sort all cores with enough remaining buffer capacity by their total NoC distance to the producer and consumer layers. Then we allocate from the nearest core to the furthest, allocating all remaining capacity of its buffer, until the requirement is satisfied. If all such buffers cannot satisfy the requirement, DRAM is used to buffer the unsatisfied part. In our experiments, L is set to one-sixth of the total required size based on our experiment observations. In this way, the number of cores to buffer will not exceed 6, and the allocated size of each buffer will not be too small; on top of that, the Buffer Allocator minimizes total NoC distance, taking great care of all three factors.

Fig. 4 shows an example of buffer allocation: The edge $L_5 \to L_6$ has a buffer requirement of 3 subbatches, which is 300KB. L is set to 50KB, and core 1,3,4,5,6 have limited unused buffer capacity that does not exceed L. Among all the cores that has enough capacity, core 8 and 9 has minimal total NoC distance, thus is prioritized to satisfy the requirement. Although all cores of L_3 have minimal total NoC distance, their remaining capacity is too small (30KB) and we do not utilize such core in consideration of fragmentation.

C. BufferProspector Implementation

We have developed an end-to-end compiler framework based on BufferProspector for our test chip. Except for BufferProspector Scheduling Part, the compiler includes a model parser that processes models from mainstream frameworks like PyTorch or TensorFlow

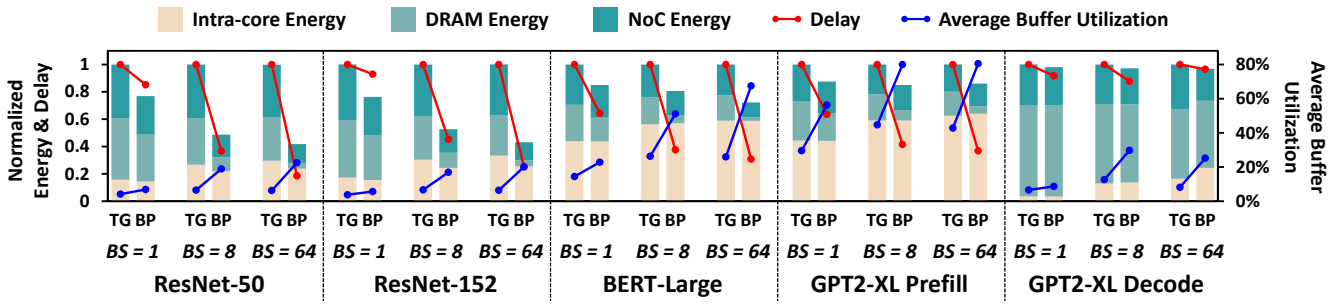


Fig. 5: Overall Comparisons Between Tangram (TG) and BufferProspector (BP). $BS = N$ is short for "batch size is N". Energy and Delay are normalized to Tangram in each pair of columns, while Average Buffer Utilization are shown in absolute values.

into the format required by our system. It also features an Intermediate Representation (IR) generator that organizes the information from the explored mapping scheme to create an IR conducive to instruction generation. Finally, an instruction generator transforms the IR into deployable hardware instructions.

V. Evaluation

A. Experiment Setup

1) Hardware Configuration: The default hardware architecture consists of 12×10 cores (this scale of core count is referenced from the well-known Tenstorrent Grayskull [5]), with each core equipped with a 1MB GBUF and 1024 MACs. The total DRAM bandwidth is set at 0.5GB/s per TOPs (120GB/s for the 120-core default accelerator). We also conduct sensitivity tests on accelerators with different scales. To systematically quantify the energy consumption for distinct operations required by the Tangram energy evaluators, such as MAC and buffer accesses, we employ a diverse array of tools and data. This includes directly obtaining data from commercial IP datasheets, or simulate RTL codes through a simulation flow developed during the chip design.

2) Workloads: In the overall comparison, we select ResNet-50 [1], ResNet-152 [1], BERT-Large [19], and GPT2-XL [20] as workloads due to their popularity in image and language applications and diverse structures; batch size is scaled among 1, 8 and 64, targeting both throughput-driven and latency-driven scenarios and demonstrating the efficiency of BufferProspector across different batches.

For the Design Space Exploration (DSE) experiments, we select BERT-Large as the default workloads regarding to its wide application; to include both cloud-based, throughput-driven scenarios and edge-based, latency-driven scenarios, batch size of 1 and 64 are selected; we also scale buffer size and DRAM bandwidth to study their impact on the effect of BufferProspector.

3) Baseline: We choose the open-sourced SOTA Tangram scheduling framework [11] as our baseline, since it employs optimized pipeline searching algorithms and mainstream intra-core tiling methods. The hardware configurations of Tangram and BufferProspector are kept the same throughout all experiments. Energy-delay Product (EDP) is selected as the optimization goal in all experiments, which matches the original Tangram paper [11].

B. Overall Comparison

As shown in Fig. 5, compared to Tangram, BufferProspector has demonstrated its value by achieving an improvement of 2.26× performance and 1.44× energy efficiency, on average across the five networks. A notable average reduction of 47.3% in DRAM energy consumption is observed, serving as the primary driver behind the observed enhancements in both performance and energy efficiency. This reduction can be primarily attributed to an increased buffer utilization, averaging 2.26 times, facilitating greater on-chip data reuse. The augmentation in buffer utilization can be attributed to two main advantages brought about by the BufferProspector strategy: (1) It relaxes the constraints on pipeline length present in Tangram (as introduced in Sec. II-B2, Tangram forfeits the mapping of layers that span more than one block due to timing mismatch challenges), allowing more layers to be computed on-chip concurrently, and enabling more feature maps to be directly generated and consumed on-chip without the need for DRAM access. (2) Feature maps that cannot be immediately consumed are temporarily stored on-chip, which avoids the need to write to DRAM and prevents stalling the overall pipeline.

1) Comparison among Networks: BufferProspector exhibits a significant performance improvement in all 6 workloads, which demonstrates the universal improvement of BufferProspector under different network topologies. The only exception here is the decode of GPT2-XL: GPT2-XL Decode has low compute density [21], and the dependency data for on-chip reuse is significantly smaller compared to the weight and KV cache data, leaving limited potential for BufferProspector to optimize.

2) Comparison among Batch Sizes: BufferProspector consistently achieves improvements over Tangram across various batch sizes. Even in latency-sensitive scenarios, where batch sizes can be as small as one, BufferProspector improves 1.29× performance and 1.19× energy efficiency compared to Tangram. The benefits of BufferProspector become increasingly pronounced with larger batch sizes, yielding 3.27× performance improvement and 1.66× energy efficiency improvement at batch size 64. This enhancement primarily attributes to the mitigation of filling and draining overheads in larger batch sizes. Since overheads brought by filling and draining is only relatively proportional to pipeline depth and remains the same in different batch sizes, lengthy pipelines will suffer less from such overheads in larger batch sizes. As a result, longer pipelines can be more efficiently mapped onto the accelerator, maximizing the advantages of BufferProspector and concurrently highlighting the constraints Tangram imposes on pipeline depth. As evidence, we observe that with the increase in batch size, BufferProspector reduces more DRAM access and enhances better buffer utilization. This indicates the ability of BufferProspector to buffer early-produced fmaps and utilize longer pipeline, thereby reducing DRAM access.

C. Design Space Exploration

To further investigate the architecture and layer-pipeline mapping of many-core accelerators, we conduct a DSE, providing in-depth analysis and discussion on the two critical factors of mapping and architecture: pipeline depth and buffer size.

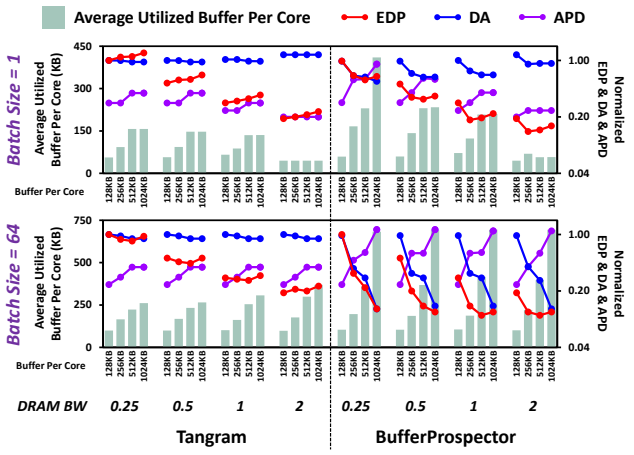


Fig. 6: DSE Experiments. The workload is Bert_Large. **DRAM BW** stands for DRAM bandwidth and 0.25 means "a DRAM bandwidth of 0.25GB/s per core", **DA** stands for DRAM Access, and **APD** stands for Average Pipeline Depth. EDP and DA in each plot is normalized to the first data point of Tangram, i.e. (128KB, 0.25GB/s), and APD is divided by 10 to match the scale. Normalized EDP, DA and APD is drawn in log-scale for better comparison.

1) Will A Deeper Pipeline Be Better? To analyze the influence of pipeline depth on overall performance, it is essential to evaluate the trade-offs associated with a deeper pipeline. On one hand, a deeper pipeline reduces the number of fmaps that need to be saved to DRAM, thereby lowering DRAM access and saving both energy and latency. On the other hand, it increases filling and draining overhead due to dependencies, as well as bubble costs caused by computational imbalances among different layers, which decreases computational utilization and negatively impacts performance. Therefore, determining whether a deeper pipeline is advantageous hinges on assessing whether the benefits outweigh the drawbacks, a trade-off that varies depending on the specific scenario.

When batch size is large, filling & draining overheads are relatively small, and its disadvantage is covered by the benefits of DRAM access reduction. Thus extending the pipeline is beneficial in most times, and both Tangram and BufferProspector will try their best to extend the pipeline depth, hoping for more DRAM access reductions. However, in a deeper pipeline, more layers are mapped onto the cores at the same time, posing a much larger buffer requirement. Thus, for BufferProspector, it will try to make full use of the buffer to pursue a deeper pipeline. This can be evidenced by the huge increase in APD and utilized buffer in Fig. 6 batch size 64, where the reduction of DRAM Access matches the increase of APD. Although Tangram will also try to extend its pipeline, it suffers from timing mismatch issues (introduced in Sec. II-B2) and has a much smaller APD limit, so APD will remain the same after hitting this limit, even with larger buffer, which greatly restricts its performance improvements. This can be seen clearly in Fig. 6 batch size 64.

When batch size is small, the pipeline will experience a much larger filling & draining overheads, thus compared with large batch sizes, extending the pipeline becomes less beneficial and its downside becomes comparable with the improvement of DRAM access reductions. As a result, pipeline tends to be longer when DRAM bandwidth is less sufficient (see Fig. 6 batch 1). Under a smaller DRAM bandwidth, the whole computation becomes more bounded by DRAM accesses, which highlights the contribution of DRAM access reductions to the reduction of overall latency, making longer pipelines more beneficial, and encouraging both Tangram and BufferProspector to extend to a larger pipeline, as shown in Fig. 6 batch 1.

In summary, in large-batch-size scenarios, BufferProspector will try its best to extend its pipeline, and pipeline depth is mostly constrained by the buffer size; while in small-batch-size scenarios, DRAM bandwidth will also be a decisive factor, and a smaller DRAM bandwidth prefers a longer pipeline. **Thus for BufferProspector, one should focus on designing larger buffers in throughput-driven scenarios and focus**

2) Will A Larger Buffer Be Better? When buffer size per core increases from 128KB to 1024KB, BufferProspector on average can reduce EDP by 47.4%. Such huge improvement mainly stems from the possibility of buffering more fmaps on-chip and reducing more DRAM access, as evidenced by a 5.21x average increment of utilized buffer and a 60.5% average reduction of DRAM access.

However, we can observe that such reduction of EDP becomes smaller when buffer size is larger, and for the 1GB/s and 2GB/s settings, the EDP forms a U-shaped curve, even increasing when buffer per core increases from 512KB to 1024KB. There are two main reasons that lead to the smaller reduction, or even increase, of EDP. Firstly, the pipeline cannot extend infinity, and fmaps that needs to be buffered is also limited, thus when the buffer is larger, fmaps that needs to be optimized to on-chip buffer is also fewer, which means expanding it brings less benefit on the EDP. In addition, accessing a larger buffer needs a larger energy cost. For example, it is shown that when the size of a buffer doubles, its access energy cost becomes 1.5 times larger [22]. Thus, an increased buffer size results in a larger energy cost, leading to an increased EDP.

In conclusion, on the one hand, larger buffer usually means better performance; but on the other hand, when buffer becomes larger, the reduction of DRAM access becomes less prominent, while the increase of energy cost becomes more dominant, giving out less performance optimizations, and when the extra energy cost outruns the reduced access, we will even get worse performance. **Thus to pursue the best performance of BufferProspector, one should not aggressively increase the buffer size, but find and follow the appropriate buffer size in accelerator designs.**

Notice that Tangram can barely reduce DRAM access with a larger buffer, thus suffering from an average increase of 14.5% in EDP when buffer size per core increases from 128KB to 1024KB.

VI. Conclusion

This paper begins by highlighting a deficiency commonly observed in existing LP mapping strategies: severe under-utilization of many cores' buffers. We then conduct qualitative and quantitative analyses of this phenomenon, deducing that under current LP mapping strategies, a linear relationship exists between a layer's DCR attribute and buffer utilization. This establishes the phenomenon's hardware independence and widespread prevalence at a theoretical level. Following this, we develop the BufferProspector strategy, which includes a Buffer Requirement Calculator applicable to any DAG and an efficient Buffer Allocator. These tools aim to leverage the substantial amounts of underused on-chip buffer resources discovered by us to address the timing mismatch challenges in layer-pipeline mapping. Finally, our empirical experiments demonstrate that BufferProspector offers significant performance and energy efficiency improvements over the SOTA open-source LP mapping framework, Tangram.

VII. Acknowledgment

This work is supported by Beijing Municipal Science and Technology Program under Grant No.Z241100004224028.

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR 2016*, pages 770–778. IEEE Computer Society, 2016.

[2] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR 2017*, pages 2261–2269. IEEE Computer Society, 2017.

[3] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In Satinder P. Singh and Shaul Markovitch, editors, *AAAI 2017*, pages 4278–4284. AAAI Press, 2017.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS 2017*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[5] Jasmina Vasiljevic, Ljubisa Bajic, Davor Capalija, Stanislav Sokorac, Dragoljub Ignjatovic, Lejla Bajic, Milos Trajkovic, Ivan Hamer, Ivan Matosevic, Aleksandar Cejkov, Utku Aydonat, Tony Zhou, Syed Zohaib Gilani, Armond Paiva, Joseph Chu, Djordje Maksimovic, Stephen Alexander Chin, Zahi Moudallal, Akhmed Rakhmati, Sean Nijjar, Almeet Bhullar, Boris Drazic, Charles Lee, James Sun, Kei-Ming Kwong, James Connolly, Miles Dooley, Hassan Farooq, Joy Yu Ting Chen, Matthew Walker, Keivan Dabiri, Kyle Mabee, Rakesh Shaji Lal, Namal Rajatheva, Renjith Retnamma, Shripad Karodi, Daniel Rosen, Emilio Munoz, Andrew Lewycky, Aleksandar Knezevic, Raymond Kim, Allan Rui, Alexander Drouillard, and David Thompson. Compute substrate for software 2.0. *IEEE Micro*, 41(2):50–55, 2021.

[6] Sean Lie. Multi-million core, multi-wafer AI cluster. In *HCS 2021*, pages 1–41. IEEE, 2021.

[7] Emil Talpes, Douglas Williams, and Debjit Das Sarma. DOJO: the microarchitecture of tesla's exa-scale computer. In *HCS 2022*, pages 1–28. IEEE, 2022.

[8] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *MICRO 2019*, MICRO '52, page 14–27, New York, NY, USA, 2019. ACM.

[9] Jingwei Cai, Xuan Wang, Mingyu Gao, Sen Peng, Zijian Zhu, Yuchen Wei, Zuotong Wu, and Kaisheng Ma. Soma: Identifying, exploring, and understanding the dram communication scheduling space for dnn accelerators. *arXiv preprint arXiv:2501.12634*, 2025.

[10] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. Inter-layer scheduling space definition and exploration for tiled accelerators. In Yan Solihin and Mark A. Heinrich, editors, *ISCA 2023*, pages 13:1–13:17. ACM, 2023.

[11] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. TANGRAM: optimized coarse-grained dataflow for scalable NN accelerators. In Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck, editors, *ASPLOS 2019*, pages 807–820. ACM, 2019.

[12] Jingwei Cai, Zuotong Wu, Sen Peng, Yuchen Wei, Zhanhong Tan, Guiming Shi, Mingyu Gao, and Kaisheng Ma. Gemini: Mapping and architecture co-exploration for large-scale dnn chiplet accelerators. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 156–171. IEEE, 2024.

[13] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel S. Emer. Timeloop: A systematic approach to DNN accelerator evaluation. In *ISPASS 2019*, pages 304–315. IEEE, 2019.

[14] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach. In *MICRO 2019*, pages 754–768. ACM, 2019.

[15] Open-sourced framework of bufferprospector. https://github.com/SET-Scheduling-Project/BufferProspector-DAC2025.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR 2015*, pages 1–9. IEEE Computer Society, 2015.

[17] Xinhan Lin, Shouyi Yin, Fengbin Tu, Leibo Liu, Xiangyu Li, and Shaojun Wei. LCP: a layer clusters paralleling mapping method for accelerating inception and residual networks on FPGA. In *DAC 2018*, pages 16:1–16:6. ACM, 2018.

[18] Xingbin Wang, Boyan Zhao, Rui Hou, Amro Awad, Zhihong Tian, and Dan Meng. Nasguard: A novel accelerator architecture for robust neural architecture search (NAS) networks. In *ISCA 2021*, pages 776–789. IEEE, 2021.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[21] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

[22] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. Interstellar: Using halide's scheduling language to analyze DNN accelerators. In James R. Larus, Luis Ceze, and Karin Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 369–383. ACM, 2020.

AutoRE: Bayesian-Optimization-based Automatic Reliability Enhancement Tool for Flow-based Microfluidic Biochips

Siyuan Liang[1], Yushen Zhang[2], Mengchu Li[2], Tsun-Ming Tseng[2], Ulf Schlichtmann[2], Tsung-Yi Ho[1]
[1] The Chinese University of Hong Kong, [2] Technical University of Munich

***Abstract*—As an emerging platform for biochemical experiments, flow-based microfluidic biochips are currently suffering from malfunctions caused by manufacturing defects, thereby having low yield. While many related studies have been conducted and reliability quantification models have been published, layout optimization methods are yet lacking. In this paper, we propose AutoRE, the first tool to automatically enhance reliability by optimizing layouts. AutoRE varies the layout within a certain range without changing its topology, and adopts Bayesian optimization (BO) to identify the most reliable variant. Experimental results demonstrate that AutoRE can efficiently and effectively improve the reliability across all testcases by around 40% on average.**

I. Introduction

Recently, flow-based microfluidic biochips have emerged as an increasingly attractive platform for biochemical experiments [1]–[3]. Such coin-sized chips incorporate many miniaturized devices to carry out complex experiments. Compared to traditional cumbersome laboratory instruments, these chips enjoy many advantages including low reagents consumption, high automation degree, and high-sensitivity [3]. Such a chip consists of two layers: a flow layer containing flow channels and a control layer containing control channels. Reagents are transported in the flow channels, and the pressure in the control channels controls the transportation [4].

Due to the limitations of material and manufacturing processes, manufacturing defects can happen to these chips and cause malfunctions, resulting in low yield [5]. As the demand for more complex experiments increases, the integration level of these chips becomes higher, introducing significantly denser and longer channels. Consequently, this increases the risk of manufacturing defects, making the reliability of these chips an even more severe concern [6]–[8].

To improve the reliability of these chips, many related works have been conducted. Apart from some works aiming to achieve breakthroughs in material reliability and manufacturing processes, most previous works focus on enhancing the layouts of these chips. Zhu et al. proposed to construct a logic forest, enabling certain valves and channel segments to share the same functionalities, realizing fault tolerance [9]. Huang et al. proposed to construct backup channels in original layouts, providing each device-device connection with multiple possible paths [10]. Moradi et al. proposed to expand the routing fabric, guaranteeing the existence of multiple paths through different transposers [11]. Liang et al. proposed a more efficient coding strategy, and proposed to merge certain channel segments to reduce the density and length of channels in multiplexers [12].

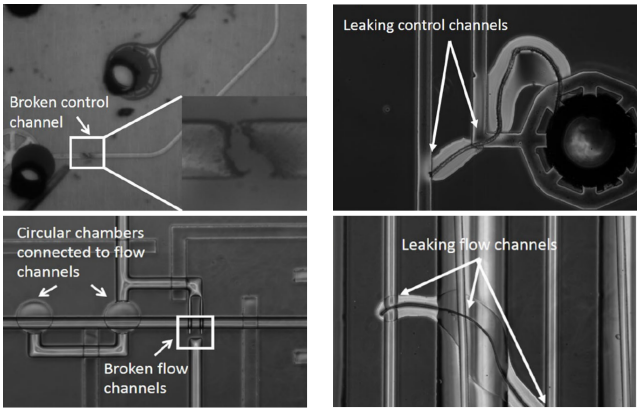
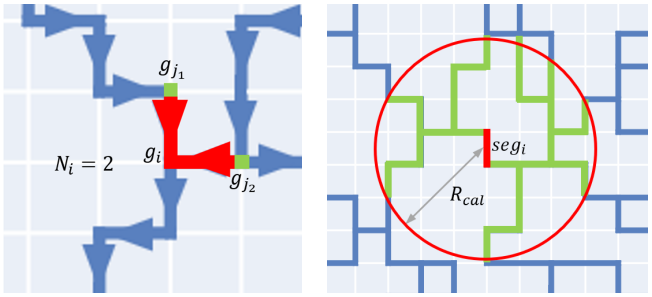


Fig. 1: Photos of common manufacturing defects [5]. (a) Blockage. (b) Leakage.

These studies mainly focus on modifying the layouts to improve the reliability. Unfortunately, all of them lack evaluation methods of the reliability, making it impossible to quantify the degree of the reliability improvement. To tackle this problem, Liang et al. proposed reliability quantification models of two main types of manufacturing defects [13]. However, there is still a lack of layout optimization methods that utilize quantification models to optimize layouts for higher reliability.

In this paper, we propose AutoRE, the first automatic reliability enhancement tool for flow-based microfluidic biochips. Specifically, we first segment the original layout, and alter the segments within a certain range while maintaining the same topology. This way, we identify a feasible domain, in which we can explore among all feasible layout variants for higher reliability. We adopt Bayesian optimization (BO) to improve the efficiency of the exploration, and utilize reliability quantification models proposed in [13] to evaluate the reliability during the process. According to experimental results, AutoRE is an efficient tool that can effectively improve the reliability across all testcases by around 40% on average.

The rest of the paper is organized as follows: Section II introduces the background knowledge. Section III presents the overall flow of the proposed method. Section IV illustrates the identification of the feasible domain. Section V demonstrates the Bayesian-optimization-based reliability enhancement. Sec-



(a) (b)

Fig. 2: Schematic of defect probability quantification models [13]. (a) Blockage. (b) Leakage.

tion VI shows the experimental results, and Section VII draws the conclusion.

II. Background

A. Reliability Quantification Models

There are two main types of manufacturing defects: blockage and leakage [5]. The corresponding quantification models of their probabilities were proposed in [13].

1) Anti-Blockage Reliability

Blockage refers to the accidentally broken channel segments. Examples of broken control and flow channels are shown in Fig. 1(a). For control channels, blockage prevents pressure transportation, leading to losing control of valves; for flow channels, blockage prevents reagents from reaching reactors. Blockage is usually caused by environmental particles and imperfect silicon wafer molds.

According to previous studies [5], [6], blockage occurs in different parts of the layout almost randomly. Based on this, the blockage probability model proposed in [13] assumes every unit channel segment in the grid-based layout to have the same probability of being blocked. When a segment fails, the probability that its backup segments can take over the fluid transportation is the most important factor in determining the anti-blockage reliability. Hence, this model uses a propagation-based strategy to calculate $P_{blockage}$, the probability of layout failure due to blockage.

As shown in Fig. 2(a), the model first derives the directions of fluid transportation using the equivalent circuit model [14]. Following the derived directions, the model progressively calculates the probability of successful fluid transportation to different parts of the layout until it reaches all the ending points. For each grid point g_i, there are N_i adjacent upstream grid points g_j, providing N_i possible paths for fluid transportation. The probability of successful fluid transportation to g_i is determined by considering the likelihood that at least one of the N_i paths successfully transports fluid from g_j to g_i. The layout is considered well-functioning only if the fluid successfully reaches all the ending points. Therefore, $P_{blockage}$ is ultimately derived from the probability of fluid failing to reach all the ending points, and the anti-blockage reliability is then determined by $1 - P_{blockage}$.

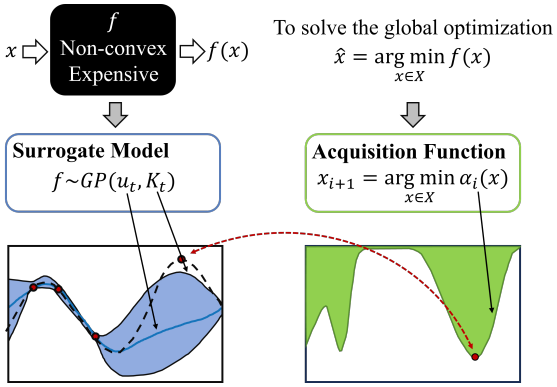


Fig. 3: Schematic of Bayesian optimization.

2) Anti-Leakage Reliability

Leakage refers to the unexpected connections between channels. Examples of leaking control and flow channels are shown in Fig. 1(b). For flow channels, leakage can make reagents unexpectedly infiltrate into other channels, resulting in catastrophic cross-contamination; for control channels, leakage transports pressure from one channel to the other, leading to unanticipated closure of valves. Leakage is usually caused by fiber pollutant and cracks in the layer.

According to previous studies [5], [6], the probability of leakage increases as the length of parallel channels increases or the distance between neighboring channels decreases. Based on this, the leakage probability model proposed in [13] assumes that each unit channel segment in the grid-based layout has the same probability to leak to another unit channel segment at a unit distance. Additionally, it assumes that leakage forms incrementally, unit distance by unit distance. In other words, the probability of leakage occurring between two unit channel segments decreases exponentially as the distance between them increases. This model combines leaking probabilities of each unit channel segment to derive $P_{leakage}$, the probability of layout failure due to leakage.

As shown in Fig. 2(b), for each unit channel segment seg_i, the model scans its nearby region within the range R_{cal}, and identifies all the nearby unit channel segments. The leaking probability of seg_i can then be calculated based on the probability of it leaking to any nearby unit channel segments. Since leakage may grow when high pressure is injected into the channels [5], the layout is considered well-functioning only if there is no leakage at all. Therefore, $P_{leakage}$ is ultimately derived from the probability of leakage occurring anywhere in the layout, and the anti-leakage reliability is then determined by $1 - P_{leakage}$.

B. Bayesian Optimization

As an efficient technique for black-box optimization method, Bayesian optimization (BO) seeks to quickly identify an input $\hat{x}$ that minimizes a function $f(x)$ within a domain $\boldsymbol{X}$, which is defined by finite lower and upper bounds for each variable.

$$\hat{x} = \underset{x \in \boldsymbol{X}}{\operatorname{argmin}} f(x) \tag{1}$$

The objective function $f(x)$ often lacks an analytical expression or derivative, limiting its evaluation to sampling at specific points x and obtaining potentially noisy responses. The feasible domain $\boldsymbol{X}$ is typically either a hyper-rectangle $\left\{x_i \in \mathbb{R}^d : a_i \leq x_i \leq b_i\right\}$ or a d-dimensional simplex $\left\{x_i \in \mathbb{R}^d : \sum_i x_i = 1\right\}$.

As shown in Fig. 3, BO relies on two critical components: (i) the surrogate model, which approximates the objective function; (ii) the acquisition function, which directs sampling to areas where an improvement over the current best observation f' is promising. The surrogate model is typically a Gaussian process (GP) [15], and many alternatives have also been been proposed for better performance, such as random forest (RF), Bayesian additive regression trees (BART), and Bayesian multivariate adaptive regression splines (BMARS) [16]–[18]. A popular acquisition function is expected improvement (EI) [19], defined as the expected value of the utility function $u(x) = \max(0, f' - f(x))$ under the GP predictive distribution for f. Other acquisition functions like log expected improvement (logEI), probability of improvement (PI), and lower confidence bound (LCB) are also widely used in different works [19]–[21]. BO iterates between fitting a model and gathering additional data. Initially, the surrogate model $p(y|x, \mathcal{D}_t)$ is trained on a small set D_t of t samples from the domain $\boldsymbol{X}$. Subsequently, new points $\{x_{t+1}, x_{t+2}, ..., x_T\}$ within the domain $\boldsymbol{X}$ are sequentially selected by optimizing an acquisition function S, which uses the current model $p(y|x, D_i)$ as a cost-effective surrogate for the expensive objective f. This guided point selection, driven by the acquisition function, enables BO to consistently identify superior points, significantly enhancing efficiency compared to random selection. Each observed point x_{i+1} is evaluated by the function f, producing an observed result $f(x_{i+1})$. The result $(x_{i+1}, f(x_{i+1}))$ is added to the historical set D_i, forming the new set D_{i+1}, which updates the surrogate model $p(y|x, D_{i+1})$ for the next observation point. Ultimately, the point $\hat{x}$ can be selected from D_T, ensuring that the objective function f approaches or reaches the global optimum at $\hat{x}$.

III. Overall Flow of the Proposed Method

The overall flow of AutoRE is shown in Fig. 4. The flow consists of two main stages: feasible domain identification, and BO-based reliability enhancement. We vary the given layout within the feasible domain to explore more reliable layout variants. In each round, we will vary the layout in only one of the x-axis and y-axis directions. The optimized layout from each round will serve as the initial solution for the next round, with the direction of variation being switched. This way, the relative locations of different parts will keep unchanged in each round, so that we can maintain the topology of the layout, preserving the functionality of the layout. Additionally, this approach simplifies the problem by reducing it from exploring a 2-D space to exploring multiple 1-D spaces, making the process more manageable and efficient.

Specifically, for a given layout, we first segment it into horizontal and vertical channel segments. In each round, only

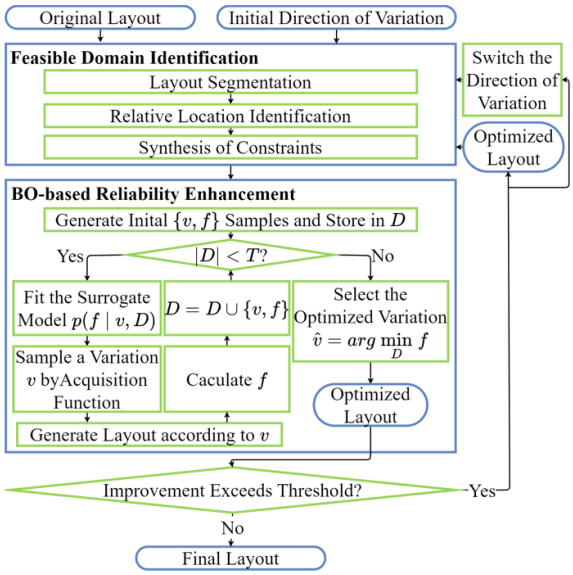


Fig. 4: The proposed overall flow of AutoRE.

the locations of the segments perpendicular to the current variation direction will be actively altered to shift along this direction, while the other segments will be extended or shortened to maintain the topology based on their connection relationships. Next, we develop an algorithm to identify the relative locations of the segments perpendicular to the current variation direction, and synthesize constraints accordingly, forming the feasible domain for the next stage. For example, when the variation direction is along the y-axis, only the y-axis locations of the horizontal channel segments will be actively altered during this round. For each horizontal channel segment, we will identify its nearest upper and lower neighbors, and only allow its y-axis location to be altered between these neighbors. This ensures that the relative locations of segments remain consistent during the variation.

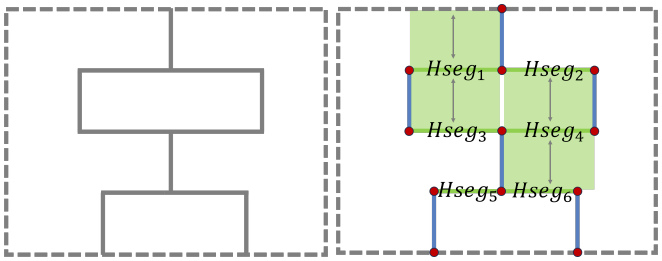
Within the feasible domain, there can be thousands of possible layout variants, making it impractical to traverse all of them. Hence, we utilize BO to efficiently explore more reliable layout variants with only a limited number of attempts. During the exploration, we repeatedly sample layout variants and derive their corresponding costs. The distribution of the cost function will be predicted based on the obtained samples, and layout variants that are predicted to have lower costs will be more likely to be sampled in the future. This way, we can use much fewer attempts to find the more reliable layout variant, significantly improving the efficiency of exploration.

IV. Feasible Domain Identification

Given a layout, we need to first identify the feasible domain, in which we can vary the layout and explore the more reliable layout variant. For example, in Fig. 5(b), the green regions represent the feasible domains of horizontal channel segments $Hseg_1$ and $Hseg_4$, respectively.

A. Layout Segmentation

To make the variation efficient, we choose to segment the layout into horizontal and vertical channel segments, and alter these segments to derive the layout variants.



(a) (b)

Fig. 5: Schematic of feasible domain identification. (a) The original layout. (b) Schematic of the identified feasible domains of two example horizontal channel segments.

Initially, we need to identify the terminals of segments. There are four types of points that can be terminals of segments: (i) starting points, where fluid is input into the design; (ii) ending points, where fluid is output from the design; (iii) branching points, which refer to points having more than two connections; (iv) corners, which refer to points with two connections forming a 90-degree angle. As shown in Fig. 5(b), the red dots represent the identified key points in the example layout shown in Fig. 5(a).

We then traverse all the terminals, identifying the horizontal and vertical connections among them as horizontal and vertical channel segments, respectively. As shown in Fig. 5(b), the green and blue segments between the red dots represent the identified horizontal and vertical channel segments, respectively. The connection relationships among these segments will also be recorded. Once segments in a certain direction are actively altered, segments in the other direction will be updated according to the connection relationships.

B. Relative Location Identification

During each round of variation, only the locations of segments perpendicular to the variation direction will be actively altered. To maintain the topology of the layout, we need to identify the relative locations of these segments, and only allow each segment to vary between its neighbors.

For example, when the variation direction is along the y-axis, for each horizontal channel segment $Hseg_i$, we will first search for all the horizontal channel segments overlapping with it in x-axis locations, and store them in $Overlap_i$. Whether or not two horizontal channel segments overlap in x-axis locations is determined by:

$$\mathbb{1}(\text{if overlap}) = \mathbb{1}(x_{i_{min}} \leq x_{j_{max}})\mathbb{1}(x_{i_{max}} \geq x_{j_{min}}) \quad (2)$$

where $\mathbb{1}(\cdot)$ is the indicator function, $x_{i_{min}}$, $x_{i_{max}}$, $x_{j_{min}}$, and $x_{j_{max}}$ represent the x-axis locations of the left and right terminals of $Hseg_i$ and $Hseg_j$, respectively.

For each $Hseg_j$ in $Overlap_i$, we then calculate d_{ij}, the y-axis distance from $Hseg_i$ to $Hseg_j$:

$$d_{ij} = y_j - y_i \quad (3)$$

where y_i and y_j represent the y-axis locations of $Hseg_i$ and $Hseg_j$, respectively. $Hseg_j$ that correspond to the smallest positive d_{ij} and the largest negative d_{ij} are identified as $Hseg_i$'s nearest upper and lower neighbors, respectively. As shown in Fig. 5(b), the upper and lower neighbors of $Hseg_4$ are $Hseg_2$ and $Hseg_6$, respectively. If there are no positive or negative d_{ij}, the upper or lower boundary of the layout will be regarded as the corresponding neighbor of $Hseg_i$. For example, in Fig. 5(b), the upper neighbor of $Hseg_1$ is the upper boundary. Besides, some $Hseg_j$ may be connected to the terminals of $Hseg_i$, making the corresponding $d_{ij} = 0$. Such a $Hseg_j$ is required to always keep the same y-axis location as $Hseg_i$, and it will be recorded in $Equal_i$ for later usage. Naturally, there can be at most two elements in $Equal_i$.

C. Synthesis of Constraints

We finally synthesize the constraints to identify the feasible domain for variation.

There are two types of constraints: (i) the upper and lower bounds of each segment; (ii) some segments need to have the same location in the variation direction. Taking the variation direction being along the y-axis as an example, we first synthesize type (i) constraints simply based on the neighbors of each $Hseg_i$. For type (ii) constraints, we refer to $Equal_i$. $Equal_i$ only contains $Hseg_j$ connected to $Hseg_i$, but the segments $Hseg_k$ connected to $Hseg_j$ are also required to have the same location as $Hseg_i$. Such relationship exists like a tree: $Hseg_i$ is the root, $Hseg_j$ in $Equal_i$ are the son nodes of $Hseg_i$, and $Hseg_k$ in $Equal_j$ are son nodes of $Hseg_j$. In this case, we recursively traverse nodes $Hseg_x$ and their corresponding $Equal_x$, until no new son nodes can be found. We then include all the upper and lower neighbors of segments in this tree into two sets UPs and $LOWs$, respectively. The type (i) constraint for $Hseg_i$ will be modified: the lowest segment in UPs and the upmost segment in $LOWs$ will become the new upper bound and lower bound, respectively. Type (i) constraints of other segments in this tree will be updated to let these segments have the same y-axis location as $Hseg_i$. These constraints build up the feasible domain for the variation, ensuring that layout variants within this domain maintain the same topology.

V. Bayesian-Optimization-based Reliability Enhancement

Since there can be thousands of possible layout variants within the previously identified feasible domain, it is impractical to traverse all of them. Hence, we utilize Bayesian optimization (BO) to efficiently explore more reliable layout variants.

As shown in Fig. 4, a set D is utilized to record all variations v and their corresponding costs f explored during the exploration. For a given feasible domain, we first randomly generate a few v and their corresponding layouts. These v and f will be recorded in D as the initial samples. When the size of D is smaller than the manually-set limit T, we will fit the surrogate model, sample a new v, derive the cost according to the corresponding layout, and update D. Once the size of D exceeds T, we will find the optimized v and its corresponding layout with the smallest cost. If this cost exceeds the improvement threshold, we will switch the direction of variation, pass the optimized layout to the

next round, and restart from the feasible domain identification stage. Otherwise, this optimized layout will be output as the final layout.

A. Problem Model

In a variation $v : \{v[1], v[2], \cdots, v[n]\}$, each element $v[i]$ relates to the location of a to-be-actively-altered segment, and n represents the number of such segments. Here, we take the variation direction being along the y-axis as an example without loss of generality. As the identified feasible domain only contains the relative locations of segments, the range of the possible location of each segment is related to other segments, thereby being unfixed in value. This makes it impractical to directly use the exact location of each segment in the variation v. To tackle this problem, for each to-be-actively-altered segment $Hseg_i$, we use a ratio between its upper and lower bounds as its corresponding $v[i]$. This way, each $v[i] \in [0, 1]$, enabling the exploration.

With a sampled v, we construct linear equations to solve the specific locations $y : \{y[1], y[2], \cdots, y[n]\}$, thereby deriving the corresponding layout variant. As a minimum distance d_{min} should be kept among channels, if both the upper and lower bounds are other segments $Hseg_{i_{up}}$ and $Hseg_{i_{low}}$, the real upper bound and lower bound of $y[i]$ should be $Realup_i = (y[i_{up}] - d_{min})$ and $Reallow_i = (y[i_{low}] + d_{min})$, respectively. For each linear equation, there are four possible scenarios, here we use b_{max} to represent the maximum boundary of the design in the current variation direction: (i) if both the upper and lower bounds are other segments:

$$y[i] - Reallow_i = v[i](Realup_i - Reallow_i) \tag{4}$$

(ii) if only the upper bound is another segment:

$$y[i] = v[i]Realup_i \tag{5}$$

(iii) if only the lower bound is another segment:

$$y[i] - Reallow_i = v[i](b_{max} - Reallow_i) \tag{6}$$

(iv) if neither of the bounds is another segment:

$$y[i] = v[i]b_{max} \tag{7}$$

We use the reliability evaluated by the quantification models proposed in [13] of the corresponding layout as the objective function f for v, which is defined as:

$$f(v) = (1 - w)P_{blockage} + wP_{leakage} \tag{8}$$

where $w \in [0, 1]$ is a user-defined parameter used to combine the anti-blockage and anti-leakage reliabilities. Our goal is to efficiently find the optimized variation $\hat{v}$ within only a limited number of attempts:

$$\hat{v} = \underset{v_i \in D}{\operatorname{argmin}} f(v_i), \ |D| <= T \tag{9}$$

Specifically, we employ a BO method named SMAC3 to address the problem [22].

B. Surrogate Model

Unlike traditional BO methods [7], [23], we use random forest (RF) [16] as the surrogate model instead of Gaussian processes (GP). RF consists of collections of regression trees, which are similar to decision trees but have real values at their leaves rather than class labels [24]. As a non-parametric approach, RF performs better on highly non-linear objective functions and is easy to parallelize [25], [26].

We construct an RF as a set of B regression trees, each built on m data points $\{v, f(v)\}$ sampled with replacement from the entire feasible domain. At each split point of each tree, a random subset of $[d \cdot p]$ of the d algorithm parameters is considered for splitting, where p is a parameter set as $p = \frac{5}{6}$. Another parameter is m_{min}, the minimum number of data points required in a node for it to be split further, which we set as $m_{min} = 10$. Finally, the number of trees is set as $B = 10$ to keep computational overhead low [27]. The RF's predictive mean μ_v and variance ${\sigma_v}^2$ for a new configuration v are computed as the empirical mean and variance of the individual trees' predictions for v.

C. Acquisition Function and Optimizer

Traditional expected improvement (EI) can suffer from vanishing gradients in high-dimensional spaces, making it difficult to optimize [20]. Therefore, we choose log expected improvement (logEI) as the acquisition function, which is defined as [20]:

$$\text{logEI}_{f(v)_{best}}(v) = \text{log_h}((\mu(v) - f(v)_{best})/\sigma(x)) + \log(\sigma(x)) \tag{10}$$

where log_h is mathematically equivalent to $\log \circ h$, and $f(v)_{best}$ is the best cost observed so far. This acquisition function ensures more stable gradients, leading to better results [28]. Meanwhile, we use a heuristic local search strategy to further improve the efficiency.

VI. Experimental Results

We demonstrate the performance of AutoRE from the perspectives of effectiveness and efficiency, respectively. All the experiments are carried out on an Intel(R) Xeon(R) Gold 6226R 2.90GHz Linux machine with 256 GB RAM.

A total of six testcases are used in our experiments: Syn1, Syn2 and Syn3 are synthesized based on the design in [29]; Real1, Real2 and Real3 are real chip designs from [10]–[12]. The reliability quantification models from [13] are utilized to evaluate the anti-blockage and anti-leakage reliabilities. According to our manufacturing experience, in the quantification models, pb_{unit}, the probability of a unit channel segment being blocked, is set as 2%; pl_{unit}, the probability of each unit channel segment leaking to any other unit channel segment at a unit distance, is set as 0.4%. For the settings of AutoRE, we let the initial variation direction to be along the y-axis, allow 150 explorations in each round of variation, and set the improvement threshold as 1%. For each testcase, we conduct three different experiments with w set as 0, 0.5, and 1, respectively.

A. Experiments on the Effectiveness

To start with, we test the effectiveness of AutoRE in enhancing the reliability of different layouts. The results are shown in TABLE I. It can be observed that AutoRE significantly enhances both the anti-blockage and anti-leakage reliabilities. Specifically, the anti-blockage reliability is improved by 42.53%, and the anti-leakage reliability is improved by 37.59% on average across all testcases.

TABLE I: Performance of Reliability Enhancement for Different Testcases

Testcase	Size	Channel Density	Improvement of Anti-Blockage Reliability				Improvement of Anti-Leakage Reliability				Average Runtime (s)
			$w = 0$	$w = 0.5$	$w = 1$	Average	$w = 0$	$w = 0.5$	$w = 1$	Average	
Syn1	15×11	20.00%	1.22%	1.22%	-0.35%	0.69%	4.07%	4.07%	6.64%	4.93%	48.47
Syn2	30×20	17.50%	152.15%	94.08%	33.44%	93.22%	-14.97%	25.31%	27.23%	12.52%	38.45
Syn3	65×40	13.10%	45.53%	37.63%	22.72%	35.29%	65.63%	188.74%	194.07%	149.48%	1425.57
Real1 [12]	39×31	14.89%	52.85%	52.85%	52.85%	52.85%	9.85%	9.85%	9.85%	9.85%	24.14
Real2 [10]	20×20	11.50%	19.99%	19.99%	-11.21%	9.59%	-0.02%	-0.02%	2.37%	0.78%	122.16
Real3 [11]	60×49	14.46%	126.09%	134.38%	-69.91%	63.52%	-23.80%	-1.43%	169.10%	47.96%	947.33
Average	1319	15.24%	66.31%	56.69%	4.59%	42.53%	6.79%	37.75%	68.21%	37.59%	434.40

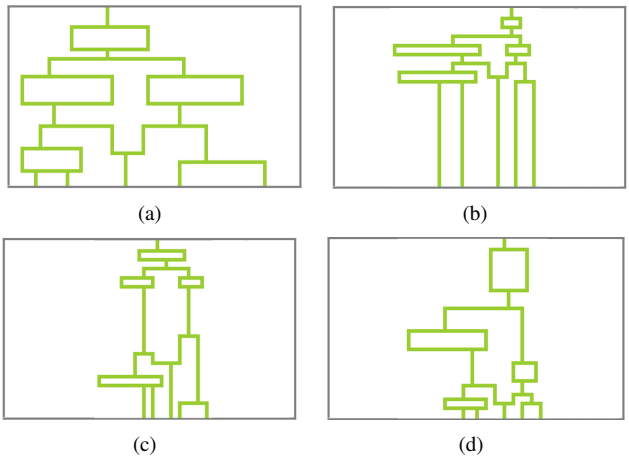


Fig. 6: Layouts of testcase Syn3. (a) The original layout. (b) The optimized layout with $w = 0$. (c) The optimized layout with $w = 0.5$. (d) The optimized layout with $w = 1$.

Additionally, the choice of w greatly influences the results. Generally, the closer w is to 0, the more AutoRE focuses on improving the anti-blockage reliability. This demonstrates AutoRE's flexibility in adapting to users' varying emphases on the two types of reliabilities. Conversely, as w increases, AutoRE tends more to enhance the anti-leakage reliability. Layout variants of Syn3 derived with different w settings are shown in Fig. 6. In extreme scenarios, such as Real3 with $w = 0$ and $w = 1$, AutoRE may even sacrifice one type of the reliability to achieve greater improvement in the other.

B. Experiments on the Efficiency

We also test the efficiency of AutoRE. According to the average runtime shown in TABLE I, AutoRE can complete the reliability enhancement for all testcases within 1500 s, which is a satisfactory speed for practical use.

To better demonstrate the efficiency of the BO-based exploration utilized by AutoRE, we compare it with the random exploration. Fig. 7 shows the result of the comparison using testcase Syn3. In the figures, each dot represents an exploration, with the y-axis value indicating the ratio between the current cost and the original cost, and the lines are drawn using the average of every ten explorations. It is evident that no matter what value w is set to adapt to users' different emphases on the two types of reliabilities, the BO-based exploration always consistently reduces the cost of

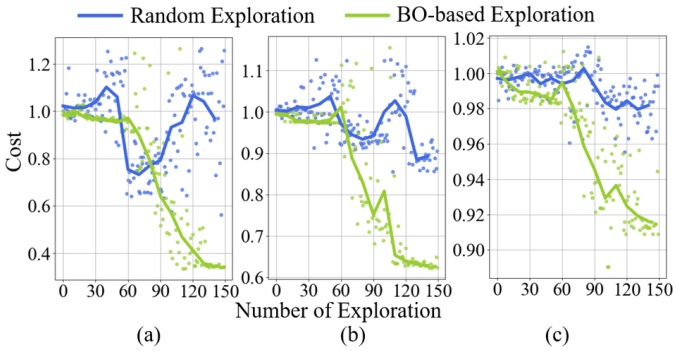


Fig. 7: Comparison of the efficiency between random exploration and the proposed BO-based exploration on testcase Syn3. (a) $w = 0$. (b) $w = 0.5$. (c) $w = 1$.

subsequent explorations, whereas the random exploration does not. Additionally, the lowest cost obtained through the BO-based approach are significantly better than those achieved by the random exploration. This demonstrates that the BO-based exploration is a more purposeful and efficient method for exploring within the feasible domain compared to the random exploration, proving AutoRE to be a highly efficient approach.

VII. Conclusion

In this paper, we have proposed the first automatic reliability enhancement tool for flow-based microfluidic biochips, named AutoRE. We have segmented the original layout and allowed segments to vary within a specified range while preserving the same topology. This approach has helped us identify a feasible domain, enabling us to explore among all feasible layout variants to achieve higher reliability. To improve the efficiency of our exploration, we have utilized Bayesian optimization (BO) and apply the reliability quantification models proposed [13] to evaluate the reliability of layout variants throughout the process. Experimental results have demonstrated that AutoRE is an efficient tool, effectively enhancing the reliability across all testcases by around 40% on average.

VIII. Acknowledgment

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust. This work is jointly supported by the Research Grants Council of Hong Kong SAR (No. CUHK14217022), and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation, No. 515003344).

REFERENCES

[1] S. Liang, M. Li, T.-M. Tseng, U. Schlichtamnn, and T.-Y. Ho, "Combinatorial-coding-based high-performance microfluidic control multiplexer: Design, synthesis, and adaptation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[2] M. Li, H. Gu, Y. Zhang, S. Liang, H. Gasvoda, R. Altay, I. E. Araci, T.-M. Tsneg, T.-Y. Ho, and U. Schlichtmann, "Late breaking results: Efficient built-in self-test for microfluidic large-scale integration (mlsi)," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[3] K. Hu, K. Chakrabarty, and T.-Y. Ho, *Computer-aided design of microfluidic very large scale integration (mVLSI) biochips*. Springer, 2017.

[4] S. Liang, Y. Zhang, R. Altay, H. Gasvoda, M. Li, I. E. Araci, T.-M. Tseng, U. Schlichtmann, and T.-Y. Ho, "Lamux: Optimized logic-gate-enabled high-performance microfluidic multiplexer design," 2024.

[5] K. Hu, F. Yu, T.-Y. Ho, and K. Chakrabarty, "Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1463–1475, 2014.

[6] K. Hu, T.-Y. Ho, and K. Chakrabarty, "Test generation and design-for-testability for flow-based mvlsi microfluidic biochips," in *2014 IEEE 32nd VLSI Test Symposium (VTS)*, pp. 1–6, IEEE, 2014.

[7] S. Liang, R. Fu, M. Li, T.-M. Tseng, U. Schlichtmann, and T.-Y. Ho, "Raber: Reliability-aware bayesian-optimization-based control layer escape routing for flow-based microfluidics," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[8] M. Li, Y. Zhang, J. Y. Lee, H. Gasvoda, I. E. Araci, T.-M. Tseng, and U. Schlichtmann, "Integrated test module design for microfluidic large-scale integration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1939–1950, 2022.

[9] Y. Zhu, B. Li, T.-Y. Ho, Q. Wang, H. Yao, R. Wille, and U. Schlichtmann, "Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[10] W.-L. Huang, A. Gupta, S. Roy, T.-Y. Ho, and P. Pop, "Fast architecture-level synthesis of fault-tolerant flow-based microfluidic biochips," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 1667–1672, IEEE, 2017.

[11] Y. Moradi, M. Ibrahim, K. Chakrabarty, and U. Schlichtmann, "Fault-tolerant valve-based microfluidic routing fabric for droplet barcoding in single-cell analysis," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1484–1487, IEEE, 2018.

[12] S. Liang, M. Li, T.-M. Tseng, U. Schlichtmann, and T.-Y. Ho, "Comux: Combinatorial-coding-based high-performance microfluidic control multiplexer design," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[13] S. Liang, M. Lian, M. Li, T.-M. Tseng, U. Schlichtmann, and T.-Y. Ho, "Armm: Adaptive reliability quantification model of microfluidic designs and its graph-transformer-based implementation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2023.

[14] W. Ji, T.-Y. Ho, J. Wang, and H. Yao, "Microfluidic design for concentration gradient generation using artificial neural network," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2544–2557, 2019.

[15] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[16] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pp. 507–523, Springer, 2011.

[17] B. Lei, T. Q. Kirk, A. Bhattacharya, D. Pati, X. Qian, R. Arroyave, and B. K. Mallick, "Bayesian optimization with adaptive surrogate models for automated experimental design," *Npj Computational Materials*, vol. 7, no. 1, p. 194, 2021.

[18] H. A. Chipman, E. I. George, and R. E. McCulloch, "Bart: Bayesian additive regression trees," 2010.

[19] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[20] S. Ament, S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy, "Unexpected improvements to expected improvement for bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 36, pp. 20577–20612, 2023.

[21] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[22] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 23, no. 54, pp. 1–9, 2022.

[23] R. Fu, J. Huang, M. Wang, Y. Nobuyuki, B. Yu, T.-Y. Ho, and O. Chen, "Bomig: A majority logic synthesis framework for aqfp logic," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–2, IEEE, 2023.

[24] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[25] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[26] R. Iranzad and X. Liu, "A review of random forest-based feature selection methods for data science education and applications," *International Journal of Data Science and Analytics*, pp. 1–15, 2024.

[27] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated f-race: An overview," *Experimental methods for the analysis of optimization algorithms*, pp. 311–336, 2010.

[28] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. Murphy, "Time-bounded sequential parameter optimization," in *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers 4*, pp. 281–298, Springer, 2010.

[29] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer optimization for flow-based mvlsi microfluidic biochips," in *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 1–10, 2014.

AmpereBleed: Exploiting On-chip Current Sensors for Circuit-Free Attacks on ARM-FPGA SoCs

Xin Zhang[124†], Yi Yang[12†], Jiajun Zou[12], Qingni Shen[12*],
Zhi Zhang[3*], Yansong Gao[3], Zhonghai Wu[1], Trevor E. Carlson[4]
[1]School of Software and Microelectronics, Peking University
[2]PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University
[3]The University of Western Australia
[4]National University of Singapore
Email: {zhangxin00, tkike, jjzou2002}@stu.pku.edu.cn, qingnishen@ss.pku.edu.cn,
{zhi.zhang, garrison.gao}@uwa.edu.au, wuzh@pku.edu.cn, tcarlson@comp.nus.edu.sg

***Abstract*—FPGAs offer superior energy efficiency and performance in parallel computing but are vulnerable to remote power side-channel attacks. Existing attacks rely on assumptions of co-resident crafted circuits and shared power delivery networks, limiting their practicality in real-world scenarios. In this paper, we present AmpereBleed, a novel `current`-based side-channel attack that exploits widely available `INA226` sensors in ARM-FPGA SoCs, bypassing the aforementioned two assumptions. AmpereBleed achieves 261× greater variations to victim activities compared to the popular ring oscillator (RO) circuit, fingerprints DNN models on the Xilinx Deep Learning Processor Unit (DPU) with 99.7% accuracy, and distinguishes the Hamming weights of RSA-1024 keys.**

I. Introduction

Over the past few decades, Field Programmable Gate Arrays (FPGAs) have gained widespread adoption due to their superior energy efficiency and performance in parallel computing applications [14], [19], [20]. For example, when performing inference tasks on Llama 2 (a large language model), the Xilinx Virtex UltraScale+ VU9P FPGA achieves up to a 12.75× reduction in energy consumption per token compared to the Intel Xeon Broadwell E5-2686 v4 CPU, and an 8.25× reduction compared to the NVIDIA RTX 3090 GPU. In addition, it delivers up to 2.46× faster inference speeds than the CPU and reaches 53% of the GPU's speed, despite the GPU's significantly higher base clock rate [20].

Despite these advancements, FPGAs face various security threats, particularly from remote power side-channel attacks, which can extract sensitive information from a victim FPGA circuit without physical proximity to it [16], [22], [34], [37], [38], [43]. Specifically, an attacker crafts a circuit that can co-reside with a victim circuit on the same FPGA board, sharing the power delivery network (PDN). As the PDN supplies power to both circuits, increases in the victim's voltage (thus power consumption) cause transient voltage drops in the crafted circuit. Further, the voltage fluctuations in the crafted circuit lead to variations in signal propagation delays within itself and can be reflected in its output. For instance, as a representative circuit design, ring-oscillator (RO) [43] uses a combinational loop to increment a counter and sample the counter at fixed time intervals, thereby observing voltage through the increments of this counter.

Critical assumptions of state-of-the-arts: Notably, all the aforementioned power side-channel attacks rely on the assumption of co-residence between malicious and benign circuits, which is feasible only in multi-tenant FPGA scenarios where the PDN is shared among tenants. While FPGA multi-tenancy has gained significant attention from academia and industry as a promising approach to virtualize FPGA resources in the cloud [25], it is not yet adopted in real-world commercial practice [15]. Also, these attacks assume a shared PDN that fluctuates when the victim circuit is actively running, with voltage fluctuations observable by the attacker. To mitigate them, PDN stabilizers are implemented to ensure the FPGA supply voltage fluctuates within a limited range (e.g., 0.825 V to 0.876 V on the Zynq UltraScale+ series FPGAs). Additionally, recent works [10], [35] have proposed isolated PDN for each tenant (e.g., ISO-TENANT [10]).

As the critical assumptions significantly raise the bar for remote power side-channel attacks, this leaves an open research question: *Can a remote power side-channel attack against FPGAs still be feasible if these assumptions are voided?*

Our work: In this paper, we answer this question affirmatively by presenting AmpereBleed, a novel attack that bypasses these limitations. Our key observation is that many embedding devices are equipped with `INA226` sensors that provide measurements of `current`, `voltage`, and `power` for various hardware components. In the context of ARM-FPGA SoCs, these sensors monitor the FPGA board and provide fine-grained `current` measurements for the FPGA which are the most sensitive to FPGA activities among the three available measurements. Further, these measurements are accessible to an unprivileged process running on the ARM processor via the `hwmon` subsystem. Leveraging this, AmpereBleed constructs a `current`-based side channel to infer FPGA activities without relying on either crafted circuits or a shared PDN.

In our evaluation, we first compare AmpereBleed's `current` with the ring oscillator (RO) circuit [43] (a well-

† Co-first authors. * Corresponding authors.

known crafted circuit that senses voltage fluctuations), regarding 161 distinct victim activation conditions. The results indicate that AmpereBleed-based `current` measurements achieve 261× greater variations than RO. Further, we validate that AmpereBleed's `current` is much more sensitive than `voltage` and `power` under the same settings. Last, we exploit AmpereBleed in two case studies. First, we mount a DNN model fingerprinting attack to show that AmpereBleed can distinguish among 39 DPU accelerators with a high accuracy of 99.7%. Second, we successfully apply AmpereBleed to distinguish Hamming weights of secret keys from an RSA-1024 circuit operating at 100 MHz.

Summary of contributions: The main contributions of this paper are as follows:

- We present AmpereBleed, a new current-based remote power side-channel attack against ARM-FPGA SoCs, without relying on either crafted circuits or a fluctuating PDN. AmpereBleed highlights the need for secure power monitoring implementation.
- We characterize how `current` measurements leak fine-grained information about victim activities and compare them with crafted circuit (i.e., RO [43]), `voltage` measurements, and `power` measurements.
- We apply AmpereBleed to two case studies, which fingerprint different DPU accelerators and distinguish the Hamming weight of RSA keys.

The source code to reproduce our experiments is released at https://github.com/Skyofmine007/AmpereBleed-DAC.

II. Background and Related Work

A. ARM-FPGA SoCs

Recent years have seen a growing demand for both energy efficiency and high performance, which has significantly driven advancements in high-performance FPGAs. Unlike CPUs and GPUs, which accelerate by optimizing instruction streams, FPGAs provide substantial benefits for customized computing due to their inherent re-programmability and high-performance potential. For example, AMD Xilinx's high-end AI Adaptive Compute Acceleration Platform (ACAP) VCK5000 achieves 1.8× frames per second per watt compared to Nvidia's flagship Ampere GPU (A100 SXM) in a standard MLPerf [31] benchmark.

ARM-FPGA System-on-Chips (SoCs) integrate ARM processor cores with an FPGA to provide a versatile solution for embedded systems [14]. The ARM cores, akin to standalone ARM processors such as the STM32 series, are equipped with various peripherals including UART, CAN, and GPIO ports, while FPGA serves as a customizable peripheral. This tight integration allows for high-speed, low-latency communication between CPU and FPGA resulting in improved performance, reduced cost, and a smaller physical footprint compared to using separate chips. Today, ARM-FPGA SoCs have become increasingly popular in Internet of Things (IoT) applications for commercial purposes, including autonomous driving [4], [9], 5G telecommunications [11], and medical devices [3], [8]. Companies like Baidu [9] and Subaru [4] have adopted this architecture in their autonomous driving and driver-assistance systems, respectively. For instance, Subaru has sold over 1 million vehicles with over 50 different models, which are equipped with ARM-FPGA SoCs as a solution for driver-assistance systems [4], highlighting the practicality of these SoCs in real-world deployments.

B. Remote Power Side Channel Attacks against FPGAs

Power side channel attacks have long been recognized as a significant source of information leakage. By analyzing power consumption, an adversary can infer victim's activities and extract sensitive data. To enable remote power side channel attacks on FPGAs, existing studies have explored a number of crafted circuits [22], [34], [37], [38], [43] that can detect on-chip voltage fluctuations. Specifically, all these works assume that a crafted circuit co-resides with the victim circuit on the same FPGA board, thereby sharing the same PDN with the victim circuit. A fundamental vulnerability identified in these studies is the inverse correlation between signal propagation delay and voltage variations at gate level. When engineered with precision, a crafted circuit can exploit voltage fluctuations, resulting in varying bit flips in its output stream. This behavior enables the creation of a novel side channel. The adversary can then leverage this output to various end-to-end attacks, including extracting cryptographic keys [34], [38], [43], fingerprinting victim workloads [18], and stealing DNN model architectures [42].

III. AmpereBleed

A. Threat Model

Aligned with prior works [14], [30], [36], [44], we consider an unprivileged attacker controlling a user process without special privileges, achievable through malicious over-the-air (OTA) updates [28], [29] or malware installation [12], [24]. The attacker's process is co-located with the victim circuits on the same ARM-FPGA SoC equipped with `INA226` sensors. The victim is a victim circuit with full control over the FPGA board for circuit deployment.

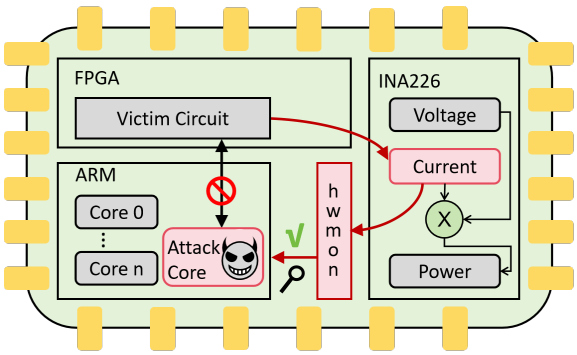


Fig. 1: Attack overview of AmpereBleed. The components inducing AmpereBleed's leakage are marked as red.

TABLE I: A comparison of the number of integrated INA226 sensors on various ARM-FPGA SoCs.

Property	ZCU102	ZCU111	ZCU216	ZCU1285	VEK280	VCK190	VHK158	VPK180
FPGA Family	Zynq UltraScale+	Zynq UltraScale+	Zynq UltraScale+	Zynq UltraScale+	Versal	Versal	Versal	Versal
FPGA Voltage (V)	0.825∼0.876	0.825∼0.876	0.825∼0.876	0.825∼0.876	0.775∼0.825	0.775∼0.825	0.775∼0.825	0.775∼0.825
CPU Model	Cortex-A53	Cortex-A53	Cortex-A53	Cortex-A53	Cortex-A72	Cortex-A72	Cortex-A72	Cortex-A72
DRAM	4 GB	4 GB	4 GB	8 GB	12 GB	8 GB	32 GB	12 GB
INA Sensors	18	14	14	21	20	17	22	19
Price ($)	3,234	14,995	16,995	32,394	6,995	13,195	14,995	17,995

B. Current-based Side Channel

As a first step for our analysis, we set out to review how existing attacks [16], [34], [43] monitor dynamic power consumption. Specifically, in a shared PDN that ideally ignores the effect of the stabilizers and can be modeled as an equivalent RLC matrix, an increase in the victim circuit's power causes transient voltage drops in neighboring circuits:

$$V_{drop} = I \cdot R + L \cdot \frac{\Delta I}{\Delta t}, \tag{1}$$

where V_{drop} depends on both steady-state current ($I \cdot R$ drop) as well as short transients ($\frac{\Delta I}{\Delta t}$ drop) caused by switching logic on the FPGA. It can be seen that current serves as an intermediate quantity of voltage change in these studies.

While previous studies have crafted various circuits [16], [22], [34], [37], [38], [43] to capture the voltage drop in the shared PDN, we note that dedicated PDN stabilizers [21] have been implemented to ensure the FPGA supply voltage fluctuates within a limited range, e.g., from 0.825 V to 0.876 V on the board with Zynq UltraScale+ series FPGAs (as shown in Table I). Besides, there are improvements in academia [10], [35] that aim at maintaining a stable voltage supply to each tenant circuit. However, it remains unclear whether the remote power side-channel threats can be fully mitigated after the voltage is completely stabilized.

Foundation: The foundation behind our work is rooted in an essential law of physics, where the overall dynamic power can also be measured by the supply voltage and current [1], as represented in Equation 2.

$$P_{dyn} = V_{dd} \cdot \sum I(LE, RAM, DSP, Clocks, \ldots), \tag{2}$$

where P_{dyn} is the dynamic power, V_{dd} is the supply voltage, and I is the current drawn by different types of computing elements, e.g., logic elements (LE), random access memory (RAM), digital signal processing (DSP), and clocks. Thus, even if the voltage remains stable, significant changes in power still cause noticeable changes in current.

To this end, we propose AmpereBleed, a novel remote power side channel attack that exploits current measurement to infer victim activities. As shown in Figure 1, AmpereBleed works on an ARM-FPGA SoC platform. Our attacker is located on one of the ARM cores without any co-resident circuit involved. To monitor on-chip current, we exploit INA226 sensors, which are integrated into a wide range of embedding devices for system monitoring, including ARM-FPGA SoCs. By accessing unprivileged current driver (i.e., hwmon) of INA226 sensors, AmpereBleed infers the activities of victim circuits through the red path.

TABLE II: Sensitive sensors that allow unprivileged access through the hwmon interfaces on the ZCU102 board [5].

Sensor	Description
ina226_u76	current, voltage, and power for full-power domain of the ARM processor cores.
ina226_u77	current, voltage, and power for low-power domain of the ARM processor cores.
ina226_u79	current, voltage, and power for FPGA's logic and processing elements.
ina226_u93	current, voltage, and power for DDR memory.

C. Unprivileged Sampling of On-chip Current

In this section, we provide implementation details about the unprivileged sampling of current. First, to demonstrate the widespread availability of the exploited INA226 sensors, we have examined a range of SoC boards across two FPGA families (i.e., Zynq UltraScale+ and Versal). Table I lists 8 representative boards, all of which include INA226 sensors. Notably, AmpereBleed's applicability extends well beyond these examples. For instance, the ZCU106 board, featuring 14 INA226 sensors, is not listed due to its similarity in price and release date to the ZCU102. Besides, Table II showcases four sensitive INA226 sensors out of the 18 integrated on the ZCU102, providing on-chip monitoring for various components including CPU, FPGA, and DRAM.

The INA226 sensors provide current and voltage measurements at specific monitoring points. They also calculate power indirectly from these two measurements when properly calibrated [2]. We identify current as the most potentially exploitable for two key reasons: First, for the victim FPGA workload, its voltage measurements are constrained by a fixed resolution of 1.25 mV and exhibit minimal changes (as shown in Table I, resulting in a limited range of voltage variations. Second, the power measurements are derived from current and voltage, with their resolution fixed at a ratio of 25 relative to the current resolution [2]. Given the limited variability in voltage, the power measurements are almost synchronized to the current measurements, but the low bits are truncated. In our evaluation, we compare current measurements against voltage and power measurements to validate the identification.

We use the hardware monitoring (also known as hwmon) subsystem to access the INA226 sensors from the ARM side without requiring any privileges. The subsystem is also available in x86 processors [23]. We access the current measurements on each INA226 sensors through

Each INA226 sensor is equipped with a dedicated shunt resistor, enabling current to be measured based on its voltage and resistance.

`/sys/class/hwmon/hwmon[0-*]/curr1_input` in the Linux file system. Notably, these interfaces provide a resolution of ±1 mA and a configurable updating interval between 2 and 35 ms. The resolution is determined by the hardware sensors, and the default updating interval is set to 35 ms. Although the updating interval can be adjusted at runtime, modifying it requires root privileges. Since AmpereBleed assumes an unprivileged attacker, we use the default updating interval throughout this paper.

IV. Evaluation

Experimental machine: We evaluate AmpereBleed on a Xilinx ZCU102 SoC, which features four Cortex-A53 cores with a base frequency of 1200 MHz and an FPGA fabric operating at 300 MHz. The FPGA fabric includes 274,080 lookup tables, 548,160 flip-flops, and 2,520 digital signal processing blocks. We use PetaLinux to customize Linux-based operating system for a specific SoC, which is a popular approach for Xilinx ARM-FPGA SoCs. All system configurations, including `hwmon` settings and dynamic voltage and frequency scaling (DVFS) policies, are kept by default.

A. Characterizing AmpereBleed

As outlined above, our AmpereBleed attack leverages the unprivileged `hwmon` interface to acquire `current` measurements that are related to the victim's behavior in theory. Here, we demonstrate that these `current` measurements are sufficient to distinguish various activities of the victim FPGA.

Varying computing workloads: Then, we deploy 160 k power virus instances described in Gand et al. [17] to cover major routing places of our ZCU102 board. They serve as a victim to stress FPGA logic, aligned with Zhao et al. [43]. We then divide them into 160 groups and each group has 1 k evenly-distributed instances. After deploying the bitstream onto the FPGA board, we can dynamically activate different numbers of these groups from the ARM processor to make the `current` vary by 161 levels.

Distinguishing different victim activities: Our first experiment is to characterize the `hwmon`'s output under different FPGA workloads. To this end, we activate a varying number of power virus instances, making the victim activities vary from 161 levels. To construct a baseline, we follow Zhao et al. [43] to reproduce RO circuits and distribute them throughout the FPGA board to average dependence on spatial proximity to activated power virus instances. At each level, we collect 10 k samples about `current`, `voltage`, and `power` of FPGA logic from `hwmon` and the RO circuits, respectively, and compute the mean of these samples as the final value to derive the Pearson correlation coefficient which quantifies the strength and direction of the linear relationship.

Figure 2 illustrates the correlation between these measurements and the number of active power virus instances. FPGA `current` and `power` exhibit a strong linear relationship with the number of activated power virus instances, with a Pearson correlation coefficient of 0.999. Additionally, FPGA `voltage` achieves a Pearson correlation of 0.958,

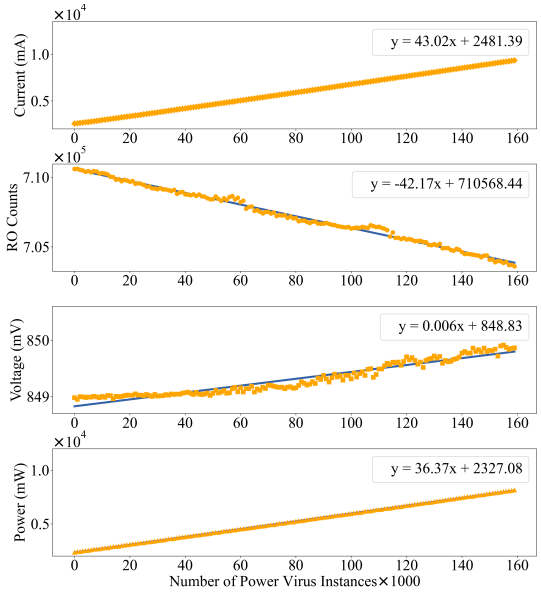
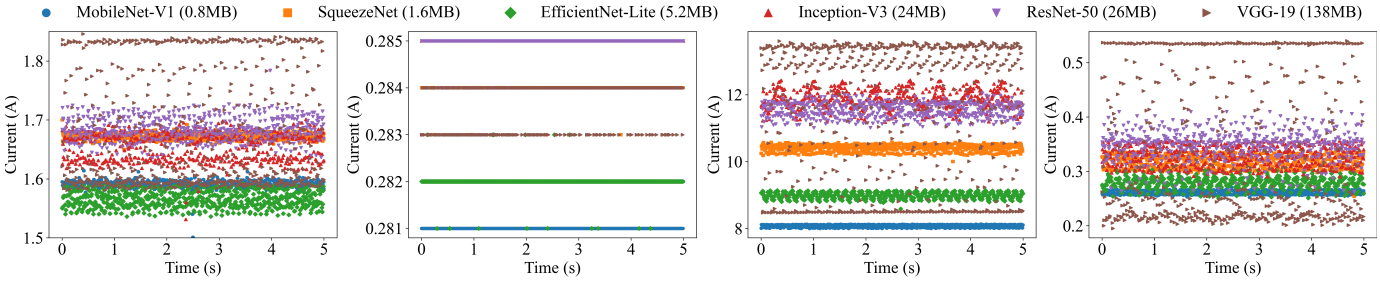


Fig. 2: The FPGA `current`, `voltage`, and `power` accessed via `hwmon`, along with RO counts, versus the number of activated power virus instances.

and RO achieves -0.996. This suggests that FPGA `current` and `power` readings, accessed via `hwmon`, correlate more strongly with victim activities under this experimental setup. Furthermore, we compute the linear function for each type of measurements. Due to `power` readings being updated at a maximum resolution of 25 mW, the difference between consecutive settings is limited to 1-2 least significant bits (LSBs). `voltage` measurements also show a limited linear correlation coefficient of 0.006 and support a fixed and coarse-grained resolution of 1.25 mV, leading to only slight LSB changes even under high workloads of 160 k activated power virus instances. In contrast, `current` measurements support a fine-grained resolution of 1 mA and vary approximately 40 LSBs per setting, enhancing their sensitivity to different victim activities and, consequently, improving attack performance as detailed in the following subsections. We clarify that the reason why `current` measurements do not start from 0 is due to the static workloads [26] caused by inactivated but deployed power virus instances.

B. Fingerprinting DPU Accelerators

Xilinx DPU is a commercial framework for deploying pre-trained DNN models on FPGA boards. To protect its confidentiality and intellectual property, DPU encrypts its hardware description language (HDL) files at the source code level, following IEEE-1735-2014 V2 standard [6]. Due to this encryption, attackers cannot determine how the model inference is performed, such as the sequence of matrix computations and intensive memory read-write activities, increasing the difficulty of attack. Till now, the only one attack against DPU is to reverse-engineer the encrypted implementation details about DPU, which requires a high-resolution probe and physical proximity to sample electromagnetic signals [19].



(a) Full-power CPU (b) Low-power CPU (c) FPGA (d) DRAM

Fig. 3: `current` patterns leaked from four sensors during different DNN model inferences.

In this subsection, we demonstrate a DNN model fingerprinting attack against the DPU framework, notably achieved without any specialized equipment. Our key insight is that distinct `current` variation patterns during inference can expose information about the underlying neural network operations. Knowledge of the DNN architecture poses a significant intellectual property risk [13], and can facilitate further AI attacks (e.g., adversarial example attack [27] and membership inference attack [40]).

Victim DPU accelerator: To deploy DPU framework on ZCU102 board, we utilize its official image provided by AMD-Xilinx [7]. This image incorporates Python version 3.9.9, Linux kernel version 5.15, and Vitis AI version 3.0. To minimize the impact of process scheduling interference, we schedule the task of triggering DPU inference onto CPU core 0, and the sampling tasks onto CPU core 3. We select a complete suite of image recognition models from Vitis AI Library as victim accelerators, including 39 architectures over 7 diverse architecture families. We conduct inference using the ImageNet ILSVRC Test set, resizing the input images to comply with the specifications of the victim accelerators. By default, the victim runs each model in series for 5 seconds.

Experimental setup: Aligned with previous work [27], [41], our model fingerprinting attack has two distinct phases: an *offline preparation* phase and an *online classification* phase. In the *offline preparation* phase, we collect traces of `current`, `voltage`, and `power` by separately sampling each interface to build a series of well-trained classifiers, each of which can classify a specific type of side channel traces to their corresponding model architectures. In the *online classification* phase, we issue queries to a black-box accelerator running on our FPGA board, triggering its model inference. Concurrently, we collect side-channel traces from `hwmon`. With the collected traces, we then use the corresponding classifier to fingerprint the architectures of encrypted DPU accelerators.

Since our fingerprinting attack is essentially a classification task with straightforward features, we use random forest (RForest) to conduct the evaluation, due to its suitability for handling high-dimensional data and identifying feature importance. We configure RForest with 100 trees and set the maximum depth to 32. The model uses Gini impurity as the splitting criterion, measuring class impurity within nodes to assess the quality of splits. We apply bootstrap sampling to build each decision tree, ensuring each tree is trained on a unique subset of data by selecting samples with replacement. For validation, we perform a 10-fold cross-validation where, in each iteration, 9 folds serve as training data and the remaining fold is used for testing.

TABLE III: Classification accuracy for encrypted accelerator fingerprinting. The baseline of random guess is 0.0256.

Sensor \ Duration		1 s	2 s	3 s	4 s	5 s (Full-length)
Current (Full-power CPU)	Top-1	0.823	0.830	0.832	0.834	0.837
	Top-5	0.981	0.980	0.982	0.981	0.982
Current (Low-power CPU)	Top-1	0.429	0.498	0.548	0.557	0.557
	Top-5	0.870	0.905	0.914	0.915	0.915
Current (DRAM)	Top-1	0.953	0.961	0.960	0.959	0.958
	Top-5	0.998	0.999	0.999	0.999	0.999
Current (FPGA)	Top-1	**0.980**	**0.985**	**0.997**	**0.997**	**0.997**
	Top-5	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Voltage (FPGA)	Top-1	0.080	0.099	0.116	0.118	0.116
	Top-5	0.317	0.323	0.327	0.326	0.330
Power (FPGA)	Top-1	0.929	0.973	0.989	0.991	0.989
	Top-5	0.997	0.997	0.997	0.996	0.996

Experimental results: Figure 3 illustrates examples of `current` traces collected via four `hwmon` interfaces while the DPU executes 6 DNN models separately, including MobileNet-V1, SqueezeNet, EfficientNet-Lite, Inception-V3, ResNet-50, and VGG-19, with their respective model sizes indicated. Each of the selected models produces a unique `current` patterns, influenced by the computing activities during the inference phase. The DPU's activity, involving the CPU, FPGA, and DRAM, is reflected in these `current` traces from these components, effectively captured by the `hwmon` interfaces and highlighting significant model fingerprinting vulnerabilities.

Table III presents the fingerprinting results of four sensitive `current` sensors, an FPGA `voltage` sensor, and an FPGA `power` sensor captured by `hwmon`, with each sensor's top-1 accuracy displayed in the first row and top-5 accuracy in the second row. We observe that all the `current` sensors achieve a high accuracy when the duration exceeds 3 s. Notably, FPGA `current` measurements reach a high accuracy of 99.7%, indicating a strong correlation between FPGA `current` variations and DPU activities. Given the greatly varied workloads of different DPU accelerators on the FPGA logic (as shown in Figure 3(c)), indirect `power` measurements that are computed from `current` and `voltage` also achieve a high success

rate of 98.9%. In contrast, the FPGA `voltage` measurements achieve a significantly lower accuracy of 11.6%.

C. Example Attack on an RSA

Beyond fingerprinting attacks, we present a concrete attack example that shows how AmpereBleed can infer the Hamming weight of secret keys from an RSA module implemented on an FPGA. Specifically, we assume an unprivileged attacker running on ARM cores targets a victim circuit performing RSA encryption. Till now, the only one attack against RSA-1024 circuit is Zhao et al. [43], where a malicious process samples the RO outputs with a high sampling frequency of 2 MHz. However, their victim RSA circuit operates at a low frequency of 20 MHz and RO circuits have been banned by commercial cloud providers (e.g., AWS [33]).

Victim RSA accelerator: We follow Zhao et al. [43] to implement an RSA-1024 circuit as the victim, and modify it to operate at 100 MHz. This circuit employs the Square-and-Multiply algorithm, which consists of two dedicated modular multiplication modules and a state machine. The state machine iterates through each bit of the 1024-bit input exponent, starting from the least-significant bit. One multiplication module is used for computing the square term and the other module is used for the multiplication operation. For every iteration, if the current least-significant bit of the iteratively-shifted exponent is '1', both the two modules are activated to compute the square, inducing high switching activities. Otherwise, only a square operation will be activated. Both multipliers are synchronized to complete computation within the same cycle for each iteration of the loop. To protect the secret key, this implementation embeds the key within the encrypted bitstream. Once the circuit is deployed on an FPGA, the private key remains inaccessible, even to privileged users.

Experimental setup: To perform power side channel attack, our attack program continuously records the FPGA `current` measurements from the `hwmon` interface, using the sampling frequency of 1 kHz to collect 100,000 samples. During the `current` collecting, the RSA modular repeatedly encrypts a random plaintext. To demonstrate that FPGA `current` measurements can indeed leak information about private keys, we construct 17 distinct keys whose Hamming weights increase in intervals of 64, except for the first key that is chosen as 1 (since the RSA circuit does not support exponentiation with a value of 0). To make a comparison, we also use the FPGA `power` measurements to repeat the above procedure.

Experimental results: Figure 4 shows the statistical distribution of FPGA `current` and `power` measurements during RSA-1024 execution. Clearly, the attacker can use the FPGA `current` measurements to infer the Hamming weights of these private keys. As a comparison, the `power` measurements could only categorize the 17 keys into 5 groups. We note that knowledge of the Hamming weight can greatly reduce the search space of RSA's key brute force attack, and serves as a precursor for statistical analysis attacks [32].

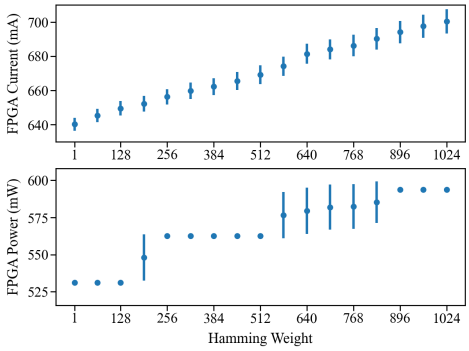


Fig. 4: The impact of the Hamming weight of RSA-1024 keys on FPGA `current` and `power` measurements.

V. Discussion

Mitigation: Since AmpereBleed exploits unprivileged access to `INA226` sensors, restricting their access to privileged users can effectively mitigate the unprivileged attacks that are demonstrated in this paper. However, this restriction may badly affect benign programs that rely on these interfaces for performance monitoring, fault detection, and system management purposes. Besides, it requires driver or kernel updates for all affected devices, and cannot protect the legacy systems.

Future work: While this study primarily focuses on the Xilinx ARM-FPGA platforms, it raises the question of whether similar vulnerabilities exist in other FPGA SoCs (e.g., Intel's ARM-FPGA SoC products) as more FPGA boards are integrating a CPU as well as on-chip sensors within a single die. Besides, this paper only considers unprivileged attack scenarios, whether these `INA226` sensors could be exploited to attack trusted execution environments (TEEs) implemented on FPGA [39] remains an open question. Last, as the on-chip `current` can also be measured by physical devices [26], we are also interested in its security implications as a physical side channel.

VI. Conclusion and Acknowledgments

In this work, we propose AmpereBleed, a new type of remote power side channel attack to infer FPGA activities. Specifically, AmpereBleed targets the ARM-FPGA SoC architectures and exploits the unprivileged `hwmon` subsystem to acquire `current` measurements. To demonstrate the viability of AmpereBleed, we first characterize the behavior of `current` measurements under varying power consumption conditions, experimental results of which show that fine-grained victim activities can be distinguished. We then successfully apply AmpereBleed to fingerprint DPU accelerators and infer the Hamming weights of different RSA private keys.

We thank the anonymous reviewers for their feedback. In addition, we thank Yuhui Zhang and Yusi Feng for their valuable comments that helped us to improve the work. This work was supported by and the Science and Technology Development Program of Two Districts in Xinjiang Province, China under Grant No. 2024LQ03004, the National Key R&D Program of China (No. 2022YFB2703301), and the Singapore Ministry of Education (Grant No. T1251RES2403).

REFERENCES

[1] "FPGA Total Power Components Introduction." [Online]. Available: https://www.intel.com/content/www/us/en/support/programmable/support-resources/power/pow-overview.html

[2] "INA226 36V, 16-Bit, Ultra-Precise I2C Output Current, Voltage, and Power Monitor With Alert." [Online]. Available: https://www.ti.com/lit/ds/symlink/ina226.pdf

[3] "Pneumonia and COVID-19 Detection from X-Ray Images using Vitis-AI and Deployed by IoT GreenGrass." [Online]. Available: https://github.com/splineai-cloud/COVID-XS

[4] "Subaru Selects AMD MPSoCs for EyeSight System." [Online]. Available: https://iot-automotive.news/baidu-launches-global-first-mass-produced-autonomous-driving-computing-platform/

[5] "ZCU102 Evaluation Board User Guide (UG1182)." [Online]. Available: https://docs.amd.com/v/u/en-US/ug1182-zcu102-eval-bd

[6] "IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)," *IEEE Std 1735-2014 (Incorporates IEEE Std 1735-2014/Cor 1-2015)*, pp. 1–90, 2015.

[7] "Quick Start Guide for Zynq UltraScale+," 2024. [Online]. Available: https://xilinx.github.io/Vitis-AI/3.0/html/docs/quickstart/mpsoc.html

[8] "Xilinx Medical Clinical Solutions," 2024. [Online]. Available: https://www.xilinx.com/publications/prod_mktg/medical-clinical-brochure.pdf

[9] "Xilinx SoC FPGA Powers Baidu's Apollo Driverless Platform," 2024. [Online]. Available: https://www.electronicdesign.com/markets/automotive/article/21119589/xilinx-soc-fpga-powers-baidus-apollo-driverless-platform

[10] M. K. Ahmed and C. Bobda, "ISO-TENANT: Rethinking FPGA Power Distribution Network (PDN): A Hardware Based Solution for Remote Power Side Channel Attacks in FPGA," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024.

[11] N. Bartzoudis, J. Rubio Fernández, D. López-Bueno, A. Román Villarroel, and A. Antonopoulos, "Agile FPGA Computing at the 5G Edge: Joint Management of Accelerated and Software Functions for Open Radio Access Technologies," *Electronics*, vol. 13, 2024.

[12] W. Diao, X. Liu, Z. Li, and K. Zhang, "No Pardon for the Interruption: New Inference Attacks on Android Through Interrupt Timing Analysis," in *IEEE Symposium on Security and Privacy*, 2016, pp. 414–432.

[13] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal, "DeepTheft: Stealing DNN Model Architectures through Power Side Channel," *IEEE Symposium on Security and Privacy*, 2023.

[14] J. Ge and F. Zhang, "SnapMem: Hardware/Software Cooperative Memory Resistant to Cache-Related Attacks On ARM-FPGA Embedded SoC," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[15] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *IEEE Symposium on Security and Privacy*, 2020.

[16] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Design, Automation & Test in Europe Conference & Exhibition*, 2020.

[17] D. R. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[18] M. Gobulukoglu, C. Drewes, W. Hunter, R. Kastner, and D. Richmond, "Classifying Computations on Multi-Tenant FPGAs," in *Design Automation Conference*, 2021, pp. 1261–1266.

[19] C. Gongye, Y. Luo, X. Xu, and Y. Fei, "Side-Channel-Assisted Reverse-Engineering of Encrypted DNN Hardware Accelerator IP and Attack Surface Exploration," in *IEEE Symposium on Security and Privacy*.

[20] A. He, D. Key, M. Bulling, A. Chang, S. Shapiro, and E. Lee, "HLSTransform: Energy-Efficient Llama 2 Inference on FPGAs Via High Level Synthesis," 2024. [Online]. Available: https://arxiv.org/abs/2405.00738

[21] H. Homulle, S. Visser, B. Patra, and E. Charbon, "Design techniques for a stable operation of cryogenic field-programmable gate arrays," *Review of Scientific Instruments*, vol. 89, no. 1, 2018.

[22] D. Jayasinghe, B. Udugama, and S. Parameswaran, "1LUTSensor: Detecting FPGA Voltage Fluctuations using LookUp Tables," *Cryptographic Hardware and Embedded Systems*, 2024.

[23] T. Kim and Y. Shin, "ThermalBleed: A Practical Thermal Side-Channel Attack," *IEEE Access*, 2022.

[24] R. Li, W. Diao, Z. Li, J. Du, and S. Guo, "Android Custom Permissions Demystified: From Privilege Escalation to Design Shortcomings," in *IEEE Symposium on Security and Privacy*, 2021, pp. 70–86.

[25] J. M. Mbongue, D. T. Kwadjo, A. Shuping, and C. Bobda, "Deploying Multi-tenant FPGAs within Linux-based Cloud Infrastructure," *ACM Trans. Reconfigurable Technol. Syst.*, 2021.

[26] A. Moradi, "Side-Channel Leakage through Static Power – Should We Care about in Practice?" in *CHES*, 2014.

[27] K. Patwari, S. M. Hafiz, H. Wang, H. Homayoun, Z. Shafiq, and C.-N. Chuah, "DNN Model Architecture Fingerprinting Attack on CPU-GPU Edge Devices," in *European Symposium on Security and Privacy (EuroS&P)*, 2022, pp. 337–355.

[28] C. Plappert and A. Fuchs, "Secure and Lightweight ECU Attestations for Resilient Over-the-Air Updates in Connected Vehicles," in *Annual Computer Security Applications Conference*, 2023, p. 283–297.

[29] Plappert, Christian and Fuchs, Andreas, "Secure and Lightweight Over-the-Air Software Update Distribution for Connected Vehicles," in *Annual Computer Security Applications Conference*, 2023, p. 268–282.

[30] G. D. RE, J. Krautter, and M. B. Tahoori, "Leaky noise: New side-channel attack vectors in mixed-signal IoT devices," in *Cryptographic Hardware and Embedded Systems*, 2019, pp. 305–339.

[31] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf Inference Benchmark," in *International Symposium on Computer Architecture*, 2020, pp. 446–459.

[32] S. Sarkar and S. Maitra, "More on correcting errors in RSA private keys: Breaking CRT-RSA with low weight decryption exponents," *Cryptographic Hardware and Embedded Systems*, 2012.

[33] A. W. Services. (2024) AWS EC2 FPGA HDK+SDK errata. [Online]. Available: https://github.com/aws/aws-fpga/blob/master/ERRATA.md

[34] D. Spielmann, O. Glamočanin, and M. Stojilović, "RDS: FPGA Routing Delay Sensors for Effective Remote Power Analysis Attacks," *Cryptographic Hardware and Embedded Systems*, pp. 543–567, 2023.

[35] B.-H. Su, J.-S. Tang, H.-J. Lee, and C.-B. Tzeng, "Noise Analysis and Improvement of Power Supply Network Based on Power Integrity," in *International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*, 2023.

[36] X. Tan, Z. Ma, S. Pinto, L. Guan, N. Zhang, J. Xu, Z. Lin, H. Hu, and Z. Zhao, "SoK: Where's the "up"?! A Comprehensive (bottom-up) Study on the Security of Arm Cortex-M Systems," 2024.

[37] B. Udugama, D. Jayasinghe, H. Saadat, A. Ignjatovic, and S. Parameswaran, "A power to pulse width modulation sensor for remote power analysis attacks," *Cryptographic Hardware and Embedded Systems*, 2022.

[38] Udugama, Brian and Jayasinghe, Darshana and Saadat, Hassaan and Ignjatovic, Aleksandar and Parameswaran, Sri, "VITI: A tiny self-calibrating sensor for power-variation measurement in FPGAs," *Cryptographic Hardware and Embedded Systems*, pp. 657–678, 2022.

[39] K. Xia, Y. Luo, X. Xu, and S. Wei, "SGX-FPGA: Trusted Execution Environment for CPU-FPGA Heterogeneous Architecture," in *Design Automation Conference*, 2021, pp. 301–306.

[40] S. Zhai, H. Chen, Y. Dong, J. Li, Q. Shen, Y. Gao, H. Su, and Y. Liu, "Membership Inference on Text-to-Image Diffusion Models via Conditional Likelihood Discrepancy," in *Annual Conference on Neural Information Processing Systems*, 2024.

[41] X. Zhang, Z. Zhang, Q. Shen, W. Wang, Y. Gao, Z. Yang, and Z. Wu, "ThermalScope: A Practical Interrupt Side Channel Attack Based On Thermal Event Interrupts," in *Design Automation Conference*, 2024.

[42] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote FPGA side-channel analysis," *IEEE Transactions on Information Forensics and Security*, 2021.

[43] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *IEEE Symposium on Security and Privacy*, 2018.

[44] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *USENIX Security Symposium*, 2019.

AcclMT: A Highly Resource-Efficient and Flexible Poseidon Hash-Based Merkle Tree Architecture

Changxu Liu[12], Hao Zhou[12], Lan Yang[12], Yifei Feng[3], Zheng Wu[12], Zhuoyuan Yang[12], Yinlong Li[4], Shiyong Wu[4*] and Fan Yang[12*]

[1] *State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China*
[2] *School of Microelectronics, Fudan University, Shanghai, China*
[3] *School of Computer Science, Fudan University, Shanghai, China*
[4] *Hardware R&D Center, Shanghai Academy of Future Internet Technology, Shanghai, China*

***Abstract*—Merkle Tree is a fundamental cryptographic primitive in Zero-Knowledge Proof (ZKP) protocols, sharing significant computational workloads with the Number Theoretic Transform (NTT) in zk-STARK schemes. Merkle Tree is a tree structure where nodes are primarily generated through hash computations. Among them, Poseidon Hash, as a ZK-friendly hash function, has emerged as one of the most widely adopted choices. Therefore, hardware acceleration of building Merkle Tree based on Poseidon Hash can significantly enhance the performance of ZKP protocols. We propose AcclMT, a highly resource-efficient and flexible Poseidon Hash-based Merkle Tree architecture. Our design employs hardware-software co-design and optimizes the hashing data flow, resulting in an area-efficient Poseidon Hash engine that improves modular multiplication resource utilization. Furthermore, AcclMT uses these engines alongside hierarchical on-chip cache and optimized task scheduling for building large Merkle Trees. It also supports flexible parameter configurations for various requirements. Experimental results show that our proposed Poseidon Hash engine achieves a 14.3 × speedup compared to the latest FPGA-based work. By improving resource utilization, it also reduces area usage by 14.8% compared to unoptimized design. AcclMT achieves up to 1665 × speedup over software implementations in building Merkle tree, with average utilization of 95.9% and 99.2% for the two hash engines.**

***Index Terms*—Zero-Knowledge Proof, Merkle Tree, Poseidon Hash, Hardware Acceleration**

I. INTRODUCTION

The Merkle Tree is a key primitive in Zero-Knowledge Proof (ZKP), a commonly used privacy-preserving algorithm. It is often utilized in zk-STARK-based ZKP [1] schemes for polynomial commitments. Unlike the Multi-Scalar Multiplication-based commitment schemes in zk-SNARK [2], Merkle Trees offer higher computational and spatial efficiency. When building a Merkle Tree, a large sequence of elements is first hashed to generate leaf nodes, which are then repeatedly hashed and combined layer by layer until the root node is obtained. The trend has shifted toward choosing ZK-friendly arithmetic hash functions like Poseidon Hash [3]. Unlike traditional hashes such as SHA, Poseidon Hash offers higher computational efficiency and fewer constraints, making it widely applied in various scenarios [4]–[6]. In zk-STARK and some newer zk-SNARK schemes, hash functions—primarily in the form of Merkle Trees—represent the largest computational workload alongside the Number Theoretic Transform, with the two accounting for over 65% of total computation [7]. Poseidon Hash, being more computationally expensive, may further increase this proportion. Previously, hardware acceleration efforts for ZKP mainly focused on Multi-Scalar Multiplication (MSM) [8]–[11] and the Number Theoretic Transform (NTT) [10]–[12], while neglecting improvements in hash computations and Merkle Tree. According to Amdahl's Law [13], exclusively accelerating MSM and NTT would leave ZK system performance severely bottlenecked by the unoptimized hash operations, which dominate the remaining computation overhead [14].

However, building a Poseidon Hash-based Merkle Tree presents significant challenges. One is efficiently managing the computational workload, as a single Poseidon Hash may require hundreds or thousands of modular multiplications, while a Merkle Tree can involve millions of such computations. Another challenge lies in handling the vast, temporally sparse intermediate data generated during computation. Despite their space efficiency, Merkle Trees can produce several gigabits of intermediate data, whose sparse distribution reduces computational unit utilization. Keeping extensive intermediate data on-chip is expensive, while frequent transfers to off-chip storage further impact efficiency.

To address these challenges, accelerating Poseidon Hash-based Merkle Tree implementations becomes crucial. ZPrize [15] introduces a track for Poseidon Hash hardware acceleration, and Ingonyama develops ICICLE [16], a cryptographic library optimized for hardware platforms, including Poseidon Hash-based Merkle Trees. TRIDENT [17], a Poseidon Hash accelerator for FileCoin on FPGA, delivers significant performance gains over advanced CPUs. Irreducible also proposes the first fully pipelined FPGA architecture for ZKP-friendly Merkle Trees using Poseidon Hash [18].

However, we believe that the above works have limitations that make them unsuitable as reference paradigms or lack practical considerations, as detailed below:

- **Lack of Flexibility and Generalizability:** Their implementations are often tailored to specific protocols, such as Irreducible's approach [18] for Plonky2 and TRIDENT [17] for the FileCoin protocol.
- **Low Resource Utilization:** The most expensive modular multipliers are not fully utilized. We estimate that nearly half of TRIDENT's [17] modular multipliers remain idle most of the time, resulting in reduced area efficiency.
- **Imbalanced Performance and Resource Overhead, with Limited Optimization Strategies:** Irreducible's approach [18] fully unrolls the Poseidon Hash for optimal performance in 64-bit Goldilocks fields. However, for ZKP protocols using larger finite fields, the hardware cost of full unrolling becomes unsustainably high.

In this paper, we present AcclMT, a highly resource-efficient and flexible Poseidon Hash-based Merkle Tree architecture. Our proposed flexible design supports the configuration of several typical parameters, catering to diverse application contexts. We optimize the proposed Poseidon Hash engine through software-hardware co-design, efficiently allocating different types of Poseidon permutations to two

*Corresponding author: yangfan@fudan.edu.cn, wusy@safit.org.cn

sub-engines, achieving higher resource utilization than unoptimized architectures. In Merkle Tree optimization, our approach efficiently decomposes and schedules tasks, using a limited hierarchical on-chip cache to build Merkle Trees of various sizes. Although our design uses a 256-bit data width, it can be adjusted for specific applications. Our key contributions are:

- We propose a Poseidon Hash engine combining a resource-efficient partial-round engine with a resource-intensive full-round engine. We optimize the scheduling of different rounds in Poseidon Hash across two sub-engines, reducing resource consumption and improving modular multiplier utilization. Compared to an unoptimized design with two full-round engines, our approach reduces area by 14.8% with just 7.04% extra time overhead and achieves a 14.3 × speedup over FPGA design.
- We develop a Merkle Tree architecture with efficient on-chip cache management and high resource utilization. Large-size Merkle Trees are broken into standard tasks requiring minimal cache for hierarchical processing. By efficiently managing around 200KB of limited on-chip cache, our design supports building large-size Merkle Trees. Merging multiple standard tasks further improves Poseidon Hash engine utilization. AcclMT can achieve up to 1665 × speedup over CPU-based implementation with an average Poseidon Hash utilization of 97.6%.
- Our proposed design natively supports flexible configuration of key parameters for the Poseidon Hash engine and Merkle Tree, enhancing its versatility across a broader range of applications.

II. Background

A. Poseidon Hash Function

Poseidon is a novel arithmetic hash function designed for the ZKP algorithm. While its native computation is more expensive than traditional hash functions like SHA, it offers greater efficiency in circuits, resulting in fewer constraints and reduced complexity for both provers and verifiers. Poseidon Hash employs a sponge/squeeze structure to map arbitrary-length strings over F_p to a fixed-length output, using iterations of the Poseidon permutation. The input, state (S), is divided into rate (R) and capacity (C) bits, with R-sized blocks absorbed into the sponge state. In the squeeze state, part of the state is output as the hash digest. The Poseidon permutation consists of a full round for higher security or a partial round for greater efficiency, with the number of rounds varying by scenario. We present an overview of the Poseidon Hash computation data flow in Fig. 1. In both cases, the algorithm includes three stages: adding round constants (ARC); an S-box layer for nonlinear diffusion using low-degree polynomials; and maximum-distance separable (MDS) matrix multiplication. In partial rounds, all S-box layers except the first are replaced with identity functions.

B. Merkle Tree in zk-STARK

A Merkle Tree (MT) is a tree structure in which each node is constructed using a collision-resistant hash function. It commits to an element sequence, allowing verification of an element's presence. The leaf nodes are obtained by hashing the elements, while non-leaf nodes are derived by concatenating and hashing their child nodes. The root node of the MT serves as a commitment to the sequence, as finding different sequences with the same root is computationally infeasible. Fig. 1 shows a simple schematic of the MT. We focus on binary MT (arity = 2), where each non-leaf node has exactly two child nodes, which is a common structure, though our proposed architecture can be configured to higher arity structures.

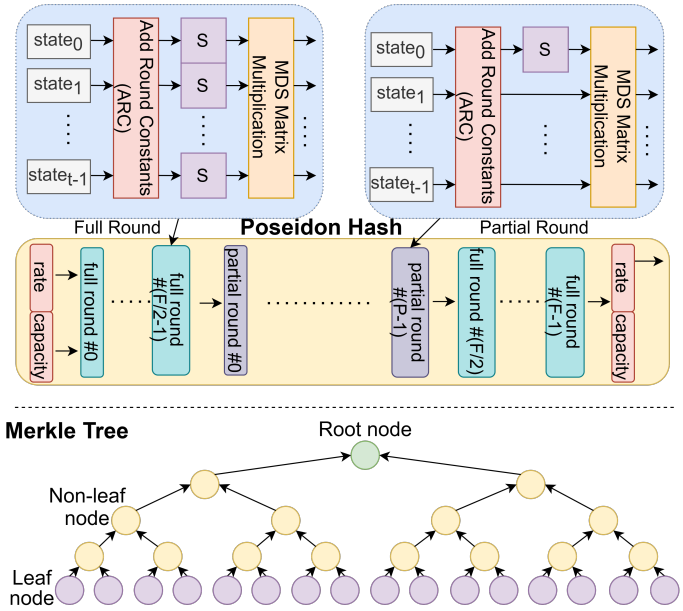


Fig. 1: Flowchart of Poseidon Hash (top) and Merkle Tree (down). For the Poseidon Hash, we assume F full rounds and P partial rounds. The case example features a 4-layer, size-16 Merkle Tree with an arity of 2.

In the zk-STARK protocol, MT is used for polynomial commitments. Fig. 2 outlines the proof generation phase, starting with converting the program into computational integrity (CI) declarations, followed by representing them as polynomials, including trace and constraint polynomials. Multiple constraint polynomials are then combined into a composition polynomial (CP), extended using the low-degree extension (LDE), and evaluated/interpolated via NTT/INTT. The composition and trace polynomials are then committed on the extended domain using MT. After the Deep operation, the polynomial undergoes low-degree testing using the FRI method. During the polynomial folding process, MT is used to commit to the polynomial at each stage. Additionally, this process encompasses querying nodes within the MT and the BCS method [19]. As these topics are not central to this paper, they are not detailed further.

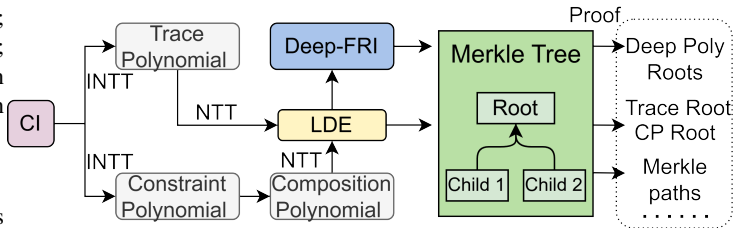


Fig. 2: Flowchart of the Proof Generation Phase in a Typical zk-STARK Protocol.

III. Proposed Design

In this section, we first present the overall architecture of AcclMT. Subsequently, we detail the microarchitecture of the Poseidon Hash engine built with two sub-engines, along with the optimizations of inter-sub-engine data flow. Finally, we present our insights at

the Merkle Tree and optimization strategies for on-chip cache and scheduling mechanisms.

A. Overall Architecture

Our proposed design, AcclMT, consists primarily of two Poseidon Hash engines along with several on-chip caches for specific purposes and a dispatch scheduler. Fig. 3 illustrates the overall architecture of AcclMT. The leaf hash engine, a Poseidon Hash engine, receives the raw element sequence and performs hash computations, storing the leaf nodes in the leaf hash buffer. The root hash engine, also a Poseidon Hash engine, constructs the non-leaf nodes through concatenation and hashing, ultimately deriving the root node of the Merkle Tree. The root hash buffer, scratch pad, and hierarchical intermediate data memory store the intermediate results of the root hash engine or provide inputs to it, with the dispatch scheduler managing the scheduling and distribution of inputs and outputs for the root hash engine. Each of these data storage units has distinct roles, and we will elaborate on the employed design strategies in Sec. III-C.

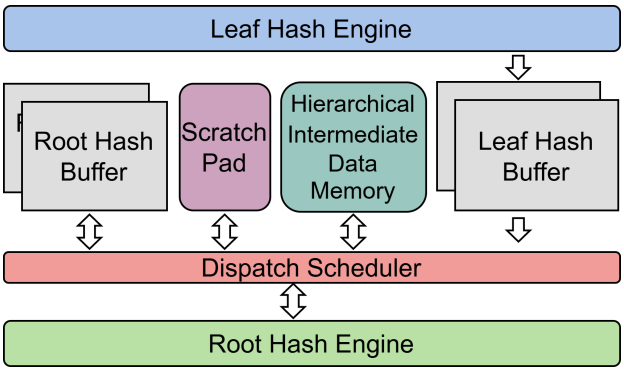


Fig. 3: Overall Architecture of AcclMT.

B. Microarchitecture of Poseidon Hash Engine

We have examined the typical parameters within the Poseidon Hash function, which mainly include the exponent in the S-box (the S-box typically utilizes an exponential function for nonlinear diffusion), the number of states in the Poseidon Hash, and the number of full rounds and partial rounds. Our goal is to develop a highly efficient and flexible Poseidon Hash engine with configurable parameters to enhance versatility.

To achieve this, we first observe the range of these typical parameters and make the following simple insights, focusing on hardware feasibility rather than delving into mathematical principles:

1) The exponent in the S-box is generally greater than or equal to 3, with 3 being almost the most common choice. Some applications may opt for 5 or 7 [3].
2) The number of states in the Poseidon Hash, which is arity + 1, often tends to be multiples of 3. Typical values for arity include 2, 4, 8, and 11 [3], [17].
3) Although the number of full and partial rounds varies across different applications, we observe that the number of partial rounds is generally significantly larger than that of full rounds.

Given these insights, we see great potential for optimizing the Poseidon Hash engine in hardware. Our solutions are as follows: For Insight 1), we design a configurable S-box with three modular multipliers for exponents 3 and 5, and it can also handle exponent 7 with an Initial Interval (II) of 2; higher exponents are not supported due to their rarity. For Insight 2), the Poseidon Hash engine, starting at a state of 3, leverages the vector multiplication and self-adder to support up to the state of 12, accommodating various arities. For Insight 3), we develop a resource-intensive full-round engine for full rounds in full mode and an area-efficient partial-round engine for partial rounds; the full-round engine can also operate in partial mode, enabling both engines to prioritize partial rounds, in line with our observations on round quantities.

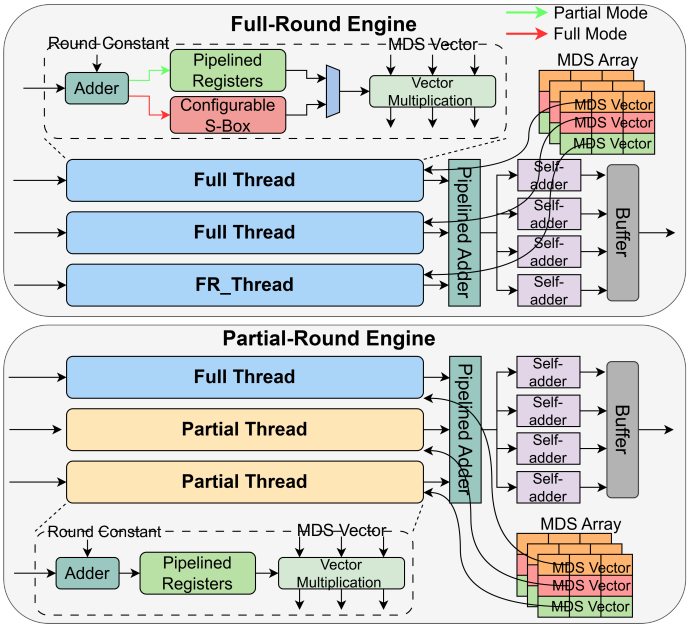


Fig. 4: Microarchitecture of Two Types of Sub-engines in Poseidon Hash Engine. The partial thread utilizes half the modular multiplication resources of the full thread. The full-round engine supports both partial-round and full-round computation, indicated by different colored arrows.

Fig. 4 displays the microarchitectures of our proposed full-round and partial-round engines. The partial-round engine includes two partial threads and one full thread, while the full-round engine comprises three full threads. The partial thread, lacking an S-box, uses half the modular multiplication modules of the full thread. The partial-round engine is dedicated to partial-round computations, while the full-round engine supports full-round computations in full mode and partial-round computations in partial mode. The sub-engine natively supports Poseidon permutation with state = 3. For higher states, an outer product algorithm improves vector-matrix multiplication efficiency. Each thread processes MDS vectors derived from the MDS matrix, and results are aggregated via an adder tree. The self-adder unit handles accumulation and outputs longer vectors in chunks for higher states.

We improve the scheduling mechanism for the two sub-engines to develop the Poseidon Hash engine. It can support arbitrary numbers of full and partial rounds by reusing the full-round engine and partial-round engine through iterative looping. Optimizing the data flow between sub-engines results in higher area efficiency and resource utilization for the Poseidon Hash engine. Fig. 5 compares our Poseidon Hash engine's optimized scheduling with an unoptimized design that uses two full-round engines, as in TRIDENT [17]. The unoptimized design keeps most S-boxes bypassed in partial mode (where partial rounds dominate), leading to considerable modular

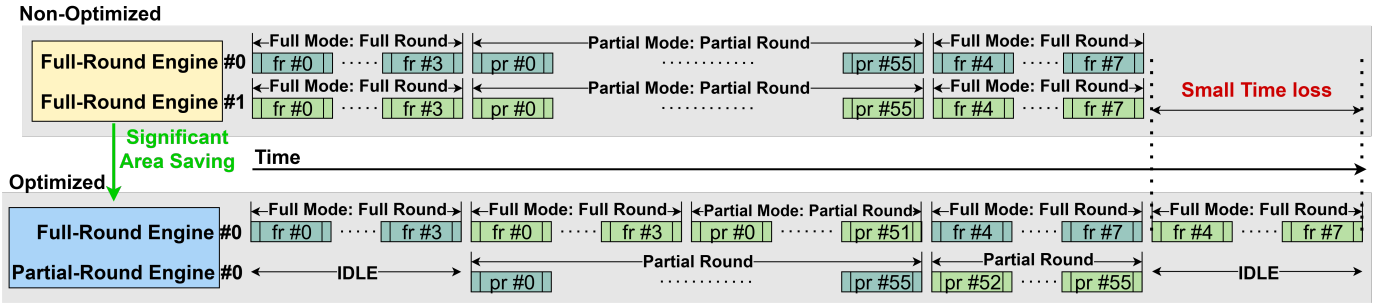


Fig. 5: Optimized Data Flow of Poseidon Hash Engine with Dual Sub-engines. We use typical parameters, with 8 full rounds and 56 partial rounds, as a case. Two different colors are used to illustrate the data flow of Poseidon Hash from different batches. 'pr' stands for partial round, 'fr' represents full round, and the symbol # followed by a number indicates the round sequence number. The partial-round engine consumes only $\frac{2}{3}$ of the modular multiplication resources compared to the full-round engine.

multiplication waste. We find it inefficient to allocate excessive resources to infrequent full rounds. So we assign more full rounds to the full-round engine and allow the more efficient partial-round engine to focus solely on partial rounds, reducing the number of modular multiplication modules.

In our scheduling mechanism, the Poseidon Hash data is divided into two batches. The full-round engine first performs half of the full rounds for both batches, then switches to partial mode for the second batch's partial rounds, while the partial-round engine completes the first batch's partial rounds. The partial-round engine then processes the second batch, with the full-round engine finishing the remaining full rounds in full mode. While the partial-round engine incurs brief idle times, adding minimal latency due to the round count difference, it provides significant area savings, as detailed in Sec. IV-B.

C. Optimization for Merkle Tree

We utilize two Poseidon Hash engines as the Leaf Hash Engine (LHE) and Root Hash Engine (RHE), along with a limited on-chip cache, to construct AcclMT. We observe that during the building of Merkle Tree, two key issues need to be addressed:

1) Merkle Tree sizes are often variable, with larger trees requiring more input and intermediate data. However, relying solely on larger intermediate caches is both inefficient and impractical.
2) The process of building the Merkle Tree involves aggregating multiple child nodes into a parent node. At higher layers of the tree, data distribution becomes sparser, leading to underutilization of the Poseidon Hash engine and reduced computational efficiency.

To address the first challenge, we propose the following solution: as shown in Fig. 6, We use a 128KB Leaf Hash Buffer for LHE-processed values and a 64KB Root Hash Buffer for RHE intermediate results, with a 256-bit data width. These buffers fully support a 4096-size Merkle Tree with arity = 2. To handle larger Merkle Trees in various applications, we decompose them into groups of standard Merkle Trees, each sized at 4096. Instead of computing the entire tree at once, which generates and requires storing gigabits of intermediate data, we pipeline each standard Merkle Tree group, reducing the on-chip cache to about 100KB for a single standard Merkle Tree's intermediate data. The root of the standard Merkle Tree will be stored in an 8.5KB hierarchical intermediate data memory (HIDM). HIDM has two levels: Level 0 stores roots of standard Merkle Trees, which are aggregated when full (corresponding to Merkle Tree layers 13 to 19), with the new root stored in Level 1. When Level 1 is full, a final aggregation is performed for layers 20 to 23, with the final root

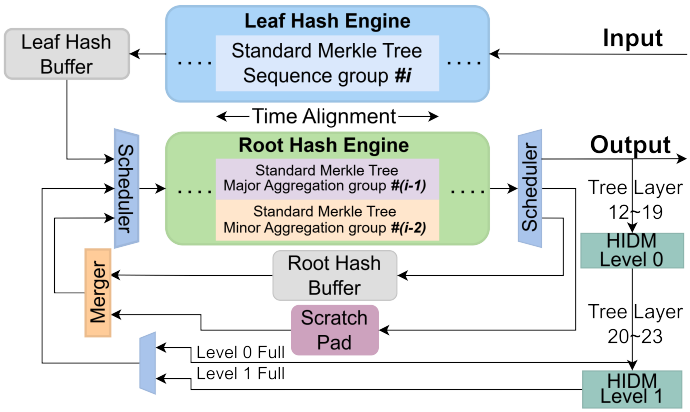


Fig. 6: Optimized Computational Flow of the Merkle Tree in AcclMT. AcclMT can leverage a limited on-chip cache and hierarchical intermediate data memory (HIDM) to efficiently build large-size Merkle Trees through an optimized task scheduling mechanism.

outputted. This hierarchical design supports building the Merkle Trees up to a size of 2^{23}. RHE processes each standard Merkle Tree group by reading data from the leaf hash buffer, and repeatedly writing and accessing intermediate results in the root hash buffer for incremental aggregation. The scheduler manages both RHE's data read and write-back operations.

To address key issue 2, we optimize the data scheduling mechanism within AcclMT. LHE efficiently handles compactly arranged input data, but as Merkle Tree layers deepen, RHE's inputs and outputs become sparser due to data dependencies, which limits its utilization and creates backpressure, reducing LHE's efficiency. We observe that RHE's pipeline utilization drops with each additional Merkle Tree layer; pipeline bubbles in RHE significantly increase by the fifth layer. As shown in Fig. 6, a 4KB scratch pad mitigates this: when aggregating leaf nodes for each standard Merkle Tree, we use root hash buffer as a data buffer, aggregating only up to 128 nodes, a process we call "major aggregation". The final step to reach the root of standard Merkle Tree is referred to as "minor aggregation". Proceeding directly with minor aggregation introduces pipeline bubbles, greatly reducing the efficiency of the Poseidon Hash engine. To improve this, we temporarily store the previous group's intermediate major aggregation results in a scratch pad and use a merger to combine these intermediate data with the next group's major aggregation sequence, effectively filling pipeline bubbles left

by the major aggregation sequence in the RHE. This optimization substantially alleviates sparsity in the RHE pipeline, improving RHE utilization and balancing it with LHE, thereby creating an efficient computational pipeline. Fig. 7 illustrates the optimized data scheduling for the RHE, further boosting the computation efficiency for each standard Merkle Tree. In experiments, the pipeline data flow between RHE and LHE limits minor aggregation to the 11^{th} layer, leaving only the final hash (12^{th} layer). To avoid inefficiency from performing a single hash, both values are stored in HIDM's Level 0, which manages tree layers 12 to 19.

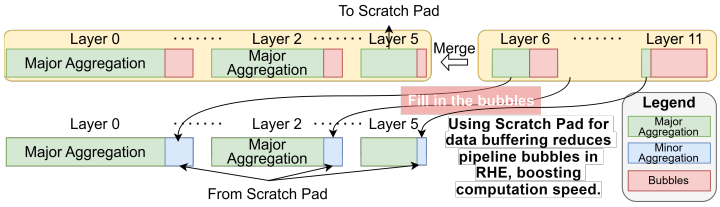


Fig. 7: Optimizing Data Scheduling for the RHE Pipeline. The pipeline combines the Merkle Tree's minor aggregations from the previous group with the major aggregations from the current group, significantly improving RHE utilization.

IV. EVALUATION

A. Experiment Setup & Overhead

In this section, we present the implementation results and performance analysis of AcclMT. AcclMT is implemented in System Verilog HDL and is based on a typical 256-bit data width. The typical parameters in Poseidon Hash, including the S-box exponent, state size, the number of full and partial rounds, and the size of the Merkle Tree, are all configurable at runtime. We synthesize AcclMT using the TSMC 28nm process library and Synopsys Design Compiler, targeting a frequency of 500 MHz. The area report from the synthesis tool indicates AcclMT has a total area of 20.87 mm^2, with the two Poseidon Hash engines taking up 93.7%, and memory occupying about 1.31 mm^2, which is roughly 6.3%. Our Merkle Tree decomposition strategy allows handling several gigabits of data with just nearly 200KB of on-chip cache.

B. Evaluation of the Poseidon Hash Engine

We compare the performance of our proposed Poseidon Hash engine with existing FPGA-based work, TRIDENT [17], and CPU-based work, NEPTUNE [4]. Using the Poseidon Hash configurations from FileCoin [4] as benchmarks, we determine the specific parameters based on Ingonyama's tool [20]. Tab. I details the data, measuring performance by hashes computed per second.

Compared to FPGA-based work, our Poseidon Hash engine offers up to a 14.3 × speedup, achieving a throughput of 14.11M hashes per second. To ensure a fair comparison, excluding frequency differences and focusing solely on clock cycles, the maximum speedup is about 2.85 ×. Notably, TRIDENT uses two units equivalent to our proposed full-round engine. However, during partial round computation, its S-box modules are bypassed and left idle, resulting in lower resource utilization compared to our design. In contrast, We optimize utilization by efficiently distributing tasks between the full-round and partial-round engines, reducing modular multiplication resource consumption by 16.7%. NEPTUNE is an optimized Rust-based open-source implementation of Poseidon Hash for FileCoin. Compared to NEPTUNE, our design offers a speedup of up to 145 ×.

TABLE I: Comparison of Poseidon Hash Computation Speed: AcclMT vs. FPGA and CPU-based Works.

Work		Ours	TRIDENT [17]	NEPTUNE [4]
Platfrom		TSMC 28nm	Xilinx Varium C1100	Intel i7-13700F 32GB
Frequency		500 MHz	100 MHz	2.1 GHz
Hash Rate (hash / sec)	Arity = 2	**14.11M**	0.99M (**14.3 × / 2.85 ×**)[a]	97.00K (**145 ×**)
	Arity = 4	5.93M	-	57.67K (103 ×)
	Arity = 8	3.79M	358K (10.6 × / 2.12 ×)	30.29K (125 ×)
	Arity = 11	2.69M	305K (8.82 × / 1.76 ×)	20.96K (128 ×)

a: The left side of the parentheses indicates the time-based speed up, while the right side represents the clock cycle-based speedup.

Our speedup is optimal at arity = 2, while performance degrades at higher arity levels. This is because, in our Poseidon Hash engine, the two sub-engines support pipelined input and output at arity = 2, resulting in the smallest initiation interval (II). However, at higher arity, the sub-engines are no longer fully pipelined, requiring block processing for input and output, which increases the II and diminishes the overall speedup of the Poseidon Hash engine. Additionally, there is space for further improvements in the scheduling mechanism of the Poseidon Hash engine at higher arity.

Significant Reduction in Area: We conduct experiments to validate that our Poseidon Hash engine, built with a full-round engine and a partial-round engine, achieves higher area efficiency than the baseline with two full-round engines. Based on the four benchmarks from FileCoin shown in Tab. I, our proposal incurs an average time loss of 7.04% relative to the baseline, primarily due to the idling of the partial-round engine, as illustrated in Fig. 5. We also synthesize a baseline design using two full-round engines. Compared to this baseline, our proposed Poseidon Hash engine achieves a 14.8% reduction in area, which closely aligns with our theoretical estimate of 16.7% area reduction. Notably, this area savings is fixed, while the time loss diminishes as the ratio of partial rounds to full rounds increases. When the full round count is 8 and the partial round count is 120 [3], the time loss drops to only 3.55%, nearly negligible, while the reduction of area overhead remains at 14.8%.

C. Evaluation of the Merkle Tree

In Tab. II, we evaluate the performance of AcclMT in building Merkle Trees of various sizes, comparing it to CPU-based implementations with an arity = 2, a common parameter. The CSF '24 [21] uses the libff [23] library in C++ and builds Poseidon Hash-based Merkle Trees with sizes up to 2^{18}. We also utilize the Rust-based library from Dusk-network [22] for another benchmark.

The results show that AcclMT achieves up to a 245 × speedup over CSF '24 and a 1665 × speedup compared to Dusk-network. As the size of the Merkle Tree increases, the speedup of AcclMT becomes more pronounced. The execution time also includes initial data loading and final aggregation in HIDM. When the size of the Merkle Tree is small, the time to build the standard Merkle Tree with AcclMT becomes shorter, and the proportion of time spent on initialization and post-processing increases. As the Merkle Tree size increases, the latency from subsequent data loading is masked by the ping-pong mechanism of data transfer, and the proportion of time spent on data aggregation in the HIDM decreases. At larger sizes, the time for initialization and post-processing can be negligible. Thus, AcclMT's execution time grows sublinearly. Compared to CPU implementations, which exhibit linear growth, AcclMT demonstrates

TABLE II: Comparison of Time (ms) Spent by AcclMT and Other Works for Building Merkle Trees of Various Sizes.

Work	Platform	Merkle Tree Size											
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Ours	TSMC 28nm 500 MHz	0.91	1.22	1.83	3.04	5.45	10.3	19.8	39.0	77.4	154	308	615
CSF'24 [21]	Intel i9-13900KF @ 6.0 GHz	75.8 (83.3 ×)	152 (125 ×)	303 (166 ×)	607 (200 ×)	1213 (223 ×)	2426 (235 ×)	4852 (**245** ×)	-	-	-	-	-
Dusk-network [22]	Intel Xeon Platinum 8358@ 2.6 GHz	498 (547 ×)	1.01 s (828 ×)	2.00 s (1093 ×)	4.00 s (1316 ×)	7.99 s (1466 ×)	16.0 s (1553 ×)	32.0 s (1616 ×)	64.0 s (1641 ×)	128 s (1654 ×)	256 s (1662 ×)	512 s (1662 ×)	1024 s (**1665** ×)

greater advantages in building larger Merkle Trees, making it particularly suited for building large-size Merkle Trees.

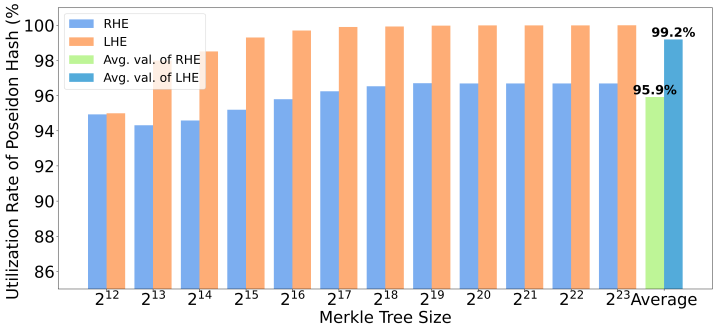


Fig. 8: Utilization Rate of the Leaf Hash Engine (LHE) and Root Hash Engine (RHE) in AcclMT for Building Merkle Trees of Various Sizes.

We also analyze the utilization rates of the LHE and RHE in AcclMT during the building of Merkle Trees of various sizes, as shown in Fig. 8. As the core component of AcclMT, one optimization goal is to improve the Poseidon Hash engine's utilization, which typically correlates with higher computational throughput. The utilization rates of LHE and RHE are quite similar, indicating that data organization for standard Merkle Tree computations is pipelined. Both engines maintain utilization rates above 95%, with RHE reaching 96.7% and LHE nearing 100% as the Merkle Tree size increases. This is expected, as RHE incurs more scheduling overhead, increasing latency. With larger Merkle Trees, both engines' utilization approaches saturation due to the higher computational workload of standard Merkle Tree computations. AcclMT achieves 99.2% and 95.9% average utilization for LHE and RHE, significantly exceeding the pre-optimization average of under 85%.

V. Conclusion

In this paper, we introduce AcclMT, a highly resource-efficient and flexible Poseidon Hash-based Merkle Tree architecture. We implement the Poseidon Hash engine through hardware-software co-optimization and data flow optimization, making it suitable not only for Merkle Trees but also for various purposes in ZKP. We also optimize the scheduling mechanism to ensure efficient acceleration of Merkle Tree computation using limited on-chip cache. The parameters of the Poseidon Hash engine and Merkle Tree size are configurable, offering AcclMT high flexibility for practical use in diverse ZKP scenarios.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant 2023YFB2704600 and in part by the Science and Technology Commission of Shanghai Municipality under Project 24BC3201000. The authors would like to thank the reviewers for their valuable feedback.

References

[1] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Paper 2018/046, 2018. [Online]. Available: https://eprint.iacr.org/2018/046

[2] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*. Berlin, Heidelberg: Springer-Verlag, 2016, p. 305–326.

[3] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for Zero-Knowledge proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 519–535. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi

[4] Filecoin, "Neptune," https://github.com/argumentcomputer/neptune, accessed: 2024-10-31.

[5] Dusk-network, "Poseidon252," https://github.com/dusk-network/Poseidon252, accessed: 2024-10-31.

[6] Polygon, "Plonky2," https://github.com/0xPolygonZero/plonky2, accessed: 2024-10-27.

[7] Kelly Olson, "ZKP MOOC Lecture 16: Hardware Acceleration of ZKP," 2023, accessed: 2024-10-27. [Online]. Available: https://www.youtube.com/watch?v=ez46At3xTjM

[8] K. Aasaraai, D. Beaver, E. Cesena, R. Maganti, N. Stalder, and J. Varela, "FPGA acceleration of multi-scalar multiplication: CycloneMSM," Cryptology ePrint Archive, Paper 2022/1396, 2022. [Online]. Available: https://eprint.iacr.org/2022/1396

[9] C. Liu, H. Zhou, L. Yang, J. Xu, P. Dai, and F. Yang, "Gypsophila: A scalable and bandwidth-optimized multi-scalar multiplication architecture," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3658259

[10] Y. Zhang, S. Wang, X. Zhang, J. Dong, X. Mao, F. Long, C. Wang, D. Zhou, M. Gao, and G. Sun, "Pipezk: Accelerating zero-knowledge proof with a pipelined architecture," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 416–428.

[11] T. Lu, C. Wei, R. Yu, C. Chen, W. Fang, L. Wang, Z. Wang, and W. Chen, "cuZK: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on GPUs," Cryptology ePrint Archive, Paper 2022/1321, 2022. [Online]. Available: https://eprint.iacr.org/2022/1321

[12] C. Wang and M. Gao, "Sam: A scalable accelerator for number theoretic transform using multi-dimensional decomposition," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[13] O. Shlomovits, "Revisiting paradigm hardware acceleration for zero knowledge proofs," https://www.ingonyama.com/blog/revisiting-paradigm-hardware-acceleration-for-zero-knowledge-proofs, 2023, accessed: 2024-10-29.

[14] H. Zhou, C. Liu, L. Yang, L. Shang, and F. Yang, "Rezk: A highly reconfigurable accelerator for zero-knowledge proof," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2024.

[15] ZPRIZE, "Accelerating the poseidon hash function," https://www.zprize.io/prizes/accelerating-the-poseidon-hash-function, accessed: 2024-10-27.
[16] Ingonyama, "Icicle," https://github.com/ingonyama-zk/icicle/tree/main, 2023, accessed: 2024-11-5.
[17] DatenLord, "Trident: Open-source project for high-performance storage solutions," https://github.com/datenlord/TRIDENT, accessed: 2024-10-27.
[18] Irreducible, "Poseidon merkle trees in hardware," https://www.irreducible.com/posts/poseidon-merkle-trees-in-hardware, 2023, accessed: 2024-10-27.
[19] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," Cryptology ePrint Archive, Paper 2016/116, 2016. [Online]. Available: https://eprint.iacr.org/2016/116
[20] Ingonyama, "poseidon-hash," https://github.com/ingonyama-zk/poseidon-hash, accessed: 2024-11-1.
[21] E. Andreeva, R. Bhattacharyya, A. Roy, and S. Trevisani, "On efficient and secure compression modes for arithmetization-oriented hashing," Cryptology ePrint Archive, Paper 2024/047, 2024. [Online]. Available: https://eprint.iacr.org/2024/047
[22] D. Network, "dusk-poseidon-merkle," https://github.com/dusk-network/dusk-poseidon-merkle, 2023, accessed: 2024-10-27.
[23] scipr lab, "libff," https://github.com/scipr-lab/libff, accessed: 2024-11-3.

ChipletEM: Physics-Based 2.5D and 3D Chiplet Heterogeneous Integration Electromigration Signoff Tool Using Coupled Stress and Thermal Simulation

Zeyu Sun[a,b,c],Weijie Tong[a,b,c], Xiaoning Ma[a,b,c], He Cao[a,b,c], Jianyun Liu[a,b,c], Zhiqiang Li[a,b,c,*],Qinzhi Xu[a,b,c,*]
[a]State Key Lab of Fabrication Technologies for Integrated Circuits, Beijing 100029, China
[b]Institute of Microelectronics of the Chinese Academy of Sciences, Beijing 100029, China
[c]University of Chinese Academy of Sciences, Beijing 100049, China
[*]Corresponding author E-mail: lizhiqiang@ime.ac.cn xuqinzhi@ime.ac.cn

ABSTRACT

A review of recent studies on up-to-date IC shows that electromigration (EM) has become one of the major challenges for 2.5D and 3D chiplet heterogeneous integration (CHI) systems. However, most existing researches on EM are focusing on 2D power delivery network without taking Through Silicon Via (TSV) and non-uniformly thermal distribution condition between dies into consideration. To address this problem, this article proposes a novel EM simulation tool ChipletEM for 2.5D and 3D CHI systems. A finite volume method (FVM) based electrical-thermal co-simulation model is employed to get initial temperature and current density inside TSV. And a finite difference time domain (FDTD) solver is used for hydrostatic stress simulation for both nucleation and post-voiding phases. Thermal migration (TM) effect is also considered in the solver. An analytical TSV thermal solver is employed for temperature distribution simulation and thermal dependent current simulation. The FDTD EM solver and TSV thermal solver are coupled together at each time step so that the interaction among EM stress, thermal stress, void growth, resistance change, IR drop and Joule heating effects can be simulated within a single simulation framework. Simulation results show that compared with Finite Element Method (FEM) tool, average error is 0.61% in nucleation phase and 2.4% in growth phase. And the error of proposed method is reduced from 22.22% to 5.24% compared with state of art atomic flux divergence (AFD) method.

KEYWORDS

Chiplet integration systems; TSV; Reliability; Electromigration; Thermal-electrical co-simulation

1 INTRODUCTION

With emergence of high performance AI algorithms, Through Silicon Via (TSV) based 2.5D and 3D chiplet heterogeneous integration (CHI) systems have become one of the most promising technology to achieve the requirement of excessive data communication [1–3]. The architecture of a 2.5D CHI systems is shown in Fig. 1.

However, due to the high current density and non-uniformly distributed high temperature on TSV, a number of reliability issues

This work was supported in part by the National Natural Science Foundation of China under Grant 92373113, in part by the National Key Research and Development Program of China (Grant 2024YFB4405401), in part by the Strategic Priority Research Program (A) of Chinese Academy of Sciences (CAS) under Grant XDA0330401, and in part by the CAS Youth Interdisciplinary Team under Grant JCTD-2022-07. (Corresponding authors: Qinzhi Xu; Zhiqiang Li.)

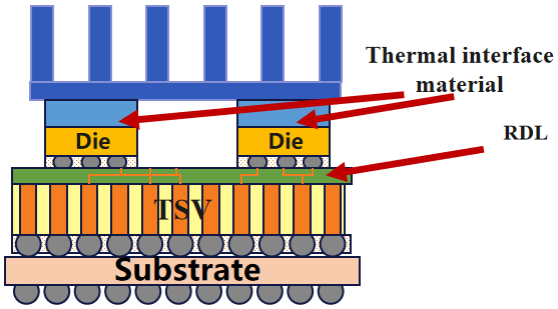


Figure 1: Architecture of a 2.5-D CHI system.

,including electromigration (EM), Joule heating, thermomigration (TM), time-dependent-dielectric-breakdown(TDDB), have become the main threaten to the advanced CHI systems[4–6]. Among these reliability issues, EM is one of the major reliability concerns in the back-end-of-line (BEOL) interconnects, especially in the advanced sub-10nm 2.5D and 3D CHI systems. According to the ITRS 2015 report, EM lifetime will be significantly reduced due to increasing current density and shirking size of net cross sections[7]. Therefore, a fast and accurate EM model is essential for advanced 2.5D and 3D CHI systems. However, existing models cannot meet the requirement of efficiency and accuracy sign-off. Traditional mean-time-to-failure (MTTF) models such as black's equation [8] and blech's product [9] are empirical fitting-based models, which are not scalable and robust enough for the modern 2.5D and 3D CHI systems. The models based on the experiment data of simple one-segment metal line cannot get an accurate EM failure prediction of complicated multi-segment wires and TSVs. Besides, these models are over conservative, bringing unnecessary constraint and increasing the difficulty of IC design.

In order to mitigate the disadvantages of traditional empirical models, a number of physics-based models are proposed [4, 10–14], which considers general interconnect for the 2D interconnect. These models are compact and can solve the stress based partial different equation (PDE) effectively. And the work in [10] has considered the TM and Joule heating effect. However, these methods focus on SoC systems and only consider the Joule heating effect of interconnects and local thermal distribution. As presented by [15–18], global thermal effect is critical and can be one of the dominant factors for EM. In the advanced 2.5D and 3D CHI systems, due to unique

topology and non-uniformly distributed temperature of TSV, a specific EM analysis tool needs to be developed.

In order to address aforementioned issues, this paper proposes a novel physics-based stress and thermal coupling analysis tool for TSV based 2.5D and 3D ICs. Specifically, the contributions are summarized up as follows:

•A finite difference time domain (FDTD) based EM-TM model, including both nucleation and post-voiding phases, is applied to TSV. Hydro-static stress is calculated by the extended Korhonen's equation combined with both EM and TM flux.

•A global thermal model of CHI system is employed in order to get the accurate temperature distribution. A realistic CHI model ,including heat sink, is implemented. And a finite volume method (FVM) is used for the global electrical-thermal co-simulation for high simulation efficiency while retaining the conservation principles inside thermal conduction equation.

•A coupled EM-TM and Joule heating simulation is employed, so that the simulator can consider the interplay among stress, void growth, resistance change and Joule heating effect in a single simulation framework.

•Simulation result shows that the accuracy of the proposed tool is very high. Compared with commercial finite element method (FEM) based tool, average error is 0.61% in nucleation phase and 2.4% in growth phase. And the error of the proposed method is reduced from 22.22% to 5.24% compared with state of art atomic flux divergence (AFD) method. The error rate is reduced by 76.4%.

•The proposed tool is applied to a 2.5D CHI system. And the simulation result shows that EM issue is more critical when power consumption is high. It can also be observed that in the early stage the dominant factor for EM is temperature while in the late stage the dominant factor is current density.

This paper is organized as follows: Section 2 reviews state of art EM models. Section 3 presents the background of EM. Section 4 introduces the proposed EM thermal co-simulation method. Section 5 demonstrates numerical results obtained from the proposed method. Section 6 concludes this paper.

2 RELATED WORKS

A few EM analysis for TSV based 2.5D and 3D ICs have been proposed recently[4, 13, 19–21]. Specifically, Frank *et al.*[19] and Pak *et al.*[4] put forward AFD based approaches for the TSV. They investigate the mechanism of the EM failure in TSV, showing that the EM void is in the adjacent metal level. Pak's model points out that the resistance change is related to the current density, temperature, cross section area of the TSV and redistribution layers (RDL) interconnect thickness. And a recent AFD based study [13] points out that the EM void is also possible to be nucleated inside the TSV instead of the the junction with RDL interconnects when TSV diameter is small and its aspect ratio is large. Wang's work[20] has considered recovery effect by converting the time variant current into an equivalent square waveform. A response surface methodology based algorithm taking current density, duty cycle and temperature as input parameter is proposed in that work. Nevertheless, the physics mechanisms of EM are not well studied by these methods since internal stress evolution inside the TSV is not counted. Thus the models, not scalable and too conservative, will lead to design redundancy. Besides, as one of the key factors of EM failure, thermal effect and Joule heating need to be taken into consideration. Liu's work[21] studies the relationship between self heating effect and EM of the TSV. The experiment result shows that increased temperature has a positive feedback on the EM effect and will cause thermal run-away effect due to the small cross section area between TSV and RDL. However, this work only focuses on the local Joule heating of TSV lacking the consideration of global thermal effect.

To address the aforementioned issues, a novel coupled EM and thermal simulation tool is proposed in the work, employing Korhonen's equation based physics models to achieve fast and accurate EM lifetime estimation. Also the proposed method has taken global thermal effect of actual complex structure with heat sink of 2.5D and 3D CHI systems into consideration.

3 A REVIEW OF STRESS BASED THREE PHASE EM MODEL

EM is a physical phenomenon of the migration of metal atoms along the direction of the applied electrical field due to momentum exchange between electrons and atoms. During the migration process, hydro-static stress, as well as the stress gradient, generated inside the embedded metal wire due to momentum exchange between lattice atoms will act as a counter-force for atomic migration along the metal line. Before the void nucleated, the stress distribution in the interconnect can be described as follows:

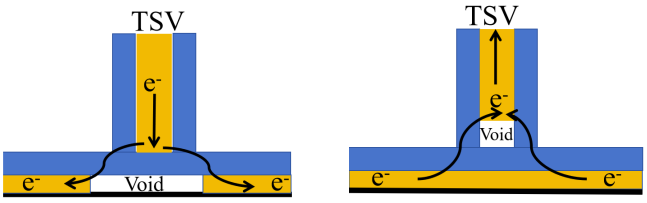
$$\begin{aligned}
&\frac{\partial \sigma}{\partial t} = \frac{\partial}{\partial x}\left[\frac{D_a B\Omega}{k_B T\,(x)}\left(\frac{\partial \sigma}{\partial x} - \frac{eZ\rho j}{\Omega} - \frac{Q}{\Omega T}\frac{\partial T}{\partial x}\right)\right] \\
&BC: \frac{D_a B\Omega}{k_B T\,(x_b)}\left(\left.\frac{\partial \sigma}{\partial x}\right|_{x=x_b} - \frac{eZ\rho j}{\Omega} - \frac{Q}{\Omega T}\frac{\partial T}{\partial x}\right) = 0 \quad 0 < t < t_{nuc} \\
&IC: \sigma\,(x,0) = \sigma_T
\end{aligned} \tag{1}$$

Here, σ is the EM hydrostatic stress, x is the coordinate along the segment length, t is time, $T(x)$ is the temperature at corresponding position, D_a is the effective atomic diffusivity, eZ is the effective charge of migrating atoms, B is the effective bulk elasticity modulus, Ω is the atomic lattice volume, ρ is the copper resistivity, j is the current density, k_B is the Boltzmann constant, T is the temperature, σ_T is initial thermal stress, and x_b is coordinate of boundary node.

When the stress at the cathode exceeds the critical level at t_{nuc}, a void will be nucleated and start growth which is called incubation phase. The normal component of stress at the void boundary is typically zero. Stress gradient between the zero stress void surface and the surrounding is shown as follows:

$$\nabla\sigma(x,t) = \frac{\sigma(x+\delta,t)}{\delta} \qquad on \quad \partial\Omega_L \cap \Gamma_{void} \tag{2}$$

Here δ is effective thickness of the copper-void boundary, Ω_L is the volume of the remaining interconnect wire, Γ_{void} denotes the void boundary, which can be a union of several discrete voids, the $\nabla\sigma(x,t)$ essentially serves as a force to move the void boundary toward the atomic flux direction. The void volume is determined by the stress distribution on the remaining section of the wire and



(a) Void nucleated in RDL junction (b) Void nucleated inside TSV

Figure 2: Location of EM void in TSV. (2a) is the condition that the void nucleated in the RDL layer and (2b) is the condition that the void nucleated inside TSV

it can be calculated by the following atom conservation equation:

$$V_{void}(t) = -\int_{\Omega_L} \frac{\sigma(t)}{B} dV \tag{3}$$

Here $V_{void}(t)$ is time dependent void volume. When the void volume is larger than critical volume (V_{crit}) which mainly depends on the cross section of the wire, the resistance starts change. That phase is called growth phase.

For the TSV, the void location depends on the directions of electron flow. Based on the experiment result in [13], the location of the void can be nucleated at the junction of TSV and RDL line or inside the TSV with different current directions. Fig. 2a shows the condition that the void nucleated at the junction with the electron flow from TSV to RDL line. The resistance can be calculated with the following equation:

$$\Delta R = \frac{\rho_B}{2 \times \pi \times t_B} \times ln\left(\frac{r_{void}}{r_{TSV}}\right) \tag{4}$$

Here ρ_B is resistivity of the barrier material(Ta/TaN). t_B is thickness of the barrier. r_{void} is radius of void and r_{TSV} is radius of TSV. And Fig. 2b shows the condition that the void nucleated inside TSV with the electron flow from RDL line to TSV. The resistance can be calculated with the following equation:

$$\Delta R = \frac{\rho_B V_{void}}{r_{TSV}^2\left((r_{TSV} + t_B)^2 - r_{TSV}^2\right)} \tag{5}$$

The work focuses on the condition that the void nucleated inside TSV shown in Fig. 2b. And the other conditions will be studied in the future work.

4 METHODOLOGY

4.1 The New Coupled TSV EM Check Flow

In this part, the flow of proposed ChipletEM tool is presented as shown in Fig. 3. The whole flow mainly consists of three major parts: *CHI system electrical-thermal co-simulation model, Fast EM solver* and *TSV thermal solver*. CHI system electrical-thermal co-simulation model is employed in order to get the initial temperature distribution in the CHI systems and current density of each TSV. Fast EM solver is implemented with a FDTD based method. Thermal migration effect is also considered in the model. TSV thermal solver use an analytical model to get temperature distribution and temperature dependent current density inside TSV. Joule heating effect and temperature dependent electrical conductivity are combined together. EM and TSV thermal solver are coupled together. EM induced resistance change will change the Joule heating condition so that the corresponding temperature distribution is also changed. And the time dependent temperature acts as one of the dominant factors in the EM simulation. The time dependent IR drop of CHI systems considering global thermal effect can be demonstrated to the users for further design optimization. The following section will present aforementioned steps in details.

4.2 CHI electrical-thermal co-simulation model

In order to get the thermal distribution and the current density in TSV, an efficient electrical-thermal co-simulation model [18] for CHI systems is employed in this work. Due to the complexity of CHI systems, detailed circuit level based power analysis is not time efficient. Thus, the employed model abstracts the parameters and maps the circuit level thermal model to system level. The Joule heating power is calculated as follows:

$$\begin{aligned} \nabla \cdot [\kappa(x,y,z)\nabla T(x,y,z)] &= -P_{total} \\ P_{Joule} &= \frac{1}{\rho(x,y,z,T)}|\nabla\varphi(x,y,z)|^2 \\ P_{total} &= P_{init} + P_{Joule} \end{aligned} \tag{6}$$

Here κ is thermal conductivity of the solid medium, P is the power consumed by interconnect, x, y, z are the coordinate of CHI system, T is temperature, ρ is the temperature dependent resistivity and φ is the potential distribution, P_{init} is the initial power consumption without Joule heating.

As can be seen, P_{Joule} is affected by temperature dependent resistivity ρ. Meanwhile, P_{Joule} is a part of heat source leading to temperature increasing.

And the temperature dependent resistivity of RDL, bump array, TSV and other interconnects are estimated with the equivalent model to improve the computational efficiency [15–17, 22]. The equivalent model can reduce the impact of complex structure and the composite material of these interconnects and get an estimated resistivity efficiently which is close to the actual resistivity. And the model is discretized by FVM method.

With the presented model, the global thermal effect of CHI system considering convective heat transfer effects and Joule heating effects can be estimated. Individual TSV initial temperature can be obtained from the global thermal distribution.

4.3 Coupled Thermal and Electromigraiton Multi-Physics Simulation

In order to get accurate EM stress for IR-drop analysis, a coupled thermal and EM simulation method is proposed. Temperature is one of the dominant conditions for the EM failure mechanism. It contributes to both stress evolution and TM effect. Meanwhile current induced Joule heating is one of the major heat sources of interconnects. So it is essential to do coupled simulation to get an accurate and EM stress evolution process. In order to improve the accuracy of the EM stress estimation in the full-chip power grid level, we have to consider multi-physics effects and their interaction.

For the Korhonen's equation based EM model from Eq. 1, a finite difference based model is employed for detailed stress analysis over time. Employed method follows the similar work [11, 12, 14]

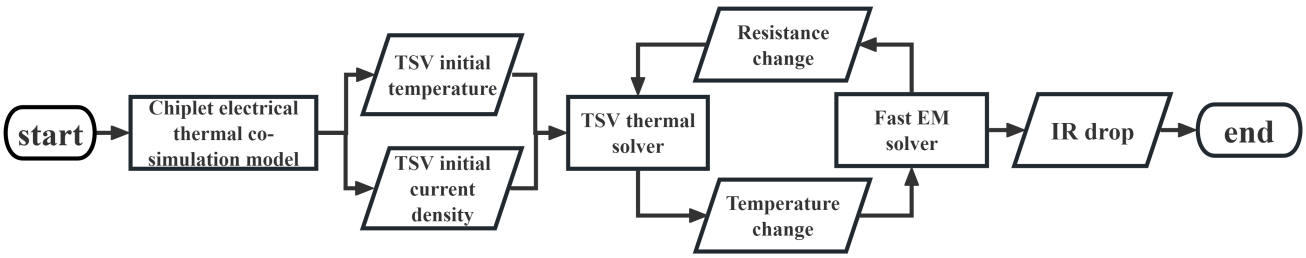


Figure 3: Simulation flow for proposed ChipletEM tool

without the model order reduction feature as shown in the following equation:

$$C\dot{\sigma}(t) = A\sigma(t) + PI(t) - D \tag{7}$$

Here C and A are $n \times n$ matrices. P is the np input matrix. D is $n \times 1$ constant matrix corresponding to TM effect, n is the number of discretization and p is the size of driving current density sources $I(t)$.

For the TSV thermal model, a close form equation from [23] considering Joule heating induced current. And the Joule heating is calculated by the following TSV thermal model in Eq. 8.

The equation is shown as follows:

$$T_{TSV}(x) = \frac{\alpha^2 T_{sub} + \theta}{\lambda^2} + \frac{(\lambda^2 - \alpha^2)T_{sub} - \theta}{\lambda^2(1 + e^{\lambda H})} \times \left[e^{\lambda(H-x)} + e^{\lambda x} \right] \tag{8}$$

Here the T_{sub} is the temperature of substrate, H is height of TSV, x is coordinate of the TSV. And the λ^2, α^2 and θ is presented as follows:

$$\lambda^2 = \frac{1}{k_m}\left[\frac{2k^*_{ins}}{t_{ins} r_{Cu}} - \frac{I^2(x)\rho_m \beta}{\pi^2 r^4_{Cu}}\right]$$

$$\alpha^2 = \frac{2k^*_{ins}}{k_m t_{ins} r_{Cu}}$$

$$\theta = \frac{I^2(x)\rho_m \left[1 - \beta T_{ref}\right]}{k_m \pi^2 r^4_{Cu}}$$

$$I(x) = \frac{V(0) - V(H)}{\int_0^H R_E(x)dx}$$

$$\int_0^H R_E(x)dx = R_0(1 - \beta T_{ref})H + R_0\beta\left[\frac{\alpha^2 T_{sub} + \theta}{\lambda^2} \cdot H + \frac{(\lambda^2 - \alpha^2)T_{sub} - \theta}{\lambda^3(1 + e^{\lambda H})} + \frac{(\lambda^2 - \alpha^2)T_{sub} - \theta}{\lambda^3(1 + e^{\lambda H})} \cdot 2(e^{\lambda H} - 1)\right] + R_{EM} \tag{9}$$

Here $I(x)$ is current, $\int_0^H R_E(x)dx$ is temperature dependent resistance, k_m is thermal conductivity of copper, R_0 is resistance under reference temperature T_{ref}, β is temperature coefficient of copper, ρ_m is resistivity of copper, r_{Cu} is ratio of copper inside TSV, t_{ins} is thickness of insulator, T_{sub} is temperature of substrate, $V(0)$ is the bottom voltage of TSV, $V(H)$ is the top voltage of TSV and R_{EM} is the EM induced resistance.

As can be seen in equation 9, $I(x)$ and $\int_0^H R_E(x)dx$ are coupled together to get the steady state solution of corresponding working condition. And the resistance changed due to EM can be calculated by Eq. 5 and 7. The resistance is updated with the change of EM conditions.

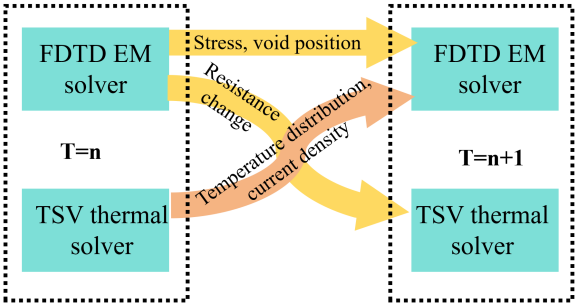


Figure 4: Block diagram of coupled FDTD EM and TSV Joule heating simulator.

As can be seen, Eq. 7, 5 and 9 are coupled together as shown in Fig. 4. Since thermal solver is static, EM simulation time step is used as the coupled simulation time step here. FDTD EM solver passes resistance change to TSV thermal solver and gets temperature distribution and Joule heating induced current density from it. Accurate EM and thermal information can be achieved with the coupled simulation at each time step.

5 NUMERICAL RESULTS AND DISCUSSION

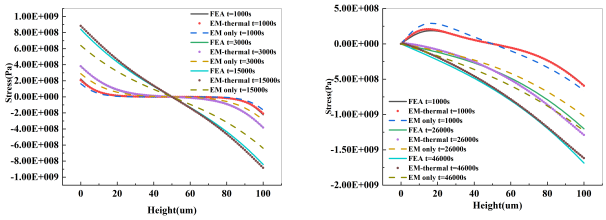
This section presents numerical experiments of the proposed ChipletEM tool which is carried out on a Windows-based simulation platform with an 3.70GHz AMD Ryzen5 5600X processor which has 6 cores. FDTD EM solver and TSV thermal solver are prototyped in MATLAB and C, respectively. The experiment is performed on a 2.5D CHI system published in [18]. This section will first validate the proposed tool with a state-of-art EM model and experiment result; then present global chiplet scale resistance change result.

5.1 Accuracy comparison against existing EM analysis methods and experiment results on single TSV

This part first compares the result of the proposed method and FEA tool, then compares the proposed method with state-of-art EM models and experiment result.

Table 1: Failure rate of high and low power mode

	Nucleation phase	Growth phase
EM-thermal average error	0.61%	2.4%
EM-thermal max error	2.62%	6.67%
EM only average error	3.48%	10.03%
EM only max error	12%	26.79%
TM contribution	11.4%	15.83%



(a) Nucleation phase (b) Growth phase

Figure 5: Coupled EM-thermal simulation result

5.1.1 Validation with FEM tool. Fig. 5 shows the comparison result between FEA tool, EM-thermal co-simulation method and EM only simulation method in nucleation phase and growth phase. Experiment is performed on a single TSV with 100*um* height. As can be seen in Table 1, the average error is 0.61% in nucleation and 2.4% in growth phase. And if only EM is considered without coupled thermal simulator, average error is 3.48% for nucleation phase and 10.43% for growth phase. Max error can reduce from 26.79% to 6.67% with the coupled EM-thermal simulation. It draws a conclusion that Joule heating is one of the key factor for EM and EM-thermal co-simulation can significantly improve the accuracy.

Besides, Table 1 also shows the stress contribution of TM effect. Simulation result shows that TM effect contributes 11.4% in the nucleation phase stress evolution and 15.83% in growth phase stress evolution. And this agrees with the conclusion from experiment in [13] saying that TM effect is relatively weak.

5.1.2 Validation with experiment data. This part validates the simulation result with published experiment data from [13]. Based on the experiment result, $\frac{\Delta R}{R}$ reaches 20% at 32.4 hours with $200°C$ and $1.5MA/cm^2$ current density. Fig. 6 compares the simulation result of the proposed method and AFD method from [13] with real EM experiment result.

It can be seen that in the experiment the TSV reaches 20% resistance change in 32.5 hours. And the simulation result of proposed method is 30.7 hours and AFD method is 25.2 hours. The error of proposed method is 5.24% and the error of AFD method is 22.22%. The error rate is reduced by 76.4%.

5.2 Simulation results on the CHI System

This part demonstrates the simulation results on a 2.5D CHI system which has nine chiplets. The parameters of the system is shown in Table 2 and its layout of the system is in Fig. 7.

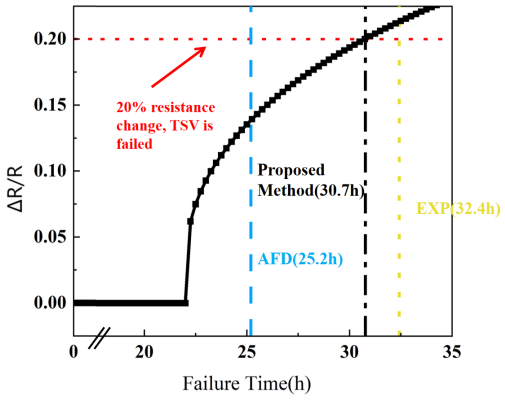


Figure 6: Resistance change trend of TSV

Table 2: Parameters of the CHI system

Name	Number	Size (mm)
Chiplet	9	$8 \times 8 \times 0.15$
Heat Sink	1	$40 \times 40 \times 0.18$
TSV	1600	Diameter = 0.01 Height = 0.1
MicroBump	2500	/

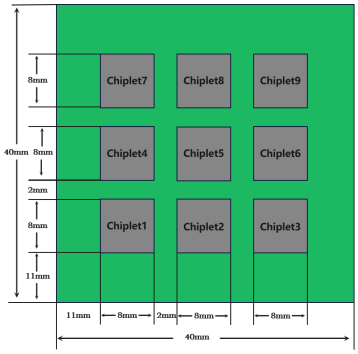
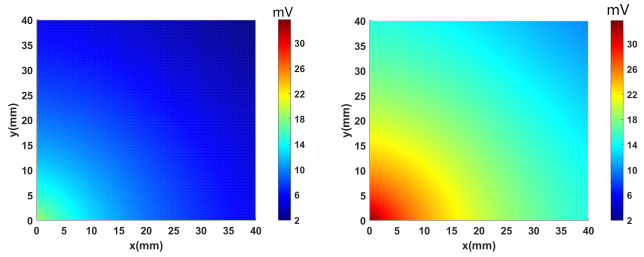


Figure 7: Layout of CHI system

There are low and high working modes for the CHI system. The voltage distribution of both modes are shown in Fig. 8. And temperature distribution results from CHI system electrical-thermal co-simulation model is shown in Fig. 9. It can be seen that the left corner of the chip has high voltage and the medium of the chip has high temperature.

The resistance change distribution of the CHI system of low and high power modes is shown in Fig. 10 and 11, in which the EM issue in the high power mode is more critical than that in the low power mode. And we can see current density and thermal are both critical issues for the EM. It can be observed that in the early stage the high temperature part has more critical EM issue. In the late stage EM issue is more critical in the high current density part.

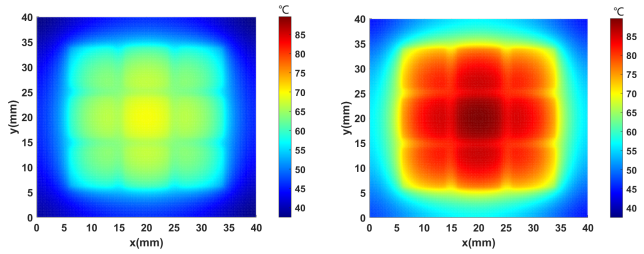
Table 3 displays that the percentage of failed TSVs is increasing over time. In the low power mode, there is no TSV failed in the first year since most TSVs are still in nucleation phase. And 47%



(a) Voltage distribution for low power mode

(b) Voltage distribution for high power mode

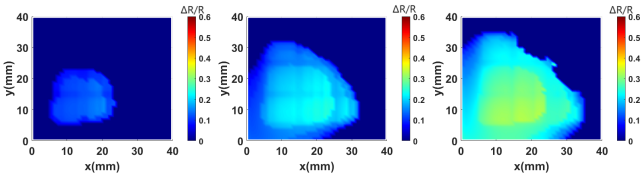
Figure 8: Voltage distribution of CHI system



(a) Temperature distribution for low power mode

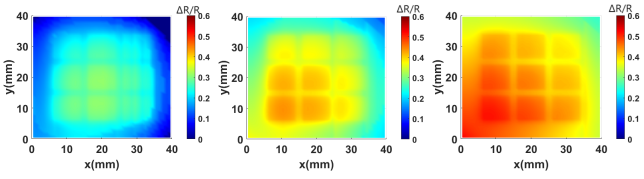
(b) Temperature distribution for high power mode

Figure 9: Temperature distribution of CHI system



(a) 1 year (b) 3 years (c) 6 years

Figure 10: Resistance change distribution of low power mode in 1, 3 and 6 years



(a) 1 year (b) 3 years (c) 6 years

Figure 11: Resistance change distribution of high power mode in 1, 3 and 6 years

TSVs failed in 6 years. EM issue is more critical in high power mode. 54.25% TSVs failed in first year and all the TSVs failed in 6 years.

Table 3: Failure rate of high and low power mode

	6 month	1 year	3 years	6 years
Low power	0%	0%	16.75%	47%
High power	26.19%	54.25%	91.5%	100%

6 CONCLUSION

This paper has proposed a novel EM simulation tool ChipletEM for 2.5D and 3D CHI systems. A FVM based electrical-thermal co-simulation model is employed to get initial temperature and current density inside TSV. A FDTD EM solver and a compact TSV thermal solver is employed for coupled simulation for interaction among EM stress, thermal stress, void growth, resistance change, IR drop and Joule heating effects. Accuracy of the proposed tool is validated with FEM tool and published experiment data. Compared with state of art AFD method the error rate is reduced by 76.4%. In the future the EM failure between RDL and TSV junction will also be taken into consideration.

REFERENCES

[1] Z. Wang, J. Sun, A. Goksoy, S. K. Mandal, Y. Liu, J.-S. Seo, C. Chakrabarti, U. Y. Ogras, V. Chhabria, J. Zhang, and Y. Cao, "Exploiting 2.5d/3d heterogeneous integration for ai computing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 758–764, 2024.

[2] G. Krishnan, S. K. Mandal, M. Pannala, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, "Siam: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 20, sep 2021.

[3] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, (New York, NY, USA), p. 14–27, Association for Computing Machinery, 2019.

[4] J. Pak, S. K. Lim, and D. Z. Pan, "Electromigration study for multiscale power/ground vias in tsv-based 3-d ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1873–1885, 2014.

[5] A. Jain, R. E. Jones, R. Chatterjee, and S. Pozder, "Analytical and numerical modeling of the thermal performance of three-dimensional integrated circuits," *IEEE Transactions on Components and Packaging Technologies*, vol. 33, no. 1, pp. 56–63, 2010.

[6] H.-J. Choi, S.-M. Choi, M.-S. Yeo, S.-D. Cho, D.-C. Baek, and J. Park, "An experimental study on the tsv reliability: Electromigration (em) and time dependant dielectric breakdown (tddb)," in *2012 IEEE International Interconnect Technology Conference*, pp. 1–3, 2012.

[7] ITRS, "International technology roadmap for semiconductors (itrs)," Mar. 2015.

[8] J. Black, "Electromigration—a brief survey and some recent results," *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, 1969.

[9] I. A. Blech, "Electromigration in thin aluminum films on titanium nitride," *Journal of Applied Physics*, vol. 47, pp. 1203–1208, 04 1976.

[10] L. Chen, S. X.-D. Tan, Z. Sun, S. Peng, M. Tang, and J. Mao, "A fast semi-analytic approach for combined electromigration and thermomigration analysis for general multisegment interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 350–363, 2021.

[11] Z. Sun, S. Yu, H. Zhou, Y. Liu, and S. X.-D. Tan, "Emspice: Physics-based electromigration check using coupled electronic and stress simulation," *IEEE Transactions on Device and Materials Reliability*, vol. 20, no. 2, pp. 376–389, 2020.

[12] C. Cook, Z. Sun, E. Demircan, M. D. Shroff, and S. X.-D. Tan, "Fast electromigration stress evolution analysis for interconnect trees using krylov subspace method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 969–980, 2018.

[13] Y. Zhang, W. Tian, H. Wang, X. Zhang, W. Shao, B. Zhou, Y. Shi, W. Gong, R. Deng, J. Zhao, P. Wang, and Y. Feng, "Electrothermal reliability analysis of electromigration in 3-d tsv-rdl interconnects," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 14, no. 1, pp. 157–165, 2024.

[14] S. Lamichhane, M. Kavousi, and S. X.-D. Tan, "Emspice 2.0: Multiphysics electromigration analysis tool for beyond moore ics," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 571–576, 2024.

[15] C. Nie, Q. Xu, and L. Chen, "Equivalent thermal model of through silicon via and bump for advanced packaging of integrated circuits," *Microelectronics and reliability*, vol. 137, no. Oct., pp. 178–189, 2022.

[16] C. Nie, Q. Xu, C. Wang, H. Cao, J. Liu, and Z. Li, "Efficient transient thermal analysis of chiplet heterogeneous integration," *Applied thermal engineering: Design, processes, equipment, economics*, 2023.

[17] C. Wang, Q. Xu, C. Nie, H. Cao, J. Liu, and Z. Li, "An efficient thermal model of chiplet heterogeneous integration system for steady-state temperature prediction," *Microelectronics Reliability*, 2023.

[18] X. Ma, Q. Xu, C. Wang, H. Cao, J. Liu, D. Zhang, and Z. Li, "An electrical–thermal co-simulation model of chiplet heterogeneous integration systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–13, 2024.

[19] T. Frank, S. Moreau, C. Chappaz, L. Arnaud, P. Leduc, A. Thuaire, and L. Anghel, "Electromigration behavior of 3d-ic tsv interconnects," in *2012 IEEE 62nd Electronic Components and Technology Conference*, pp. 326–330, 2012.

[20] S. Wang, T. Kim, Z. Sun, S. X.-D. Tan, and M. B. Tahoori, "Recovery-aware proactive tsv repair for electromigration lifetime enhancement in 3-d ics," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 531–543, 2018.

[21] Y. Liu, M. Li, M. Jiang, D. W. Kim, S. Gu, and K. N. Tu, "Joule heating enhanced electromigration failure in redistribution layer in 2.5d ic," in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, pp. 1359–1363, 2016.

[22] C. Wang, Q. Xu, C. Nie, H. Cao, J. Liu, D. Zhang, and Z. Li, "A multiscale anisotropic thermal model of chiplet heterogeneous integration system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 1, pp. 178–189, 2024.

[23] J. Chai, G. Dong, Z. Mei, and W. Zhu, "Steady state electrical–thermal coupling analysis of tsv," *Journal of Semiconductors*, 2018.

KVO-LLM: Boosting Long-Context Generation Throughput for Batched LLM Inference

Zhenyu Li†, Dongxu Lyu†, Gang Wang, Yuzhou Chen, Liyan Chen, Wenjie Li, Jianfei Jiang, Yanan Sun, Guanghui He*

State Key Laboratory of Micro/Nano Engineering Science, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

{ambitious-lzy, lvdongxu, wangganginjstu, Huygens, liyan.chen, wenjieli, jiangjianfei, sunyanan, guanghui.he}@sjtu.edu.cn

***Abstract*—With the widespread deployment of long-context large language models (LLMs), efficient and high-quality generation is becoming increasingly important. Modern LLMs employ batching and *key-value (KV) cache* to improve generation throughput and quality. However, as the context length and batch size rise drastically, the KV cache incurs extreme external memory access (EMA) issues. Recent LLM accelerators face substantial processing element (PE) under-utilization due to the low arithmetic intensity of attention with KV cache, while existing KV cache compression algorithms struggle with hardware inefficiency or significant accuracy degradation. To address these issues, an algorithm-architecture co-optimization, KVO-LLM, is proposed for long-context batched LLM generation. At the algorithm level, we propose a KV cache quantization-aware pruning method that first adopts salient-token-aware quantization and then prunes KV channels and tokens by attention guided pruning based on salient tokens identified during quantization. Achieving substantial savings on hardware overhead, our algorithm reduces the EMA of KV cache over 91% with significant accuracy advantages compared to previous KV cache compression algorithms. At the architecture level, we propose a multi-core jointly optimized accelerator that adopts operator fusion and cross-batch interleaving strategy, maximizing PE and DRAM bandwidth utilization. Compared to the state-of-the-art LLM accelerators, KVO-LLM improves generation throughput by up to 7.32×, and attains 5.52∼8.38× better energy efficiency.**

***Index Terms*—Large language model, Long-context, Batching, KV cache, Algorithm-architecture co-design**

I. Introduction

Long-context large language models (LLMs) show exceptional dexterity across a spectrum of natural language processing tasks with notable proficiency, especially in long-context generation [1]. With the increasing demand for long-context generation, the maximum context length has increased dramatically from 2K to millions of tokens. To improve the generation throughput, a prevalent strategy is to batch requests, exploiting data sharing [2], [3]. Moreover, LLMs broadly employ the pivotal *key-value (KV) cache* technique, which stores the intermediate attention keys and values to avoid heavy re-computation during long-context generation [4], [5].

However, with a drastic increase in batch size and context length, KV cache incurs extreme memory issues. As depicted in Fig. 1(a), the KV cache external memory access (EMA) increases linearly with batch size. Taking Llama-2-7B [6] as an example, as the batch size rises, the EMA of KV cache exceeds that of the model weights under 4K context length. Furthermore, as shown in Fig. 1(b), the KV cache related operations predominantly influence the generation throughput under large batch size. With the increase of context length, the challenges become more pronounced. For example, the EMA of KV cache even increases to 128 GB when the context length reaches 32K, becoming more than 20× the weight size. This extreme EMA accounts for 96% of the generation latency for Tender [7], forming a distinct bottleneck to throughput.

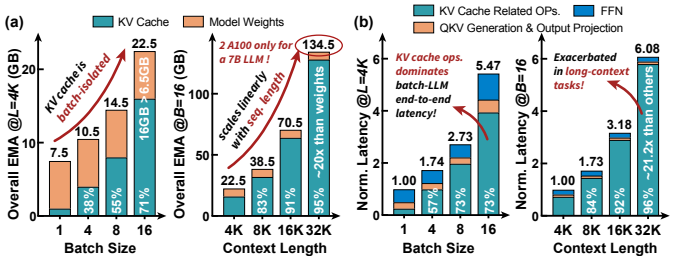


Fig. 1. Breakdown of EMA and latency for Llama-2-7B generating one token under different batch size B and context length L. Weights, activations, and KV cache are in INT8 format. The latency is measured on Tender [7] with a peak throughput of 2048GOPS@INT8 and a DRAM bandwidth of 256 GBps.

The above *KV cache issues have not been adequately addressed* in existing LLM accelerators [7]–[9]. Though they focus on lessening computational workload, there seems to be a common oversight regarding the low arithmetic intensity issues of KV cache. For example, Tender [7] and OPAL [8] quantize the LLMs into low precision to boost the processing element (PE) efficiency. MECLA [9] addresses the similar issue by compressing the weight matrices through fine-tuning. Nonetheless, in attention layers, their generation throughput is severely hindered by intensive EMA and low arithmetic intensity which causes serious PE under-utilization (e.g., only ≈25% of Tender under 16-batched Llama-2-7B decoding [7]).

Many previous algorithms have been dedicated to compressing the KV cache through *KV quantization (KV-Q)* or *KV eviction (KV-E)*. The state-of-the-art (SOTA) KV-Q works [10], [11] involve extra element-wise floating-point (FP) multiplication and accumulation (MAC) during matrix operations, which significantly degrades hardware efficiency. Meanwhile, other works of KV-E aim to retain a limited budget of important tokens, while evicting the trivial ones [12], [13]. However, discarded tokens might cause huge semantic information loss

†: Equal contribution.

This work was supported by the National Science Foundation of China under Grant 92464302. (*Corresponding author: Guanghui He).

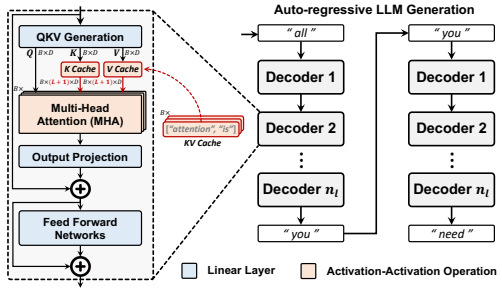


Fig. 2. Model architecture and auto-regressive inference of LLMs. As decoder layers focus on matrix computation, we exclude the layer normalization and position embedding in the diagram.

[14]. Despite its significant EMA savings, KV-E leads to noticeable accuracy loss on long-context generation [15].

In summary, KV cache emerges as a key barrier to high-throughput LLM inference while existing works leave a clear gap away from the memory-efficient design with high generation quality. Therefore, we consider three key requirements: ❶ *hardware efficiency* of the compression policy; ❷ *generation quality* under KV cache compression; and ❸ *utilization maximization* of PE and DRAM bandwidth. To meet these requirements, we propose an algorithm-architecture co-optimization, **KVO-LLM**. The key contributions are as follows:

1) We propose an effective KV cache quantization-aware pruning algorithm, which adopts differential and salient-token-aware quantization (DSQ) firstly. Then based on the identified salient tokens, it employs historical-current attention guided pruning (HCAP) to prune KV channels and tokens. Our method not only achieves *substantial savings on hardware overhead* ❶, but also delivers *high-quality generation comparable to that with full KV cache* ❷ under a high KV cache compression ratio (CR). This algorithm is open-sourced and available for public use.
2) We design a multi-core jointly optimized architecture with operator fusion and cross-batch interleaving strategy (OFBI) to schedule the compute flow wisely. Compared to naively scheduling our algorithmic flow, OFBI improves *PE utilization from 25% to 93% for multi-head attention and DRAM bandwidth usage from 25% to 67% for Query-Key-Value generation on average* ❸, thus achieving up to 1.72× throughput improvement.
3) We implement our design on TSMC 28nm technology and extensively evaluate it on various long-context generation tasks. KVO-LLM achieves comparable performance under a CR of over 91% as the model with full KV cache, while SOTA KV-Q and KV-E works still exhibit a notable accuracy loss even with a smaller CR. Compared with SOTA LLM accelerators, KVO-LLM improves generation throughput by up to 7.32× and achieves 5.52∼8.38× gain in energy efficiency.

II. Background and Motivation

A. Large Language Models

The LLM inference flow is typically organized into two phases: the *prefilling* phase and the *decoding* phase. The former calculates and stores the KV cache of the input prompt,

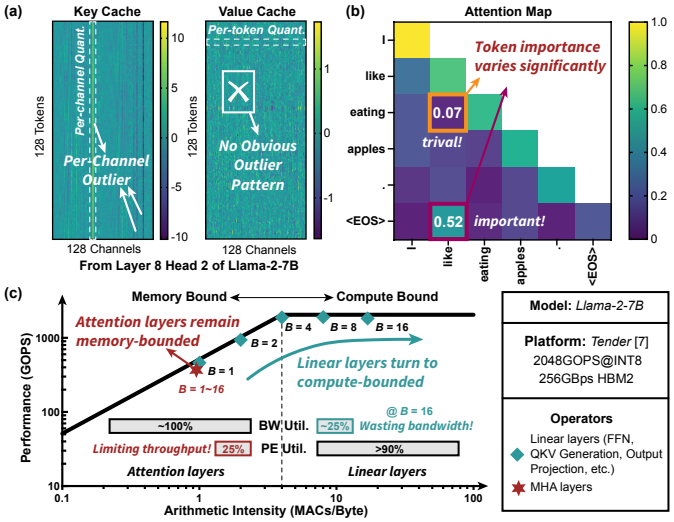


Fig. 3. (a) Magnitude of key and value cache from layer 8 head 2 of Llama-2-7B [6]. (b) The attention map of the input prompt "I like eating apples." (c) Roofline model of the Tender [7] for the decoding phase of Llama-2-7B when L is 4096, where BW represents the DRAM bandwidth.

and generates the first token. The latter phase, as illustrated in Fig. 2, generates the output tokens one by one, updating the KV cache during each regression loop. Since the prefilling phase is widely researched [7]–[9] and accounts for a minor portion of latency under long-context generation relatively [1], this paper focuses primarily on the *decoding* phase.

Fig. 2 shows that LLM is mainly constructed by several stacked decoder layers, each comprising four major parts: (1) Query($\mathbf{Q}$)-Key($\mathbf{K}$)-Value($\mathbf{V}$) generation, (2) multi-head attention (MHA), (3) output projection, and (4) feed forward networks (FFNs). (1), (3), and (4) are composed of linear layers, while the MHA primarily involves multiplication between activations. Denote the batch size, current context length, and embedding dimension by B, L, and D, respectively. After QKV generation, the newly generated $\mathbf{K}_i$, $\mathbf{V}_i \in \mathbb{R}^{B\times D}$ need to be concatenated with the cached $\mathbf{K}_C$, $\mathbf{V}_C \in \mathbb{R}^{B\times L\times D}$:

$$\begin{aligned} \mathbf{K},\mathbf{V} &= \texttt{Concat}(\mathbf{K}_C,\mathbf{K}_i),\texttt{Concat}(\mathbf{V}_C,\mathbf{V}_i),\\ \mathbf{K}_C,\mathbf{V}_C &= \mathbf{K},\mathbf{V}, \end{aligned} \tag{1}$$

where $\mathbf{K}$ and $\mathbf{V}$ participate in the following MHA: attention score computation $\mathbf{S} = \texttt{Softmax}(\mathbf{Q}_i\mathbf{K}^T)$, and $\mathbf{x}_{out} = \mathbf{SV}$.

B. Motivations

To compress KV cache, the most effective method is to reduce the total bytes it occupies, i.e., KV-Q. We profile the magnitude distribution of KV cache in Llama-2-7B [6] in Fig. 3(a). It could be observed that some $\mathbf{K}$ channels demonstrate large absolute magnitudes. Such features suggest a natural preference of per-channel $\mathbf{K}$ quantization [10]. However, this incurs an expensive overhead of FP MACs, as the $\mathbf{Q}_i\mathbf{K}^T$ computation of MHA is along the token dimension. Nevertheless, the per-token method, which is relatively hardware-friendly, introduces significant accuracy loss when quantizing vastly volatile channels [10]. Additionally, KV-E is an effective KV cache compression method by identifying and eliminating tokens with trivial context. However, as demonstrated in Fig.

3(b), the importance of tokens varies significantly across different context positions. Although research like H_2O [13] relies on historical accumulated attention scores to distinguish trivial tokens as precisely as possible, evicted tokens may become important in future generation, causing significant loss of semantic information [14]. Above all, the challenge for current KV cache compression lies in *ensuring both hardware efficiency and generation quality*.

Compared to the aforementioned algorithmic optimization, existing LLM accelerators often neglect the KV cache compute issues. We conduct a roofline analysis on Llama-2-7B in Fig. 3(c) based on Tender [7]. For the linear layers, batching improves the arithmetic intensity with weight sharing, whereas attention layers remain memory-bound consistently regardless of how batch size and context length increase. Divergent features determine that Tender suffers severe DRAM bandwidth or PE under-utilization in these two types of LLM layers respectively. As exhibited in Fig. 1(b), while KV cache scaling up with B and L, it exacerbates the aforementioned bottlenecks. Therefore, there is an urgent need for *the approach for PE efficiency and memory bandwidth maximization in batched LLM inference* to achieve high generation throughput.

III. Proposed KV Cache Compression Approach

In this section, we propose an effective quantization-aware pruning algorithm combining differential and salient-token-aware quantization (DSQ) with historical-current attention guided pruning (HCAP). DSQ identifies salient tokens and achieves extremely low-bit KV cache quantization. Based on the identified salient tokens, HCAP prunes a great number of trivial channels and tokens to enhance the KV cache CR. In Sec. III-A, we first introduce the DSQ. Then the HCAP will be described in Sec. III-B. Finally, Algorithm 1 details each step of our KVO-LLM algorithm combining DSQ with HCAP.

A. Differential and Salient-Token-Aware Quantization

To minimize the FP MACs, it is essential to ensure that the quantization dimension align with the accumulation. Therefore, we should employ per-token $\mathbf{K}$ and per-channel $\mathbf{V}$ quantization. However, using it naively causes huge accuracy drop [10]. To overcome this dilemma, the DSQ is proposed.

1) Differential Per-Token $\mathbf{K}$ Quantization. As illustrated in Fig. 3(a), adjacent $\mathbf{K}$ exhibit high numerical similarity along the channel dimension. This property indicates that the differences between adjacent tokens are minimal. Leveraging this opportunity, we propose a differential quantization method that partitions all tokens into groups, each containing one base token ($\mathbf{K}_{base}$) and several Δ tokens ($\mathbf{K}_{\Delta}$). When quantizing each group, the $\mathbf{K}_{base}$ is directly quantized to INT8, while the $\mathbf{K}_{\Delta}$ are represented by the low-bit quantized differential values relative to the $\mathbf{K}_{base}$:

$$\texttt{Quant}(\mathbf{K}_{\Delta}) = \texttt{Quant}(\mathbf{K}_{\Delta} - \mathbf{K}_{base}, \texttt{INT2}) \tag{2}$$

A new challenge arises in selecting the $\mathbf{K}_{base}$ and implementing effective grouping strategies. According to [13], selecting tokens with the highest accumulated attention scores as base tokens like H_2O [13] is the most appropriate approach. However, this can lead to an uneven distribution in memory, resulting in irregular memory access. Therefore, we propose a window-based $\mathbf{K}_{base}$ selection strategy at line 18-20 in Algorithm 1. Every n_{win} tokens form a group, and within each group, the token with the highest accumulated score from the most recent R generations is identified as the salient token, with its corresponding attention key serving as $\mathbf{K}_{base}$.

2) Salient-Token-Aware Per-Channel $\mathbf{V}$ Quantization. As shown in Fig. 3(a), $\mathbf{V}$ does not exhibit an obvious outlier pattern like $\mathbf{K}$. As a result, for per-channel quantization, the precision of salient tokens is susceptible to the quantization process imposed on trivial tokens. Given that the result ($\mathbf{x}_{out}$) of $\mathbf{SV}$ primarily depends on the combination of a few important tokens, compared to $\mathbf{K}$ quantization, we do not employ a differential process on $\mathbf{V}$ to mitigate interactions between tokens. We allocate high bit-width (8-bit) for salient tokens, and assign low bit-width (2-bit) to trivial tokens.

Algorithm 1: Proposed KVO-LLM Algorithm

Input: Query $\mathbf{Q} \in \mathbb{R}^{L\times D}$, KV cache $\mathbf{K}$, $\mathbf{V} \in \mathbb{R}^{L\times D}$, KV cache budget $k \in \mathbb{N}$, recent generation observation length R, window size n_{win}, pruning ratio P
Output: MHA results $\mathbf{x}_{out}$

1 Set $\mathbf{S}_0 = 0$, $\mathbf{S}_0 \in \mathbb{R}^L$; and $i = 0$;
2 **for** $i = 1 \sim L$ **do**
3 $\quad \mathbf{Q}_i = \mathbf{Q}[i, :]$
4 $\quad$ // **Pre-compute Stage**;
5 $\quad$ // *Sec. III-B: Current Attention Guided Pruning*;
6 $\quad$ // *Estimate current attention scores based on* $\mathbf{K}_{base}$;
7 $\quad \mathbf{A}_{est} = \mathbf{Q}_i\mathbf{K}_{base}^T$;
8 $\quad$ *Select* k *budget tokens* $\mathbf{K}_{bgt}$, $\mathbf{V}_{bgt}$ *based on* $\mathbf{A}_{est}$;
9 $\quad$ *Select recent tokens* $\mathbf{K}_{rec}$, $\mathbf{V}_{rec}$ *based on* R;
10 $\quad \mathbf{K}_s, \mathbf{V}_s = \texttt{Concat}(\mathbf{K}_{bgt}, \mathbf{K}_{rec}), \texttt{Concat}(\mathbf{V}_{bgt}, \mathbf{V}_{rec})$;
11 $\quad$ // **Formal-compute Stage**;
12 $\quad \mathbf{S}_{cur} = \texttt{Softmax}(\mathbf{Q}_i\mathbf{K}_s^T)$;
13 $\quad \mathbf{x}_{out} = \mathbf{S}_{cur}\mathbf{V}_s$;
14 $\quad$ // *Accumulated with historical attention scores* $\mathbf{S}_{i-1}$;
15 $\quad \mathbf{S}_i = \mathbf{S}_{i-1} + \mathbf{S}_{cur}$;
16 $\quad$ **if** $(i \,\%\, R == 0)$ ***and*** $(i \geq 2R)$ **then**
17 $\quad\quad$ // *Sec. III-A: Select base tokens;*
18 $\quad\quad \mathbf{S}_{win} = \mathbf{S}_i[i - 2R : i - R]$;
19 $\quad\quad \mathbf{S}_{win} = \mathbf{S}_{win}.\texttt{reshape}(n_{win}, -1)$;
20 $\quad\quad slt_{idx} = \textbf{topk}(\mathbf{S}_{win})$; // *Salient tokens index;*
21 $\quad\quad \mathbf{K}_{cmps}, \mathbf{V}_{cmps} = \mathbf{K}, \mathbf{V}[i - 2R : i - R, :]$;
22 $\quad\quad$ // *Sec. III-A: DSQ;*
23 $\quad\quad (\mathbf{K}_{cmps}, \mathbf{V}_{cmps}).\texttt{Quant}(slt_{idx})$;
24 $\quad\quad$ // *Sec. III-B: Historical Attention Guided Pruning;*
25 $\quad\quad (\mathbf{K}_{cmps}, \mathbf{V}_{cmps}).\texttt{Prune}(slt_{idx}, P)$;

B. Historical-Current Attention Guided Pruning

Further reduction in quantization bit-width can significantly impair accuracy. To enhance the CR, it is necessary to prune the trivial information within the KV cache. We rely on both historical accumulated attention scores and current attention scores, i.e., current $\texttt{Softmax}(\mathbf{Q}_i\mathbf{K}_{base}^T)$ results, to guide the pruning process and mitigate semantic information loss.

1) Historical Attention Guided Pruning. Evicting tokens thoroughly based on historical accumulated attention scores causes significant loss of semantic information. However,

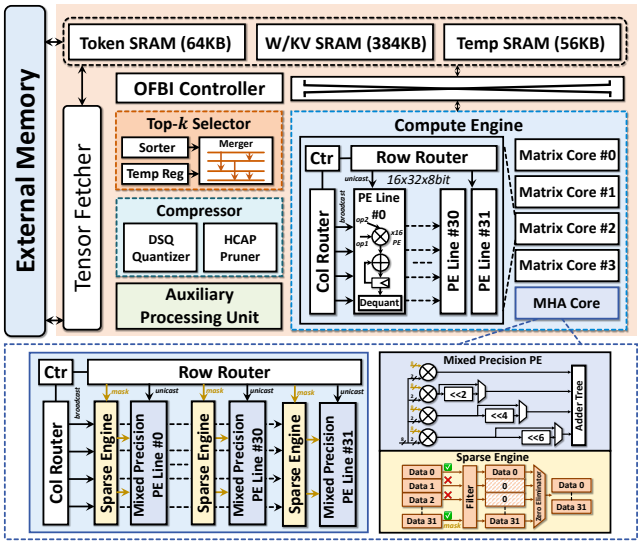


Fig. 4. Overview of the KVO-LLM accelerator.

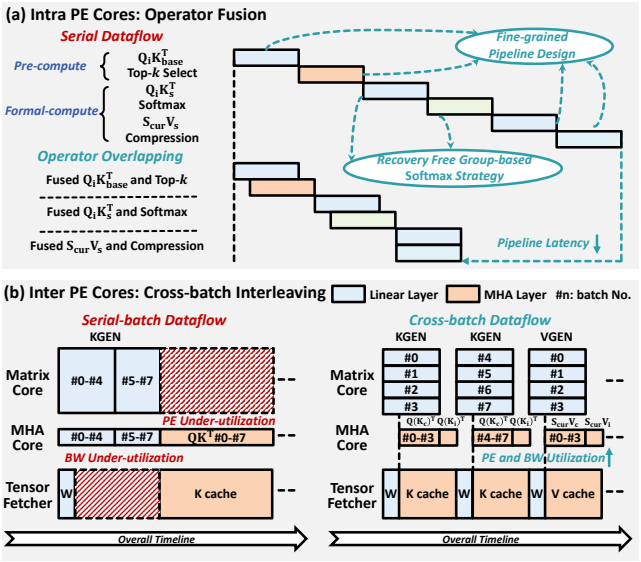


Fig. 5. Proposed operator fusion and cross-batch interleaving (OFBI) dataflow. (a) Intra PE cores operator fusion to reduce pipeline latency. (b) Inter PE cores cross-batch interleaving to improve PE and DRAM bandwidth utilization.

there are many channels with small magnitudes of trivial tokens delivering negligible contribution to the final results. Therefore, we selectively prune these trivial channels at line 25 in Algorithm 1. Based on the absolute values, we adopt a structured pruning method that eliminates the lowest N values in each quantization group. When pruning ratio is $1-P$, the compression ratio (CR) of one group is reduced to:

$$G_{CR}(P) = (8D + 2DP(n_{win} - 1))/16DL \tag{3}$$

2) Current Attention Guided Pruning. Large pruning ratio (>75%) solely based on historical attention scores results in significant accuracy degradation. Fortunately, prior research demonstrates the inherent sparsity in attention scores [14]. Therefore, to mitigate accuracy loss while achieving a high CR, we propose to estimate the current attention scores and utilize them to select a small portion of tokens participating in current computation at line 7-8 in Algorithm 1. First, instead of multiplying $\mathbf{Q}_i$ with all $\mathbf{K}$, we approximate attention scores by multiplying $\mathbf{Q}_i$ and $\mathbf{K}_{base}$. The scores estimate the importance of each KV group. Then, given a KV cache budget k, only the top-(k/n_{win}) groups with the largest $\mathbf{Q}_i\mathbf{K}_{base}^T$ results participate in the following computations.

When $L\%R = 0$, the $\mathbf{K}$ cache EMA CR using our approach is reduced to

$$\mathbf{K}_{CR} = 1 - (\underbrace{R/L}_{recent} + \underbrace{G/2L}_{base} + \underbrace{(k/n_{win})G_{CR}(P)}_{k\ budget}), \tag{4}$$

where $G = (L-R)/n_{win}$, which is # of groups. For $\mathbf{V}_{CR}$, it only includes the EMA of recent tokens and k budget tokens.

IV. Proposed Generative LLM Accelerator

As depicted in Fig. 3(c), existing LLM-specific designs face severe PE or DRAM bandwidth under-utilization issues. We design a multi-core jointly optimized architecture which schedules the compute flow wisely leveraging operator fusion and cross-batch interleaving (OFBI) strategy. We first present the overview of our accelerator in Sec. IV-A. Then, we show how operators are fused within one core in Sec. IV-B. The cross-batch interleaving strategy is described in Sec. IV-C.

A. Overview

Fig. 4 depicts the overall architecture of KVO-LLM. It primarily consists of 504KB on-chip buffer, an auxiliary processing unit, a compute engine, and specialized functional units. The on-chip buffer is divided into three main parts for input feature maps, weights/KV cache, and temporary results respectively. The auxiliary processing unit is mainly responsible for `Softmax` and other non-linear functions. The compute engine is composed of five PE cores focused on matrix multiplication, including one MHA core dedicated to multi-head attention. The specialized functional units include a top-k selector and a compressor to support our KV cache quantization-aware pruning algorithm.

B. Intra PE Cores: PE Design and Operator Fusion

As shown in Fig. 4, each PE core consists of 32 PE lines, and each PE line is composed of 16 PEs. Each PE core performs $(1 \times 16) \times (16 \times 32)$ matrix-vector multiplication per cycle, and it follows an output stationary dataflow, where partial sums are accumulated in each PE line. The MHA core is composed of precision-configurable PEs and sparse engines based on zero elimination to support mixed-precision quantization and pruning respectively.

When performing MHA in a single batch, our algorithm necessitates data recovery of $\mathbf{Q}_i\mathbf{K}_{\Delta}^T$ because of DSQ, top-k selection because of HCAP, and compression of new KV tokens. Naively performing these operations in a serial manner causes long pipeline latency, which could be seen in Fig. 5(a). We propose a fine-grained fusion dataflow to improve PE cores utilization significantly. In the pre-compute stage, once a valid $\mathbf{Q}_i\mathbf{K}_{base}^T$ result is finished, the top-k selector compares it with previous results. In the formal-compute stage, the `Softmax`

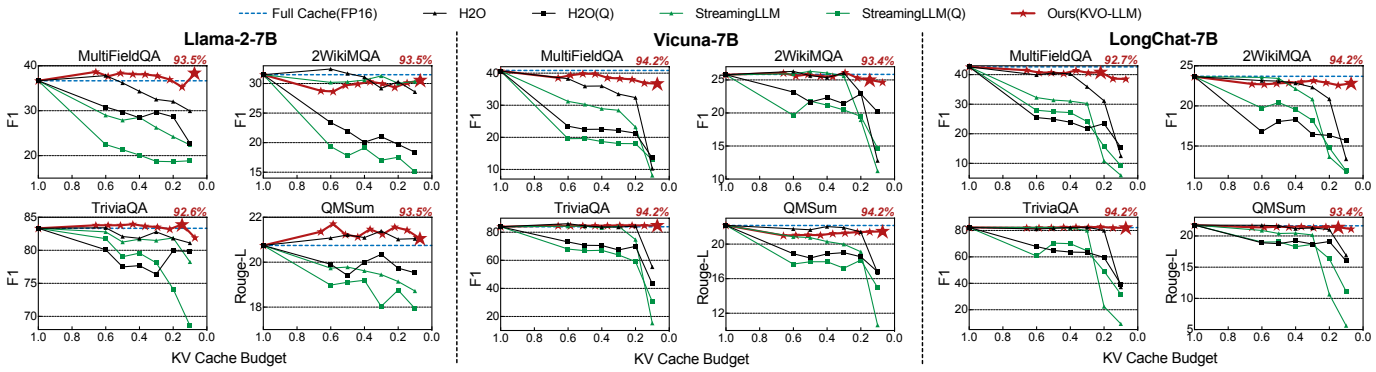


Fig. 6. Comparsion results between the baseline model with full cache, our KVO-LLM algorithm, and the existing KV-E methods [12], [13] with or without quantization. We choose 4 representative datasets for distinct tasks: single-document QA, multi-document QA, synthetic tasks, and summarization, respectively, from the extensively utilized LongBench [15]. For each diagram, we highlight one point with significant CR and high accuracy compared to SOTA works, and we present the actual CR. Our method could compress the KV cache EMA over 91% while demonstrating negligible accuracy loss.

can overlap with $\mathbf{Q}_i\mathbf{K}_s^T$ by fusing data recovery process with the `Softmax` operation and utilizing a group-based `Softmax` strategy. More specifically, we implement `Softmax` for each group of $\mathbf{Q}_i\mathbf{K}_s^T$, i.e., n_{win} results. By designating $\mathbf{Q}_i\mathbf{K}_{base}^T$ as the `Softmax` maximum, we eliminate the necessity for completing data recovery. After finishing $\mathbf{Q}_i\mathbf{K}_s^T$, we update group-based `Softmax` results accordingly. Then the compute engine begins $\mathbf{S}_{cur}\mathbf{V}_s$. Compression on new KV tokens can be operated simultaneously on the specially designed compressor.

C. Inter PE Cores: Cross-Batch Interleaving

As shown in Fig. 3 (c), in the linear layers, batching enhances arithmetic intensity, while attention layers consistently exhibit memory bound, irrespective of batch size and context length. Fig. 5(b) presents an unoptimized execution timeline for the multi-batch computational process, which executes the QKV generation and MHA serially. This execution flow leads to severe under-utilization of both PE and DRAM bandwidth.

Therefore, we introduce a cross-batch interleaving approach. Fig. 5(b) illustrates an example of this technique, which is accomplished by partitioning the compute cores into separate matrix and MHA cores to accommodate distinct workloads. Since there is no dependency between KV generation and MHA with cached $\mathbf{K}_C, \mathbf{V}_C$, MHA is divided into two computational segments: MHA with cached $\mathbf{K}_C, \mathbf{V}_C$ and MHA with new $\mathbf{K}_i, \mathbf{V}_i$. We dispatch them with other batches to optimize resource utilization. To be specific, because of the compute-bounded property in batched KV generation, there is a long period of DRAM idleness. Instead of scheduling all compute cores to operate on KV generation, we schedule one MHA core for MHA with cached $\mathbf{K}_C, \mathbf{V}_C$, fully utilizing the wasted DRAM bandwidth. Cross-batch interleaving facilitates the concurrent execution of compute-intensive linear layers alongside memory-bound attention layers, and thus effectively improves PE and DRAM bandwidth utilization.

V. Evaluation

A. Evaluation Methodology

We evaluate KVO-LLM on four datasets in the widely-used *LongBench* [15] to demonstrate our superiority over diverse tasks, e.g., document QA, summarization, and synthetic tasks. *LongBench* provides a holistic evaluation of long-context generation abilities of LLMs across various lengths, distributions, and long dependency patterns [15]. Detailed configurations and implementation are derived from the official repository of H_2O [13] and KIVI [10]. Meanwhile, three representative long-context LLMs are chosen for our evaluation: Llama-2-7b-chat-4k (*Llama-2-7B*) [6], LongChat-v1.5-7b-32k (*LongChat-7B*) [16], Vicuna-7b-v1.5-16k (*Vicuna-7B*) [17]. When evaluating accuracy, we use the PyTorch [18] framework and the Hugging Face library [19].

For hardware implementation, we implement the compute and control logic of KVO-LLM with SystemVerilog and estimate the area and energy consumption of on-chip memory with CACTI [20]. The prototype is synthesized with Synopsys Design Compiler under TSMC 28nm CMOS technology. Logic power consumption is estimated using Synopsys PrimeTimePX by annotating switching activity from the datasets above. For off-chip DRAM, we use an HBM2 with 256GB/s bandwidth, which consumes 1.2pJ/b for data access [21].

TABLE I
Comparisons with Quant-only Strategies

LLM	KV Quant	Evaluation Scores			KV CR
		MultiFieldQA	2WikiMQA	QMSum	(%)
Llama-2-7B	FP16	36.70	31.51	20.75	0
	NINT2†	28.49	20.09	19.70	87.50
	KIVI⋆	35.07	29.04	20.70	85.90
	Ours⋆⋆	**37.74**	**29.99**	**21.24**	**90.89**
LongChat-7B	FP16	42.66	23.66	21.75	0
	NINT2†	32.94	20.65	19.19	87.50
	KIVI⋆	38.79	22.74	**21.64**	86.10
	Ours⋆⋆	**40.63**	**23.17**	21.36	**91.93**
Vicuna-7B	FP16	40.68	25.79	22.19	0
	NINT2†	23.36	19.96	19.00	87.50
	KIVI⋆	34.65	25.36	21.42	86.10
	Ours⋆⋆	**38.34**	**26.16**	**21.78**	**91.93**

† *NINT2*: Naive per-token INT2 quantization for KV cache following the same group granularity as KVO-LLM.
⋆ *KIVI*: Except for setting the residual length to 64, all other configurations remain the same as in the original paper.
⋆⋆ Our results are tested under $k = 896$, $R = 64$, $P = 50\%$.

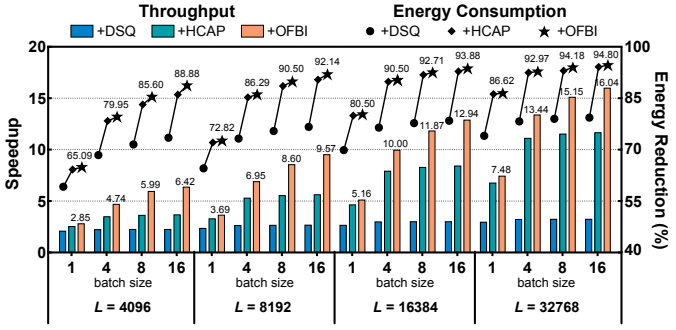


Fig. 7. Normalized throughput and energy consumption improvements through DSQ, HCAP and OFBI. Note that "+" means this method is appended on the previous approach. Baseline has the same hardware architecture with these functions deactivated.

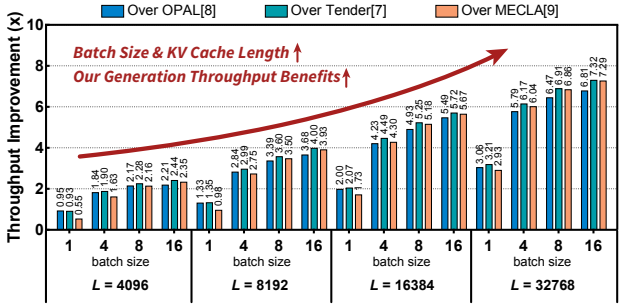


Fig. 8. Normalized throughput improvements of KVO-LLM with SOTA LLM accelerators [7]–[9]. We ensure the peak throughput is consistent across all works to make a fair comparison.

B. Evaluation on Proposed Algorithm

We first evaluate our method with SOTA KV-Q works, such as KIVI. As shown in Table I, NINT2 suffers from significant accuracy loss due to the outlier channels in $\mathbf{K}$. Although KIVI mitigates the quantization error by adjusting the quantization dimensions, it introduces inefficient FP MACs. Compared to KIVI with up to a 14.82% accuracy drop and an 86.10% KV cache CR, KVO-LLM only has up to a 5.76% accuracy drop despite achieving up to 91.93% CR. KVO-LLM outperforms existing methods on both accuracy and KV cache CR.

We further compare our algorithm with SOTA KV-E works across different KV cache budget in Fig. 6. To clearly demonstrate our strengths, the KV cache budget shown in the figure is calculated as k/L. However, with the inclusion of DSQ and HCAP, the CR using our method significantly surpasses $1 - k/L$, as calculated in (4). We can observe that: (1) KVO-LLM with a CR above 91% can achieve comparable accuracy to the model with full KV cache (FP16). (2) KVO-LLM shows increasingly significant accuracy advantages under high CR. (3) Combining DSQ and HCAP, KVO-LLM demonstrates consistent and significant accuracy improvements over KV-E works with quantization.

C. Evaluation on Proposed Hardware Accelerator

Fig. 7 presents the throughput and energy consumption evaluations over three key techniques: DSQ, HCAP, and OFBI. We set $k = L/8$ for the following evaluations.

When compared to the baseline, DSQ and HCAP yield over 3.72× enhancement in throughput and over 86.3% reduction in power consumption under $B = 16, L = 4096$. The introduced OFBI strategy further improves throughput by up to 1.72× and decreases power consumption by up to 1.24×. Additionally, as illustrated in Fig. 7, the performance improvement is typically greater with larger batch size and context length, as these conditions elevate the latency and EMA contribution from the MHA stage within the processing flow.

TABLE II
COMPARISON WITH OTHER LLM ACCELERATORS

	OPAL [8]	Tender [7]	MECLA [9]	Ours
Technology(nm)	65	28	28	28
Frequency(MHz)	500	1000	1000	500
Area(mm^2)	0.69‡	3.98	22.02	4.83
Memory(KB)	514	608	1280	504
Throughput(GOPS)⋆	71	1328	1460	**3287**
Power(mW)	700	1600	1976	717
Energy Efficiency† (GOPS/W)	546.60	830.00	738.87	**4584.38**

⋆ Evaluation on Llama-2-7B under $L = 4096$, $B = 16$. ‡ Without SRAM.
† Technology normalized to 28nm as follows: s = Technology/28nm, $f \sim s$, and $P \sim (1/s)$, where f and P is denoted as frequency and power, respectively.

D. Comparison with SOTA LLM Accelerators

We conduct a comparison of KVO-LLM with leading LLM accelerators [7]–[9], and ensure the peak throughput is consistent across all works. Compared with them, KVO-LLM effectively reduces the EMA of KV cache with a CR >91% by employing DSQ and HCAP. Moreover, utilizing the OFBI, KVO-LLM significantly improves PE utilization from 25% to 93% for MHA and DRAM bandwidth usage from 25% to 67% for QKV generation. As a result, KVO-LLM improves throughput by up to 7.32× in Fig. 8.

Table II presents the detailed implementation results. Utilizing TSMC 28nm technology, the KVO-LLM prototype demonstrates an effective performance of 4584.38 GOPS/W at 500 MHz, with a power consumption of 717 mW. The accelerator occupies an area of 4.83 mm^2. We also reconstruct the method of [7]–[9] for the LLM generation phase to evaluate their energy efficiency. Leveraging DSQ, HCAP and OFBI, KVO-LLM shows a 5.52∼8.38× increase in energy efficiency compared to these accelerators.

VI. Conclusion

This paper introduces an algorithm-architecture co-optimization, KVO-LLM. The key insight is to alleviate the significant KV cache EMA and improve the throughput for long-context batched LLM generation. Through quantization-aware pruning, KVO-LLM not only achieves substantial savings on hardware overhead, but also compresses KV cache EMA by over 91% with high generation quality. Besides, KVO-LLM hardware accelerator integrates the OFBI strategy to maximize PE and DRAM bandwidth utilization. Comprehensive evaluations demonstrate KVO-LLM surpasses SOTA works in various metrics such as accuracy, throughput, and energy efficiency across diverse tasks and model types.

REFERENCES

[1] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li, S. Yan, G. Dai, X. Zhang, Y. Dong, and Y. Wang, "A Survey on Efficient Inference for Large Language Models," *CoRR*, vol. abs/2404.14294, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2404.14294

[2] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, R. Gupta, N. B. Abu-Ghazaleh, M. Musuvathi, and D. Tsafrir, Eds. ACM, 2024, pp. 722–737. [Online]. Available: https://doi.org/10.1145/3620666.3651380

[3] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, R. Gupta, N. B. Abu-Ghazaleh, M. Musuvathi, and D. Tsafrir, Eds. ACM, 2024, pp. 103–119. [Online]. Available: https://doi.org/10.1145/3620665.3640422

[4] W. Lee, J. Lee, J. Seo, and J. Sim, "InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management," in *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, A. Gavrilovska and D. B. Terry, Eds. USENIX Association, 2024, pp. 155–172. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/lee

[5] Y. Zhao, D. Wu, and J. Wang, "ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching," in *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 2024, pp. 1005–1017. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00077

[6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open Foundation and Fine-Tuned Chat Models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[7] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization," in *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 2024, pp. 1048–1062. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00080

[8] J. Koo, D. Park, S. Jung, and J. Kung, "OPAL: Outlier-Preserved Microscaling Quantization Accelerator for Generative Large Language Models," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657323

[9] Y. Qin, Y. Wang, Z. Zhao, X. Yang, Y. Zhou, S. Wei, Y. Hu, and S. Yin, "MECLA: Memory-Compute-Efficient LLM Accelerator with Scaling Sub-matrix Partition," in *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 2024, pp. 1032–1047. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00079

[10] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=L057s2Rq8O

[11] Z. Zhang, S. Liu, R. Chen, B. Kailkhura, B. Chen, and A. Wang, "Q-Hitter: A Better Token Oracle for Efficient LLM Inference via Sparse-Quantized KV Cache," in *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*, P. B. Gibbons, G. Pekhimenko, and C. D. Sa, Eds. mlsys.org, 2024. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/hash/bbb7506579431a85861a05fff048d3e1-Abstract-Conference.html

[12] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient Streaming Language Models with Attention Sinks," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=NG7sS51zVF

[13] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. W. Barrett, Z. Wang, and B. Chen, "H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html

[14] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=KzACYw0MTV

[15] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, and J. Li, "LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 3119–3137. [Online]. Available: https://doi.org/10.18653/v1/2024.acl-long.172

[16] D. Li, R. Shao, A. Xie, Y. Sheng, L. Zheng, J. E. Gonzalez, I. Stoica, X. Ma, and H. Zhang, "How Long Can Open-Source LLMs Truly Promise on Context Length?" June 2023. [Online]. Available: https://lmsys.org/blog/2023-06-29-longchat/

[17] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, "Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90% Chatgpt Quality," March 2023. [Online]. Available: https://lmsys.org/blog/2023-03-30-vicuna/

[18] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," 2017.

[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and b. . P. o. t. . C. o. E. M. i. N. L. P. S. D. Alexander M. Rush", "Transformers: State-of-the-Art Natural Language Processing." Online: Association for Computational Linguistics, oct 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[20] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), 1-5 December 2007, Chicago, Illinois, USA*. IEEE Computer Society, 2007, pp. 3–14. [Online]. Available: https://doi.org/10.1109/MICRO.2007.33

[21] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17, New York, NY, USA, 2017, p. 41–54.

GLOVA: Global and Local Variation-Aware Analog Circuit Design with Risk-Sensitive Reinforcement Learning

Dongjun Kim[1,3*], Junwoo Park[1*], Chaehyeon Shin[2,3*], Jaeheon Jung[2],
Kyungho Shin[3], Seungheon Baek[3], Sanghyuk Heo[3], Woongrae Kim[3], Inchul Jeong[3], Joohwan Cho[3] and Jongsun Park[1]
[1]Department of Electrical Engineering, Korea University, Seoul, Republic of Korea
[2]Department of Semiconductor System Engineering, Korea University, Seoul, Republic of Korea
[3]SK hynix, Icheon, Republic of Korea

***Abstract*—Analog/mixed-signal circuit design encounters significant challenges due to performance degradation from process, voltage, and temperature (PVT) variations. To achieve commercial-grade reliability, iterative manual design revisions and extensive statistical simulations are required. While several studies have aimed to automate variation-aware analog design to reduce time-to-market, the substantial mismatches in real-world wafers have not been thoroughly addressed. In this paper, we present GLOVA, an analog circuit sizing framework that effectively manages the impact of diverse random mismatches to improve robustness against PVT variations. In the proposed approach, risk-sensitive reinforcement learning is leveraged to account for the reliability bound affected by PVT variations, and ensemble-based critic is introduced to achieve sample-efficient learning. For design verification, we also propose μ-σ evaluation and simulation reordering method to reduce simulation costs of identifying failed designs. GLOVA supports verification through industrial-level PVT variation evaluation methods, including corner simulation as well as global and local Monte Carlo (MC) simulations. Compared to previous state-of-the-art variation-aware analog sizing frameworks, GLOVA achieves up to 80.5× improvement in sample efficiency and 76.0× reduction in time.**

***Index Terms*—Analog circuit synthesis, PVT variation, Reinforcement learning**

I. Introduction

Analog/mixed-signal circuit design is an extremely labor-intensive process, relying on human expertise, experience, and intuition. As CMOS technology continues to scale down, it has become increasingly challenging to meet the demand for higher performance and shorter design cycles. Consequently, both industry and academia are driven toward developing advanced design automation tools. Over the years, numerous studies have incorporated machine learning to mitigate reliance on manual design. [1], [2] utilize genetic algorithms with deep neural network-based models for optimization. Bayesian optimization (BO) [3]–[5] and reinforcement learning (RL) [6]–[9] are also widely used to efficiently explore large design space.

A key challenge in analog circuit design is dealing with performance degradation caused by process, voltage, and temperature (PVT) variations. In particular, process variations during manufacturing introduce substantial silicon mismatches. As shown in Fig. 1, these variations are classified into global mismatches, which occur broadly across the entire wafer, and local mismatches within individual dies [10], [11]. Voltage variation results from non-idealities of external voltage sources, while temperature variation arises from changes in operating conditions. Since the performance of analog circuits under PVT variations is highly sensitive and unpredictable, addressing these effects is crucial. In industry, where maintaining yield is paramount, this challenge is even more pressing. Therefore, extensive verification across various PVT conditions is conducted to ensure chip reliability, even though it is time-consuming and costly.

*These authors contributed equally to this work.

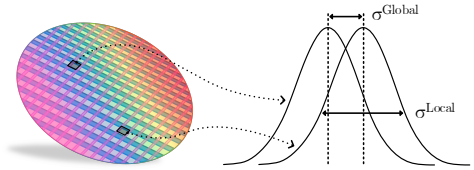


Fig. 1. Schematic of global (die-to-die) and local (within-die) variations on a wafer [11]. The median difference between two dies is determined by global variation (σ^{Global}), while variations within each die occur around its median due to local variation (σ^{Local}).

Existing design automation tools primarily focus on typical condition and cannot effectively account for diverse variations [1]–[5], [12]. Although [4], [12] introduce PVT corners, they merely test all PVT conditions every iteration, which limits sample efficiency. To tackle this, [8] employs multi-task RL, treating each PVT corner as a separate task, and improves efficiency by testing only the dominant corners identified through clustering. Nonetheless, using random initial sampling limits both the sample efficiency and success rate of the optimization process. To overcome this, [9] incorporates TuRBO [13] for the initial sampling, enhancing optimization efficiency.

While [4], [7]–[9], [12] provide insights into addressing PVT variations, their focus remains primarily on global process corners, leaving the impact of extensive random mismatches both across and within dies underexplored. Although [9] examines transistor-level mismatches, it is limited to a few cases, making it difficult to reflect the comprehensive impact of mismatches. Additionally, the consideration of mismatches is handled separately from the optimization process, requiring additional iterations and reducing overall design efficiency. In terms of verification, ensuring chip reliability necessitates a large number of statistical simulations to address the significant impact of random mismatches in real-world wafers [14]. Nevertheless, prior works have not considered comprehensive verification in the design process. Therefore, it is essential to thoroughly address these factors within the design flow to overcome the challenges in both optimization and verification processes.

In this paper, we present GLOVA, an efficient optimization and verification framework for variation-aware analog circuit design automation. Experimental results demonstrate that the proposed GLOVA efficiently and effectively manages diverse PVT conditions and a wide range of mismatch cases compared to previous state-of-the-art approaches.

Our contributions are summarized as follows,

- We propose an automated optimization and verification framework for PVT-aware analog circuit design, accounting for broad and diverse global and local mismatches via risk-sensitive reinforcement learning.
- Ensemble-based critic is proposed to estimate the bound of

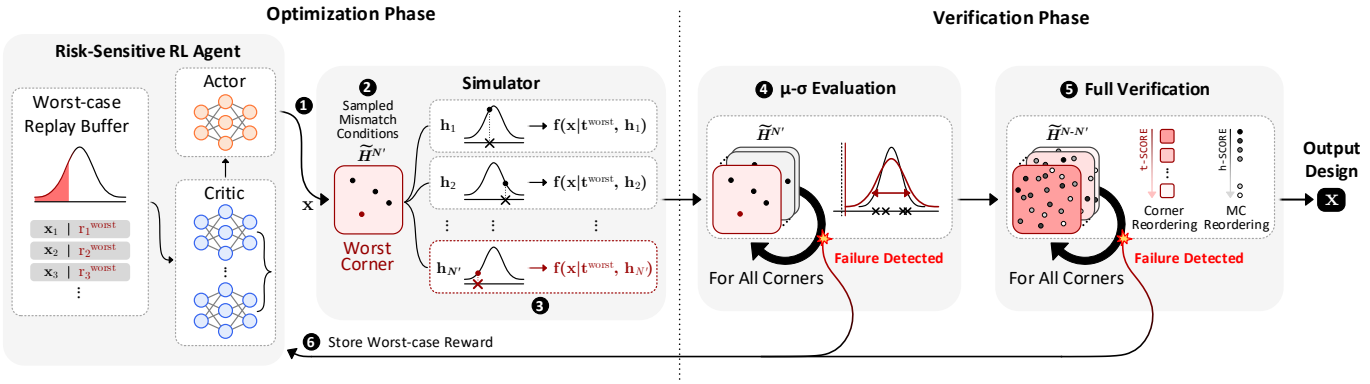


Fig. 2. Framework overview of the proposed GLOVA.

uncertain worst-case variations, improving sample efficiency and optimization convergence.

- μ-σ evaluation method is proposed to determine if an optimized design is feasible and whether to further verify it, improving sample efficiency and reducing overall runtime.
- Simulation reordering method is proposed, significantly reducing the verification cost of a large number of statistical simulations.
- The produced designs are verified through real-world simulation scenarios, including corner simulation, corner with local Monte Carlo (MC) simulation, and corner with global-local MC simulation.
- Extensive experimental results demonstrate that GLOVA achieves up to 80.5× improvement in sample efficiency and 76.0× runtime reduction compared to state-of-the-art tools.

II. Preliminaries

A. Simulation Methods for Analyzing PVT Variation Effects

Corner and Monte Carlo (MC) simulations are commonly used to evaluate analog circuit performance under varying process, voltage, and temperature (PVT) conditions. Corner simulation tests extreme conditions by analyzing predefined PVT corners, such as process corners $\{\mathrm{TT}, \mathrm{SS}, \mathrm{FF}, \mathrm{SF}, \mathrm{FS}\}$, voltage levels $\{0.8\mathrm{V}, 0.9\mathrm{V}\}$, and temperatures $\{-40\,^\circ\mathrm{C}, 27\,^\circ\mathrm{C}, 80\,^\circ\mathrm{C}\}$. Regarding process variation, corner simulation primarily reflects global variation between dies. In contrast, MC simulation statistically assesses process variations by evaluating numerous random parameter combinations. By exploring a large number of combinations, it can probabilistically reveal worst-case scenarios. In practical design, corner simulation and MC simulation can be appropriately combined to experiment with a wide range of mismatch cases under various PVT conditions [15]. To ensure robust circuit design, thorough verification using these methods is the fundamental requirement.

B. Risk-Sensitive Reinforcement Learning

Deep reinforcement learning (RL) is an emerging field used to optimize decision-making in complex environments. In traditional RL, the goal is typically to maximize the expected cumulative reward. In contrast, environments with significant uncertainty introduce unexpected occurrences of negative outcomes that can lead to potential risks. Thus, to focus not only on reward maximization but also on risk management, risk-sensitive RL [16]–[18] aims to balance the expected reward and risk by penalizing deviations from the expected outcomes during simulated rollouts. The objective function J in risk-sensitive RL, referred to as risk-sensitive criterion [18], modifies the standard objective to prioritize stability in worst-case scenarios under stochastic uncertainty, and it is expressed as:

$$J = \mathbb{E}[R] + \beta\sigma[R] \tag{1}$$

where $\mathbb{E}[R]$ is the expected reward, $\sigma[R]$ is the standard deviation of reward, and β is the risk-sensitivity parameter. Adjusting β controls the level of risk, with $\beta > 0$ for risk-seeking, $\beta < 0$ for risk-avoidance, and $\beta = 0$ for risk neutrality [17]. By modeling and carefully accounting for worst-case scenarios in the optimization process of stochastic and unpredictable systems, this approach helps to minimize failure costs. As a result, it allows desired targets to be reached more efficiently, especially when the cost of failure is high.

III. Framework Overview of Proposed GLOVA

A. Problem Formulation

Our goal is to find a sizing vector ensuring that the circuit's performance meets the design targets under PVT variations. Therefore, the problem can be formulated as a constraint satisfaction problem:

$$\begin{aligned} &\text{minimize} \quad 0 \\ &\text{subject to} \quad \max_{\mathbf{h}\in\widetilde{H}_j^N} \mathcal{F}_i(\mathbf{x}|\mathbf{t}_j,\mathbf{h}) < c_i, \quad i=1,...,m, \quad j=1,...,k, \end{aligned} \tag{2}$$

where $\mathbf{x} \in \mathcal{X}^p$ represents the design solution in a p-dimensional design space $\mathcal{X}$, and $\mathcal{F}_i(\mathbf{x}|\mathbf{t}_j,\mathbf{h})$ denotes the i-th performance metric under the j-th PVT corner $\mathbf{t}_j$ and mismatch condition $\mathbf{h}$. The term c_i represents the i-th target constraint. The function $\mathcal{F}(\mathbf{x}|\mathbf{t},\mathbf{h})$ nonlinearly maps $\mathbf{x}$ to performance metric, relying on SPICE simulation to reflect circuit's physical behavior. The PVT corner $\mathbf{t} \in T$ is an element of the predefined set T of PVT corner conditions. The vector $\mathbf{h} \in \widetilde{H}^N$ represents a r-dimensional mismatch condition, and the mismatch condition set $\widetilde{H}^N$ is obtained by randomly sampling N times from the distribution over $\mathbb{R}^r$, as follows:

$$\begin{aligned} \mathbf{h}^{(1)} &\sim \mathcal{N}(0, \Sigma^{Global}(\mathbf{x})) \\ \widetilde{H}^N &:= \{\mathbf{h}_1^{(2)}, \mathbf{h}_2^{(2)}, \ldots, \mathbf{h}_N^{(2)} \mid \mathbf{h}^{(2)} \sim \mathcal{N}(\mathbf{h}^{(1)}, \Sigma^{Local}(\mathbf{x}))\}. \end{aligned} \tag{3}$$

Eq. (3) outlines a hierarchical process for generating the set $\widetilde{H}^N$. Initially, a global variation sample $\mathbf{h}^{(1)}$ is drawn from a normal distribution $\mathcal{N}\big(0, \Sigma^{\text{Global}}(\mathbf{x})\big)$, where $\Sigma^{\text{Global}}(\mathbf{x})$ is a diagonal matrix containing the variances of each global process variation parameter. Subsequently, conditional on $\mathbf{h}^{(1)}$, local mismatch parameters $\mathbf{h}^{(2)}$ are sampled from $\mathcal{N}\big(\mathbf{h}^{(1)}, \Sigma^{\text{Local}}(\mathbf{x})\big)$. Here, $\Sigma^{\text{Local}}(\mathbf{x})$ is a diagonal matrix containing the variances of the device-specific variations, which depend on the design solution $\mathbf{x}$ [19]. Finally, the mismatch

TABLE I
OPERATIONAL CONFIGURATION OF THE FRAMEWORK

Verif.	Predefined Corner $\mathbf{t}$			Var. of Mismatch $\mathbf{h}$		# of Samples	
Method	P	V	T	Global	Local	Optim.	Verif.
C	Y	Y	Y	0	0	1	k
C-MC$_{\mathrm{L}}$	Y	Y	Y	0	Σ^{Local}	N'	$k \times N$
C-MC$_{\mathrm{G\text{-}L}}$	N	Y	Y	Σ^{Global}	Σ^{Local}	N'	$k \times N$

condition $\mathbf{h}$ is sampled by combining global and local variations for given design configuration [11], [15].

B. Operational Configuration

The GLOVA framework provides flexibility in selecting the target verification method (e.g., corner simulation, MC simulation, or a combination of both). With this adaptability, GLOVA efficiently adjusts the sampling methods and quantity during the optimization process. The configuration of the operational variables based on the chosen verification method is summarized in Table I.

C: Corner simulation. Predefined corners are considered without including mismatch conditions.

C-MC$_{\mathrm{L}}$: Corner and local MC simulation. Mismatch conditions are sampled from the local variation distribution for predefined corners.

C-MC$_{\mathrm{G\text{-}L}}$: Corner and global-local MC simulation. Mismatch conditions are sampled from the combined distribution of global and local variations for predefined corners.

C. Overall Workflow

An overview of the GLOVA framework is provided in Fig. 2. Initially, it utilizes TuRBO [13] to generate design solutions that meet constraints under the typical condition. This initial sampling is adopted from [9]. Each design solution is then simulated across sampled mismatch conditions under all PVT corners. The worst-case reward from these simulations is stored in the worst-case replay buffer, forming the initial dataset. Additionally, the last worst-case buffer records the last worst reward of each PVT corner. In each iteration, the following steps are executed. ❶ **Generate a design solution.** Put the last design solution into an actor to get a new design solution. ❷ **Sample PVT conditions.** Select the worst PVT corner by comparing the last worst-cases of corners, and sample N' mismatch conditions from the distribution via Eq. (3). ❸ **Simulate the design solution under the sampled conditions.** Get performance metrics under the sampled N' mismatch conditions given the worst PVT corner. ❹ **Evaluate the design solution via μ-σ metric. If it is decided not to verify further, go to Step 6.** ❺ **Fully verify with reordered PVT conditions.** Simulate the design solution under the targeted verification method. If the design meets all constraints in all cases, the framework terminates. Otherwise, continue the optimization process. ❻ **Store the worst reward in the replay buffer and update RL agent with data stored in the buffer.** Only the worst reward is stored across PVT conditions.

IV. OPTIMIZATION PHASE OF PROPOSED GLOVA

A. Risk-Sensitive Reinforcement Learning Agent

Risk-sensitive RL [18] is an approach that accounts for stochastic uncertainty within a system. To avoid uncertain and costly failures, agents are trained to maximize the worst-case robustness of the reward. This risk-sensitivity potentially leads to faster learning for unpredictable systems. In GLOVA, a risk-sensitive RL agent is employed to find the desired design solution in a situation where performance metrics fluctuate by various PVT conditions. By treating each variation as a potential risk and adopting a risk-avoidance policy, this approach reduces the cost of numerous simulations required by industrial-level verification methods, such as corner and MC simulations.

Algorithm 1: Risk-sensitive RL in GLOVA

Given replay buffer B^{worst} and the last design $\mathbf{x}_{last}$;
Given actor network $A(\mathbf{x}|\theta^A)$ and critic network $Q(\mathbf{x}|\theta^Q)$ with a set of base models $\{Q_i(\mathbf{x}|\theta_i^Q)\}$;
Given the number of samples N' and the worst corner $\mathbf{t}^{worst}$;
for *iteration = 1, M* **do**
  Sample a batch of $(\hat{\mathbf{x}}, \hat{r})$ from B^{worst};
  for *i = 1, ensemble size* **do**
    Update the base model by minimizing the loss: $\mathcal{L}_{Q_i} = MSELoss(\hat{r}, Q_i(\hat{\mathbf{x}}|\theta_i^Q) + bias)$;
  end
  Update the actor by minimizing the loss: $\mathcal{L}_A = MSELoss(0.2, Q(A(\hat{\mathbf{x}}|\theta^A)|\theta^Q) + bias)$;
  Select a new design according to the current policy and exploration noise: $\mathbf{x}_{new} = A(\mathbf{x}_{last}|\theta^A) + noise$;
  Sample mismatch conditions $\widetilde{H}^{N'}$ via Eq. (3);
  Simulate the $\{\mathbf{x}_{new}|\mathbf{t}^{worst}, \widetilde{H}^{N'}\}$ to get rewards $\{r\}$;
  Select the worst-case reward: $r^{worst} = \min\{r\}$;
  Store the data ($\mathbf{x}_{new}$,r^{worst}) in B^{worst};
end

Reward. The actor-critic agent, which is a widely used method in RL [20], is trained to optimize the multiple performance metrics to meet constraints. To simplify this process, we use a reward function that consolidates multiple objectives into a single target. The reward is defined as follows:

$$r = \begin{cases} r', & r' < 0 \\ 0.2, & r' \geq 0 \end{cases} \tag{4}$$

with r' calculated as:

$$r' = \sum_{i=1}^{m} \min\left(f_i, 0\right) \tag{5}$$

where f_i is the normalized current simulated i^{th} performance metric, defined as $f_i = (c_i - \mathcal{F}_i)/(c_i + \mathcal{F}_i)$, and c_i is the corresponding constraint. As we want $f_i \leq c_i$, a smaller reward indicates a worse design. If all constraints are satisfied, the reward is set to 0.2. This reward formulation is modified from [8], [9].

Actor and critic. The actor is a 4-layer neural network. The actor's input is the previous design $\mathbf{x}_{last}$, a p-dimensional normalized design solution, where each dimension represents a design parameter (e.g. width or length in a transistor). The output of the actor is the next design $\mathbf{x}_{new}$, a p-dimensional normalized design solution. On the other hand, the critic includes a set of 4-layer neural networks, which are called base models. The input of the critic is also a p-dimensional normalized design solution. The critic outputs a scalar value representing the input's predicted reliability bound under various PVT conditions. The process of predicting uncertain design reliability bounds using the base models is detailed in Section IV. B.

Training. Algorithm 1 outlines agent's training procedure. Here, M is the maximum number of optimization iterations. The training process is modified from DDPG [21]. Unlike risk-neutral training, the risk-avoidance process evaluates each design iteration based on worst-case scenarios under sampled PVT conditions. In other words, while multiple rewards are obtained from simulations, only the worst

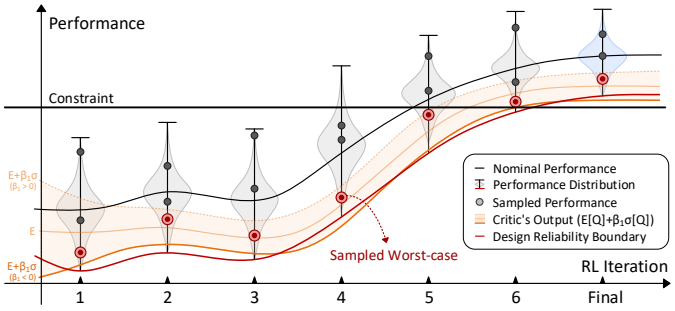


Fig. 3. Design reliability bound estimation under PVT variations using an ensemble-based critic trained on sampled worst-case scenarios.

reward is stored in the replay buffer and used for training. Each of the critic's base models is independently trained with a distinct batch sampled from the replay buffer. The actor is trained to find the desired design solution using the predicted reliability bound, which is derived by aggregating the outputs of the critic's base models.

B. Ensemble-based Critic

In risk-sensitive RL, modeling worst-case scenarios is crucial, yet impractical due to the need for over 1,000 samples per iteration to determine reliability bounds under PVT variations [22], [23]. Therefore, to achieve sample-efficient risk management, we propose an ensemble-based critic that approximates design reliability bounds for variation by sampling worst cases. This approach allows the actor to explore a wide range of variations without simulating each scenario individually, improving sample-efficiency.

The proposed critic extends the risk-sensitive criterion [18] by employing an ensemble of base models. Fig. 3 qualitatively illustrates the modeling of the design reliability boundary with the ensemble-based critic during the optimization process. First, in each iteration, a small number N' (typically 2 to 5) of mismatch conditions $\widetilde{H}^{N'}$ are sampled and simulated under the worst-case corner to capture performance variability. The critic then learns from the worst-case among N' sampled variations, using an ensemble to estimate design reliability bound with awareness of uncertainty. Each base model is trained with different data batches, benefiting from randomness and varying initialization. This process allows the critic to compensate for the uncertainty in the design reliability bound caused by limited statistical samples. The output of critic is defined as follows:

$$Q(\mathbf{x}|\theta^Q) = \mathbb{E}[Q_i(\mathbf{x}|\theta_i^Q)] + \beta_1 \sigma[Q_i(\mathbf{x}|\theta_i^Q)] \tag{6}$$

where $Q_i(\mathbf{x}|\theta_i^Q)$ represents the output of the i^{th} base model with weights θ_i^Q, $\mathbb{E}[\cdot]$ denotes the average of the base model outputs, $\sigma[\cdot]$ refers to the standard deviation of the base model outputs, and $\beta_1 < 0$ is the risk-avoidance parameter. The critic manages risk and provides reliability bounds to guide the actor in finding a feasible design solution.

V. Verification Phase of Proposed GLOVA

Extensive simulations, which are time-intensive, are inevitable to ensure circuit reliability under PVT variations. If a design fails to meet its target during the verification process, a large number of simulations can become an inefficient use of resources. Therefore, to achieve sample-efficient verification, it is crucial to detect failures early and halt the verification process, returning to the optimization phase. To address this, we propose a hierarchical verification algorithm comprising the μ-σ evaluation and the simulation reordering method. The verification workflow of GLOVA is presented in Algorithm 2.

Algorithm 2: Verification Algorithm of GLOVA

```
Given the number of samples N for full verification and a
  subset of samples N';
Sort T based on the last worst-case buffer;
for each t_j in sorted T do
    Sample H̃_j^{N'} from the distribution via Eq. (3);
    Simulate {x|t_j, H̃_j^{N'}} to obtain {r, f_i(x|t_j, H̃_j^{N'})};
    if μ-σ evaluation passes then
        Calculate t-SCORE_j;
        Calculate Pearson correlation coefficient vector ρ_j;
    end
    else
        Verification failed;
    end
end
Sort T by {t-SCORE_j};
for each t_j in sorted T do
    Sample H̃_j^{N-N'} from the distribution via Eq. (3);
    Calculate h-SCORE_{j,n} for each h_{j,n} ∈ H̃_j^{N-N'} and ρ_j;
    Sort H̃_j^{N-N'} by {h-SCORE_n}_j;
    for each h_{j,n} in sorted H̃_j^{N-N'} do
        Simulate {x|t_j, h_{j,n}} to obtain {r, f_i(x|t_j, h_{j,n})};
        if r ≠ 0.2 then
            Verification failed;
        end
    end
end
```

A. μ-σ Evaluation Method

The μ-σ evaluation first analyzes a subset N' of the total N MC simulations under the given corner conditions. Based on this analysis, it determines whether it is worthwhile to proceed with the remaining $N - N'$ simulations for full verification. Specifically, it statistically estimates the performance distribution of the full mismatch condition set $\widetilde{H}^N$ using the pre-sampled subset $\widetilde{H}^{N'}$. The μ-σ evaluation is conducted sequentially across all given corners, starting with the worst corner from the last worst-case buffer. Note that, during the optimization phase, the $\widetilde{H}^{N'}$ for the worst corner has already been simulated and can be reused. If the evaluation fails to pass the criteria set in Eq. (7), the design is deemed to have failed verification. The evaluation criterion is provided by the following equation:

$$e_i = \mathbb{E}[f_i] + \beta_2 \sigma[f_i] \leq c_i \tag{7}$$

where f_i represents the i^{th} normalized performance metric and β_2 is a reliability factor. The reliability factor β_2 is set to a positive value of 4 or higher, as higher values in the performance metric, unlike rewards, indicate worse performance. This reliability factor compensates for the incomplete nature of the distribution caused by a lack of samples. The μ-σ evaluation method conservatively assesses whether a given design is feasible for full verification, reducing the likelihood of verification attempts that would ultimately result in failure. Consequently, this approach saves valuable resources and reduces overall runtime.

B. Simulation Reordering Method

If the design passes the μ-σ evaluation for the N' samples across all given corners, full verification is performed for the remaining $N - N'$ samples for each corner. To detect failure early and halt

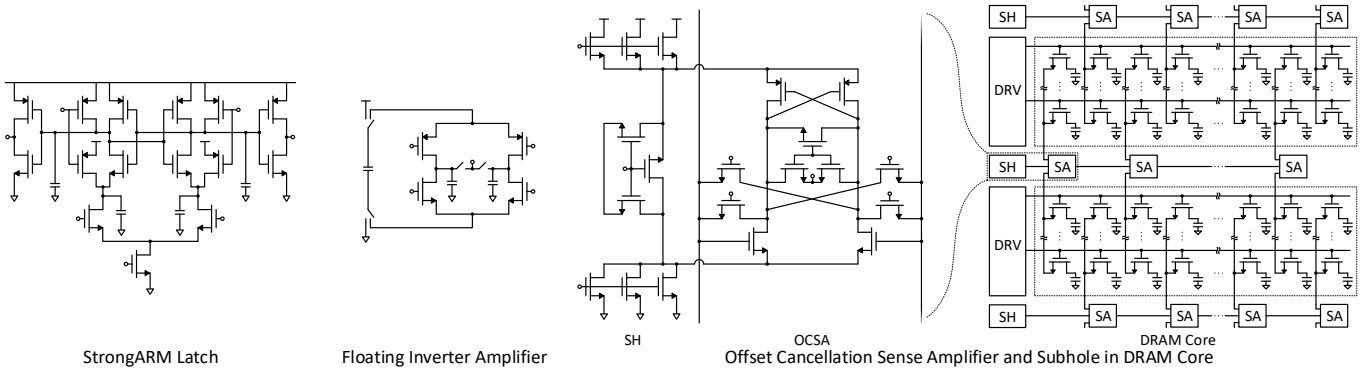


Fig. 4. Schematics of three analog/mixed-signal testcase circuits.

the verification process, the simulation sequence is determined by the proposed corner and MC reordering methods, prioritizing simulations with a higher likelihood of failure.

Corner reordering. For the pre-sampled mismatch condition set $\widetilde{H}_j^{N'}$ of each corner $\mathbf{t}_j$, the $\texttt{t-SCORE}_j$ is calculated to rank the severity of degradation caused by mismatch conditions as follows:

$$\texttt{t-SCORE}_j = \sum_{i=1}^{m} e_{j,i} \tag{8}$$

where e_i is computed by Eq. (7). Corners with a higher t−SCORE, indicating a greater potential for failure, are selected for simulation first.

MC reordering. For the given corner, the simulation list for the remaining mismatch condition set $\widetilde{H}_j^{N-N'}$ is determined. To prioritize the mismatch conditions most likely to result in failure, the Pearson correlation coefficient is introduced to assess the relationship between the mismatch parameters and performance. Specifically, for each $\mathbf{h}_{j,n}$ vector in the pre-sampled set $\widetilde{H}_j^{N'}$, the Pearson correlation coefficient vector ρ_j is calculated between each element of $\mathbf{h}_{j,n}$ and the corresponding performance metric $g_{j,n}$, where $g = \sum_i f_i$, as defined by the following equation:

$$\rho_j = \frac{\sum_{n=1}^{N'}(\mathbf{h}_{j,n} - \bar{\mathbf{h}_j})(g_{j,n} - \bar{g_j})}{\sqrt{\sum_{n=1}^{N'}(\mathbf{h}_{j,n} - \bar{\mathbf{h}_j})^2}\sqrt{\sum_{i=1}^{N'}(g_{j,n} - \bar{g_j})^2}}. \tag{9}$$

Then, the $\texttt{h-SCORE}_{j,n}$ is defined as the weighted sum of the mismatch condition vector $\mathbf{h}_{j,n}$ from the mismatch condition set $\widetilde{H}_j^{N-N'}$ and correlation vector ρ_j, as follows:

$$\texttt{h-SCORE}_{j,n} = \sum_{i=1}^{r} \left[(\mathbf{h}_{j,n})_i \circ (\rho_j)_i\right]. \tag{10}$$

Mismatch conditions with a higher h−SCORE, indicating a greater potential for failure, are prioritized for simulation for the given corner.

By reordering both the corners and mismatch conditions, failures in the verification process can be detected early, effectively reducing the number of unnecessary simulations. This simulation reordering method reduces the verification cost and improves the overall efficiency of the framework.

VI. Results

A. Analog/mixed-signal Circuits

Real-world designs from previous work [8], [9] are adopted as testcases to compare the design efficiency of GLOVA. The testcases include the strongARM latch (SAL) [24] and the floating inverter amplifier (FIA) [25]. These circuits are selected due to their fully dynamic operation, which makes them highly sensitive to PVT variations. Additionally, we include the offset cancellation sense amplifier (OCSA) [26] and subhole (SH) in DRAM core with an $6F^2$ open bitline architecture, which consists of 2K wordlines and peripherals for memory operation [27]. This testcase is particularly challenging due to large parasitic array capacitance and extensive mismatches, which necessitate a significantly higher number of statistical simulations to achieve a high yield. The topologies are presented in Fig. 4.

All testcases are designed using advanced $28\mathrm{nm}$ CMOS technology and simulated with a SPICE-based simulator [15] under 30 PVT conditions, given by $\{\mathrm{TT}, \mathrm{SS}, \mathrm{FF}, \mathrm{SF}, \mathrm{FS}\} \times \{0.8\mathrm{V}, 0.9\mathrm{V}\} \times \{-40\,^\circ\mathrm{C}, 27\,^\circ\mathrm{C}, 80\,^\circ\mathrm{C}\}$. Each circuit comprises multiple transistors and capacitors, where the transistors are defined by parameters such as gate width and length, and the capacitors by their capacitance values. Mismatch parameters are considered for each device, with variances following the PDK rules for the same technology.

StrongARM latch. The sizing vector of this circuit consists of 14 parameters: 6 transistor widths, 6 transistor lengths, and 2 capacitances. The range for each parameter is $[0.28, 32.8]\,\mu\mathrm{m}$ for width, $[0.03, 0.33]\,\mu\mathrm{m}$ for length, and $[0.005, 5.5]\,\mathrm{pF}$ for capacitance with a total design space of 10^{28}. The mismatch parameter includes all devices in the circuit. The performance metrics are power, set delay, reset delay, and noise. The design targets are as follows, which are the same as [9]:

$$c = \begin{cases} \text{Power} \leq 40\,\mu\mathrm{W} \\ \text{Set delay} \leq 4\,\mathrm{ns} \\ \text{Reset delay} \leq 4\,\mathrm{ns} \\ \text{Noise} \leq 120\,\mu\mathrm{V}. \end{cases}$$

Floating inverter amplifier. The sizing vector of this circuit consists of 6 parameters: 2 transistor widths, 2 transistor lengths, and 2 capacitances. The range for each parameter is the same as those of the strongARM latch with a total design space of 10^{12}. The mismatch parameter includes all devices in the circuit. The performance metrics are energy consumption per conversion and noise. The design targets are as follows, taking technology scaling into account in [9]:

$$c = \begin{cases} \text{Energy/conv.} \leq 0.1\,\mathrm{pJ} \\ \text{Noise} \leq 130\,\mathrm{mV}. \end{cases}$$

Offset cancellation sense amplifier and subhole in DRAM core. The sizing vector of this circuit consists of 12 parameters: 6 transistor widths and 6 transistor lengths. The range for each parameter is $[0.28, 1.028]\,\mu\mathrm{m}$ for the transistor width in the OCSA, $[5, 15]\,\mu\mathrm{m}$ for the transistor width in the SH, and $[0.03, 0.06]\,\mu\mathrm{m}$ for all transistor length, resulting in a total design space of 10^{24}. These

TABLE II
OPTIMIZATION RESULTS ON REAL-WORLD CIRCUITS

Testcases		SAL			FIA			OCSA and SH in DRAM Core		
Verification		C	C-MC$_L$	C-MC$_{G\text{-}L}$	C	C-MC$_L$	C-MC$_{G\text{-}L}$	C	C-MC$_L$	C-MC$_{G\text{-}L}$
RL Iteration	Ours	6	8	12	18	26	48	21	84	129
	PVTSizing	19	24	27	48	71	138	72	138	238*
	RobustAnalog	104	124	297	533	840*	1,733*	760	1,166*	2,064*
# Simulation	Ours	83	3,103	8,809	248	3,203	6,461	390	6,916	72,853
	PVTSizing	186	10,748	31,221	322	87,773	293,076	2,066	300,332	224,768*
	RobustAnalog	442	12,683	75,920	2,151	146,889*	361,066*	6,406	557,050*	753,048*
Norm. Runtime	Ours	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	PVTSizing	2.77	3.45	3.81	1.71	26.28	43.53	3.85	40.59	3.07*
	RobustAnalog	11.17	4.43	9.63	14.94	45.26*	55.02*	21.24	76.03*	10.40*
Success Rate	Ours	100%	100%	100%	100%	100%	100%	100%	100%	100%
	PVTSizing	100%	100%	100%	100%	100%	100%	100%	100%	87%
	RobustAnalog	100%	100%	100%	100%	95%	90%	100%	83%	53%

*In tests where the success rate is below 100%, only data from successful optimizations are included.

TABLE III
ABLATION STUDY

Verification		C	C-MC$_L$	C-MC$_{G\text{-}L}$
RL Iteration	Proposed	21	84	129
	w/o EC[1]	26	92	199*
	w/o μ-σ[2]	-	101	239*
	w/o SR[3]	-	-	-
# Simulation	Proposed	390	6,916	72,853
	w/o EC	1,218	18,232	212,153*
	w/o μ-σ	-	136,217	476,721*
	w/o SR	2,448	253,738	765,375*
Norm. Runtime	Proposed	1.00	1.00	1.00
	w/o EC	3.75	3.02	3.32*
	w/o μ-σ	-	21.97	7.45*
	w/o SR	7.05	40.80	11.93*
Success Rate	Proposed	100%	100%	100%
	w/o EC	100%	100%	90%
	w/o μ-σ	100%	100%	95%
	w/o SR	100%	100%	95%

*In tests where the success rate is below 100%, only data from successful optimizations are included.
[1]Ensemble-based critic, [2]μ-σ evaluation, [3]Simulation reordering.

parameter ranges are determined with consideration for cell pitch, as the transistors need to be integrated near the cell array. The mismatch parameter includes all devices in the DRAM core. The simulation methods and metrics are based on [27]. The performance metrics are low data sensing voltage ΔV_{D0}, high data sensing voltage ΔV_{D1}, and energy consumption per 1-bit sensing. Note that since both the low and high data sensing voltages are metrics that need to be maximized, their signs are inverted. The design targets are as follows:

$$c = \begin{cases} \Delta V_{D0} \geq 85\,\mathrm{mV} \;\rightarrow -\Delta V_{D0} \leq -85\,\mathrm{mV} \\ \Delta V_{D1} \geq 85\,\mathrm{mV} \;\rightarrow -\Delta V_{D1} \leq -85\,\mathrm{mV} \\ \text{Energy/bit} \leq 30\,\mathrm{fJ}. \end{cases}$$

B. Evaluation and Analysis

To validate GLOVA's design solution across various PVT corner conditions and extensive mismatches, we incorporate three verification scenarios: 30 PVT corner simulations (C), 0.1K local Monte Carlo simulations on 30 PVT corners (C-MC$_L$), and 1K global-local Monte Carlo simulations on 6 VT corners (C-MC$_{G\text{-}L}$). Each method requires 30, 3,000, and 6,000 simulations, respectively, to complete full verification. In our setting, simulations are conducted in parallel with a sample size of 3 during the optimization phase, while the verification phase utilizes the maximum available resources. For training, the batch size is set to 10, and the risk-avoidance parameter β_1 and reliability factor β_2 are set to -3 and 4, respectively.

The optimization results across three real-world circuits are shown in Table II. GLOVA achieves the highest success rate, with up to 80.5× greater sample efficiency and 76.0× lower time consumption compared to PVTSizing and RobustAnalog. Notably, for the OCSA and SH in the DRAM core case, which is verified using the C-MC$_{G\text{-}L}$ method, GLOVA demonstrates 1.9× improvement in success rate and 10.3× improvement in sample efficiency over PVTSizing and RobustAnalog. However, this case appears to consume more RL iterations and simulations than other cases. This is due to the challenge of designing a solution that simultaneously satisfies low data sensing voltage and high data sensing voltage—two conflicting metrics—in scenarios where the DRAM core circuit is highly sensitive to mismatch cases across numerous devices within the cell array.

Table III presents the results of an ablation study conducted on a DRAM core to assess the contributions of the proposed methods. The ensemble-based critic facilitates the derivation of feasible design solutions during the optimization process, improving in both success rate and sample efficiency. Additionally, the μ-σ evaluation and simulation reordering methods effectively reduce the number of simulations required in the verification process, thereby lowering verification costs and enhancing the overall efficiency of the framework.

VII. CONCLUSION

We present GLOVA, an efficient optimization and verification framework that effectively manages diverse PVT conditions and extensive mismatches. GLOVA employs risk-sensitive reinforcement learning and introduces an ensemble-based critic for variation-aware optimization. Incorporating μ-σ evaluation and simulation reordering methods significantly alleviates the simulation burden during verification, improving the overall efficiency of the framework. The proposed GLOVA supports industrial-level corner and Monte Carlo simulations for design verification. Experimental results on real-world circuits, including challenging DRAM core, demonstrate that GLOVA substantially reduces simulation and time costs compared to previous state-of-the-art frameworks.

ACKNOWLEDGMENT

This paper is result of the research project supported by SK hynix Inc.

REFERENCES

[1] T. Pessoa et al., "Enhanced analog and rf ic sizing methodology using pca and nsga-II optimization kernel," in Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 660-665.

[2] Y. Li et al., "An artificial neural network assisted optimization system for analog design space exploration," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, vol. 39, no. 10, pp. 2640-2653.

[3] M. Liu et al., "Parasitic-aware analog circuit sizing with graph neural networks and bayesian optimization," in Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1372-1377.

[4] B. He et al., "An efficient bayesian optimization approach for analog circuit synthesis via sparse gaussian process modeling," in Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1372-1377.

[5] S. Zhang et al., "An efficient batch-constrained bayesian optimization approach for analog circuit synthesis via multiobjective acquisition ensemble," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, vol. 41, no. 1, pp. 1-14.

[6] H. Wang et al., "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in Proceedings of the Design Automation Conference (DAC), 2020, pp. 1-6.

[7] W. Shi et al., "Trust-region method with deep reinforcement learning in analog design space exploration," in Proceedings of the Design Automation Conference (DAC), 2021, pp. 1225-1230.

[8] B. He et al., "Robustanalog: Fast variation-aware analog circuit design via multi-task rl," in Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD), 2022, pp. 35-41.

[9] Z. Kong et al., "Pvtsizing: A turbo-rl-based batch-dampling optimization framework for pvt-robust analog circuit synthesis," in Proceedings of the Design Automation Conference (DAC), 2024.

[10] K. Kuhn et al., "Process technology variation," IEEE Transactions on Electron Devices, 2011, vol. 58, no. 8, pp. 2197-2208.

[11] S. Dongaonkar et al., "From process corners to statistical circuit design methodology: Opportunities and challenges," IEEE Transactions on Electron Devices, 2018, vol. 66, no. 1, pp. 19-27.

[12] Y. Yang et al., "Smart-msp: A self-adaptive multiple starting point optimization approach for analog circuit synthesis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017, vol. 37, no. 3, pp. 531-544.

[13] D. Eriksson et al., "Scalable global optimization via local bayesian optimization," in Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2019, vol.32, pp. 5496-5507.

[14] C. McAndrew et al., "Statistical modeling for circuit simulation," in Proceedings of the International Symposium on Quality Electronic Design (ISQED), 2003. pp. 357-362.

[15] Synopsys, "Hspice® user guide: Simulation and analysis," 2008.

[16] P. Geibel et al., "Risk-sensitive reinforcement learning applied to control under constraints," Journal of Artificial Intelligence Research, 2005, vol. 24, pp. 81-108.

[17] P. Geibel et al., "Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning," in Proceedings of the International Conference on Machine Learning (PMLR), 2018. pp. 1184-1193.

[18] J. Garcıa et al., "A comprehensive survey on safe reinforcement learning," Journal of Machine Learning Research, 2015, vol. 16, no.1, pp. 1437-1480.

[19] P.G. Drennan et al., "Understanding mosfet mismatch for analog design," IEEE Journal of Solid-State Circuits, 2003, vol. 38, no. 3, pp. 450 - 456.

[20] V. Konda et al., "Actor-critic algorithms," in Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 1999, vol.12, pp.1008-1014.

[21] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, arXiv:1509.02971.

[22] C.-C. Wang, "Tutorial: Design of high-speed nano-scale cmos mixed-voltage digital i/o buffer with high reliability to pvt variations," IEEE Transactions on Circuits and Systems II: Express Briefs, 2021, vol. 68, no. 2, pp. 562–567.

[23] K. Chen et al., "An improved mos self-biased ring amplifier and modified auto-zeroing scheme," IEEE Transactions on Very Large Scale Integration Systems, 2023, vol. 31, no. 4, pp. 606-610.

[24] B. Razavi, et al., "The strongarm latch [a circuit for all seasons]," IEEE Solid-State Circuits Magazine, 2015, vol. 7, no. 2, pp. 12 - 17.

[25] X. Tang, et al., "An energy-efficient comparator with dynamic floating inverter amplifier," IEEE Journal of Solid-State Circuits, 2020, vol. 55, no. 4, pp. 1011 - 1022.

[26] Y. Seo, "Sensor amplifier, memory device comprising same, and related method of operation," U.S. Patent No. 9,202,531. 1 Dec. 2015.

[27] S. Kim et al., "Sensing margin enhancement technique utilizing boosted reference voltage for low-voltage and high-density DRAM," IEEE Transactions on Very Large Scale Integration Systems, 2019, vol. 27, no. 10, pp. 2413 - 2422.

REMU: Memory-aware Radiation Emulation via Dual Addressing for In-orbit Deep Learning System

Longnv Xu†, Meiqi Wang†, Han Qiu*, Jun Liu, Yuanjie Li, Hewu Li*
Tsinghua University, Beijing, China
Email: {xln23@mails., qiuhan@}tsinghua.edu.cn

***Abstract*—The deployment of commercial-off-the-shelf (COTS) GPUs in space has emerged as a promising approach for supporting in-orbit deep neural network (DNN) inference. However, unlike terrestrial environments, understanding the impact of space radiation on COTS GPU-enabled DNNs is critical. This is challenging because existing methods, such as real-world radiation testing and software emulation, fail to link radiation-induced memory errors to runtime DNN behaviors. In this paper, we propose `REMU`, a memory-aware Radiation EMUlator to fill this gap. `REMU` introduces a dual addressing mechanism across virtual, physical, and DRAM memory spaces, enabling precise mapping and efficient injection of radiation-induced errors from DRAM to runtime DNN inference. Extensive evaluations across 10 well-known DNN models and 2 typical in-orbit computing tasks demonstrate the effectiveness of `REMU`, providing valuable insights for understanding the resilience of runtime DNN inferences on space radiations.**

***Index Terms*—Deep learning, AI on satellites, memory emulation.**

I Introduction

In-orbit computing [1], [2] has emerged as a novel form of "edge computing", where data is processed directly on satellites to reduce downlink bandwidth [3], [4] or to perform latency-sensitive tasks in orbit, such as wildfire monitoring [5]. In order to support complex data processing tasks like deep neural networks (DNNs), a promising trend [6] is to launch commercial-off-the-shelf (COTS) GPUs into space (see statistics from public information in Fig. 1) instead of relying on radiation-harden chips which are not only 10× more expensive [7] but also unable to support DNN inference [8].

However, different from terrestrial edge computing scenarios, COTS GPUs (e.g., NVIDIA NX) without radiation hardening are vulnerable to radiation-induced errors in space. On the one hand, both ESA's 10-year in-orbit experimental records [9] and various ground-based radiation tests [10], [11] have demonstrated that space radiation particles can induce bit flips in COTS memory. These include single-event upsets (SEUs), which flip a single bit, and multiple-cell upsets (MCUs), which flip multiple spatial-correlated bits, occurring at a non-negligible rate (e.g., at least 1×10^{-7}/bit/day [12]). On the other hand, runtime DNN models are especially susceptible since their memory-intensive data flow amplifies the probability of error accumulation during critical computations. Current efforts to investigate radiation-induced errors can be classified into two approaches. (1) Real-world validation rely on beam tests [11] or in-orbit flight experiments [12], which requires significant resources and time (e.g., years in [9]), to obtain physical error distributions on memory cells. However, there is still a huge gap between memory hardware errors and the DNN inference behaviors. (2) Software-based error injection offers a cost-effective alternative [13] but existing tools cannot be directly adopted to evaluate how radiation-induced errors influence COTS GPU-based DNN inference. Particularly, they either rely on complex instrumentation close to the hardware [14]–[16] or adopt naïve error injection in static DNN models [17]–[19], regardless of the characteristics of hardware errors and the runtime dynamics of DNN inference. Therefore, we summarize the challenge as follows: *how to rigorously understand the radiation-induced errors' impact on those COTS GPU-enabled runtime DNN's performance.*

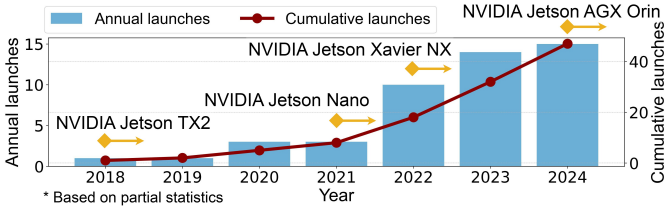


Fig. 1: COTS GPUs launched into space in recent years.

In this paper, we tackle the above challenge by proposing `REMU`, a memory-aware Radiation EMUlator that bridges radiation-induced memory hardware errors to runtime DNN's behavior. (1) `REMU` can *precisely* trace memory bit errors across the system (i.e., from low-level DRAM to high-level process space) to realize an end-to-end error mapping. (2) To *efficiently* navigate error tracking in massive DNN memory usage, we develop a novel dual-addressing mechanism using bitmap-based trees, reducing error injection overhead from $\sim 100\times$ to $\sim 3\times$. (3) `REMU` supports mainstream satellite payload platforms (e.g., TensorRT on NVIDIA Jetson series [20]) but can also be *extended* with other configurations.

Based on `REMU`, we investigate how COTS GPU-enabled DNN inference's performance is influenced by radiation-induced memory errors with 10 well-known DNN models on 2 typical in-orbit computing tasks. Our 3 key findings are as follows. (1) Different DNN model structures exhibit different inherent error-tolerance to radiation-induced errors (i.e., lightweight models such as EfficientNet [21] are more sensitive while compacted models such as ViT [22] are robust). (2) Some uncovered sensitivities in runtime DNN inference can lead to a more severe accuracy collapse (i.e., degrading decision-making to a random guess) than the DNN model's parameter errors. (3) Hardware-layer spatial-correlated errors (i.e., MCU) in memory *cannot* be simplified as multiple single-bit errors (i.e., SEU) since the sensitivity in DNN inference manifests as clustered bits.

II Preliminaries

A. Space radiation is a threat

Harsh space radiation has emerged as a potent "adversary" for in-orbit computing. Satellites in space are surrounded by radiation particles mainly consisting of trapped protons, secondary neutrons from nuclear interactions, and high-energy cosmic rays [13]. Radiation affects in-orbit electronic components, especially the COTS devices that are rarely hardened for lower cost, causing permanent damage (e.g., degradation and burning) or transient damage (e.g., logical state flip) in the unprotected memory[1] [24]. The transient damages,

†Co-first authors.
*Corresponding authors.

[1]The main memory chips in current in-orbit COTS devices mostly lack support for error-correcting code (ECC) and are not protected by other hardening techniques, such as triple modular redundancy and shielding [23].

although seemingly trivial, are more prevalent and threatening for in-orbit computing since they can stealthily inject wrong control logic and data values into the computing process and disrupt the outcomes, such as the decision-making results of DNN inference. These errors in memory can be classified into two categories.

Single-event upset (SEU) is induced by ionizing radiation striking that makes electron-hole pairs in transistor gates generate and diffuse, causing a state alteration in a memory cell, i.e., a single-bit error. The probability of SEUs can reach 10^{-7} to 10^{-6}/bit/day [9]. Even one SEU may damage the performance of in-orbit DNN inference.

Multiple-cell upset (MCU) occurs when neutron particles react with impurities and produce a large number of secondary particles with sufficient energy [25], leading to multiple memory cells flipped simultaneously as shown in Fig. 2, i.e., a spatial correlated multi-bit error. In space, MCUs happen more frequently in COTS due to the advanced nanotechnology and more compact chips in size [25].

Existing research [26] has shown that the reliability of DNN inference is fragile as bit-flips can result in *stealthy* wrong inference results without crashes, which may compromise safety-critical in-orbit applications, such as disaster monitoring and ship detection [27]. Thus, a practical emulation of SEU/MCU errors in DNN inference is significant to discover its sensitivity at scale.

B. Memory system of COTS GPUs on satellites

To emulate the radiation-induced errors, especially spatial correlated MCUs from the low-level memory cells to the upper process space of DNN inference, precise memory system modeling is essential. We consider a commonly-used payload, NVIDIA Jetson Xavier NX [28] with 16GB 128-bit LPDDR4 (JESD209-4 [29]) as its main memory. The modeling bridges the process space to the dynamic random access memory (DRAM) via hierarchical addressing. First, the memory management unit (MMU) translates virtual addresses in the process space into physical addresses by mapping virtual pages to physical frames using memory paging. Second, the memory controller (MC) translates physical addresses into DRAM and accesses the data.

As shown in Fig. 2, DRAM typically consists of six hierarchies, including die, channel, rank, bank, array, and cell. LPDDR4 [29] defines dual-channel dies with 16DQ (data I/O pins for parallel data transfer) per channel of varying memory densities, such as 16Gb. To enhance data parallelism, multiple dual-channel dies can be employed (e.g., 128-bit LPDDR4 utilizes 8 dual-channel dies). Each DARM channel consists of multiple banks, and each bank contains three-dimensional arrays: row-, column-, and DQ-dimension. DRAM cells are organized into arrays, with each cell using a capacitor to store logical states (charging to represent logic 1 and discharging to represent logic 0) and a transistor to control access to the capacitor during read and write operations. Besides, MCU's adjacency manifests in three dimensions: consecutive rows sharing the same bitline, consecutive columns sharing the same wordline, and consecutive DQ bits in a logical word.

C. Existing emulation methods

The practical emulation of radiation-induced errors requires addressing the complex interaction between hardware errors and application behavior, making it challenging to balance *efficiency* and *extensibility*. Most existing emulations focus on injecting errors closer to the hardware (mainly targeting registers and computing pipelines rather than DRAM), such as at the intermediate representation [14], the low-level assembly [15], or the microarchitecture [16]. Although these methods are extensible to most upper applications, the complexity of use leads to inefficient error injection and limits their adaptability for sensitivity analysis of runtime DNN inference.

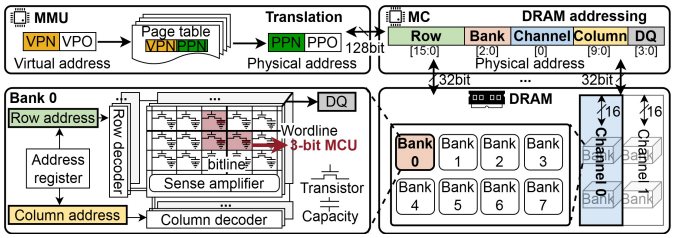


Fig. 2: Hierarchical addressing from process space to DRAM by two steps: translation and DRAM addressing. Detailed organization in a DRAM bank shows MCU's adjacency in three dimensions.

To be more specialized, many error injection tools tailored for deep learning (DL) frameworks have been proposed with expert knowledge, such as TensoFI [17], PyTorchFI [18], and Ares [19]. These methods inject random bit-flip errors directly in DNN models' weights and feature maps for different DL frameworks (e.g., TensorFlow [30] and PyTorch [31]). Although these methods are efficient due to targeted injection in small-scale values, the *offline* model-level error injection lacks critical insights into the runtime process space, such as optimized data flow and actual memory allocation. Thus, it fails to extend to other applications and also misses sensitive points in DNN inference. Besides, all existing methods overlook the impact of spatial correlated MCUs in memory, causing a lack of *correctness*. Thus, we fill the gap to balance the efficiency and extensibility, and also intensify the correctness in our emulation (Sec.III).

III REMU framework

A. Overview

To explore reliable DNN inference under space radiation, emulation of radiation-induced bit-flip errors (i.e., SEUs/MCUs) is imperative. However, existing error injection tools [17]–[19] overlook the spatial correlation of MCUs, leading to impractical error models. To address the gap, we propose `REMU` (see overview of `REMU` in Fig. 3) to precisely trace SEU/MCU errors from low-level DRAM to high-level process space. `REMU` has three main goals to achieve. (1) Correctness: SEU/MCU errors occurring in the DRAM must be precisely addressed in the upper DNN inference (Sec.III-B). (2) Efficiency: error injection should be fast to facilitate large-scale testing (Sec.III-C). (3) Extensibility: the emulation should extend to various DRAMs and radiation-induced error models (Sec. III-D).

B. DNN to DRAM mapping

To precisely inject SEU/MCU errors during runtime DNN inference, the DNN should be first mapped to DRAM. We observe that the process space of DNN inference is vast and radiation-induced SEU/MCU errors are unpredictable. Thus, we simplify the mapping by focusing exclusively on the *inference-critical*, *time- and memory-intensive* DNN engine (i.e., an optimized runtime DNN model used to execute inference, more details are in Sec.IV-A), which is allocated in the main memory. We realize the mapping via two steps.

First, we leverage the kernel interface (e.g., */proc/pid/pagemap*[2]) to translate the userspace virtual addresses to physical addresses (❶ in Fig. 3). In modern memory management, such as memory paging, allocations are separate due to the fragmented and limited physical memory left when the process is running. Thus, a contiguous virtual memory space of the DNN engine is typically allocated to scattered physical frames (constituting a huge space) with the order varying

[2]https://www.kernel.org/doc/Documentation/vm/pagemap.txt in Linux.

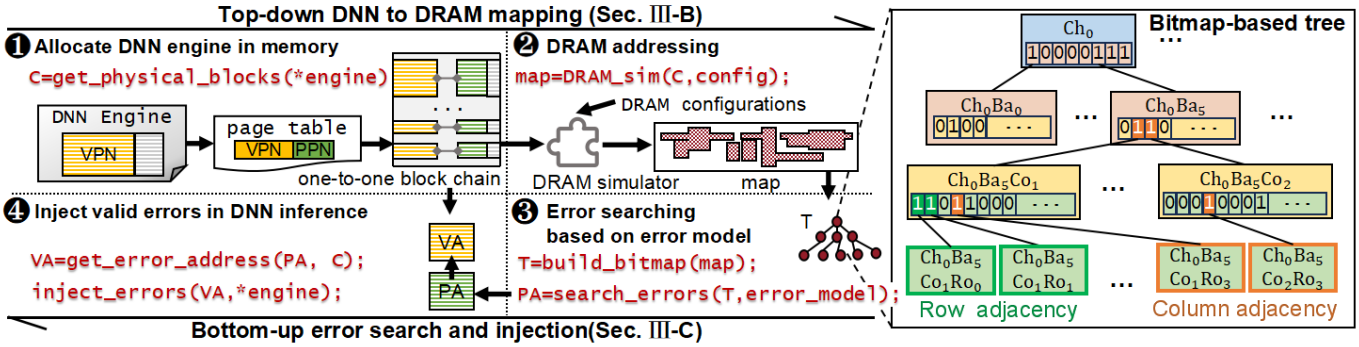


Fig. 3: Overview of REMU for four-step error injection: ❶ Allocate DNN engine's process space to physical blocks and construct a one-to-one mapping chain of virtual and physical blocks. ❷ Configure the DRAM simulator and perform DRAM addressing based on the addresses in physical blocks to get a vast memory space (i.e., map). ❸ Build the bitmap-based tree based on the physical blocks and search spatial adjacent errors in DRAM via traversal. ❹ Reveresely convert valid error physical addresses (PA) to DNN inference's process space (VA).

from the virtual pages. We construct a one-to-one mapping chain of virtual and physical blocks via the process-context page table.

Second, we leverage the correctness-verified extensible DRAM simulator, Ramulator [32], to address the translated physical memory space in DRAM (❷ in Fig. 3). This benefits from generalized modeling of the DRAM system and extending by configurable DRAM standards (e.g., LPDDR4 [29] in this paper) and DRAM addressing. We improve it to further support the emulation of error injection in DRAM. Specifically, we determine the range of each DRAM level based on the target physical memory space. For example, in a 4KB space, only the lowest 12 bits of the physical address serve for addressing, typically containing 2^0 row and 2^8 columns in LPDDR4. Then, SEUs/MCUs can be emulated in the range and flip the corresponding bit in the virtual memory space. The above two-step extraction from a real memory system can achieve the correct mapping from DNN inference to DRAM.

C. Dual addressing

When executing DNN inference, memory accesses follow a top-down manner from the virtual memory space, translating to the physical memory space, and finally addressing the DRAM cells (Sec.III-B). However, when radiation-induced errors occur in DRAM, their impact on DNN inference flows in an opposite direction. The fragmented physical blocks cause those random errors to get lost in the wilderness (i.e., a vast physical memory space irrelevant to DNN inference). Thus, REMU introduces a dual addressing mechanism utilizing bitmap-based trees to optimize the addressing process and narrow the searching space of relevant errors, which can efficiently inject SEU/MCU errors in DNN inference among the tremendously vast physical memory space (❸ in Fig. 3).

Bitmap-based tree. We establish the bitmap-based tree by traversing all physical address blocks and performing DRAM addressing of these addresses, in which each level of the tree represents a DRAM level and each node in the level contains a bitmap to index the next level. For example, the bitmap 10000111 in Ch_0 indexes $Ba_{0,5,6,7}$, indicating that these banks in the channel are associated with the physical blocks allocated by the DNN inference engine, in which errors are valid. As mentioned in Sec.II-B, MCU's adjacency manifests in three dimensions: rows, columns, and DQs. For a fast search, we place the *static* bank and channel levels closer to the root and the *dynamic* row, column, and DQ levels closer to the leaves. Thus, the subsequent adjacency checks of low-level nodes (i.e., row and column) will not impact the initial fast selection of high-level nodes (i.e., channel and bank) in a *depth-first traversal*. The search for a valid error mainly consists of two steps as illustrated in Fig. 3.

First, randomly select a high-level node based on its parent's bitmap. For example, select Ba_0 based on the bitmap in Ch_0. Second, select a low-level node based on the error type. 1) For SEU, the selection of a row and column is the same as the high-level node. 2) For column- and row-adjacent MCUs, the selection of columns and rows is based on the adjacency indices (i.e., 11) in their parent's bitmap. If no adjacency is found in the bitmap, the adjacency check will be continuously performed on the parent's sibling nodes adhering to the depth-first principle. Once the adjacent columns are determined, the rows at the next level can be selected based on the bitwise *AND* operation of these columns' bitmaps. This two-step search can efficiently identify a valid SEU/MCU error in DRAM. However, if only one error is searched per traversal, injecting multiple errors remains costly. Thus, we calculate the memory size of the sub-tree for each high-level node and assign the number of errors layer by layer, starting from the root. For example, Ch_0 has 10M address space and Ch_1 has 20M, approximately 1/3 of the SEUs/MCUs falling in Ch_0 and 2/3 in Ch_1. Therefore, all errors can be searched in a single depth-first traversal.

After error searching in physical memory space, we obtain a series of valid physical addresses that store data relevant to DNN inference, which should be further converted into valid virtual addresses using the previously mentioned one-to-one mapping chain between virtual and physical blocks (❹ in Fig. 3). Additionally, We optimize the mapping by combining several consecutive physical blocks into a large one, thereby further reducing the search complexity.

Complexity analysis. By decomposing the vast linear physical memory space into a nonlinear, bitmap-based tree, the complexity of error searching is significantly reduced. The reasons are two-fold. First, building a tree based on valid physical addresses and searching within it can eliminate the time overhead caused by rechecking invalid errors. Second, spatially adjacent MCUs in DRAM may span distant addresses due to data parallelism and row addressing by high bits of the physical address, making them tricky to map with a simple linear structure. Specifically, the complexity of error searching consists of two parts: error searching in the bitmap-based tree, and error converting from physical address space to virtual address space. The first one reduces from $O(mn)$ to $O(n)$, where n represents the number of valid physical addresses and M denotes the number of attempts on invalid addresses (e.g., for $k = 100$, m typically exceeds 30,000). The second one reduces from $O(n)$ to $O(s)$, where s represents the number of physical blocks (typically thousands in practice) with $O(1)$ addressing within each block using offsets.

D. Extension for usage

REMU requires extensive characterization of the target DRAM device and error model, which is achieved by user-friendly configurations. Following [32], we use external files of the target DRAM device (e.g., *DRAM standard file* and *DRAM addressing file*) as inputs to set up the DRAM simulator. For the default error model evaluated in this paper, we statistically configure SEUs and MCUs based on the extensive in-orbit and ground radiation tests [11]. SEUs are spatially independent and thus can be modeled as random bit-flips. MCUs are spatial adjacent along three dimensions (i.e., row, column, and DQ) with varying multiplicity (e.g., typically ranging from 2-bit to 8-bit), proportions, and patterns. For instance, the probability of flipping two DRAM cells along one wordline is higher (80%) than along one bitline (20%) [11]. The default error model is a combination of SEUs and MCUs with more details in Sec.IV-A. Finally, we pursue REMU in ease of use via source code instrumentation linking with $libREMU_mem.so$ library to automate memory addressing and inject SEUs/MCUs in the runtime DNN inference.

IV Experiments and results

A. Setup

DNN models. We describe the DNN baselines used for evaluating the effectiveness of REMU in uncovering radiation-induced sensitivities. Table I lists 10 commonly-used DNN models we evaluate. We target the classical Earth observation dataset, RESISC45 [33], and 7 DNN models for image classification tasks in orbit. We perform an extensive evaluation of diverse DNN backbones, covering traditional CNN-based (e.g., ResNet-50 [34], DenseNet-161 [35]), recent transformer-based structures (e.g., ViT [22]), also the lightweight models widely used on resource-constrained edge (e.g., EfficientNet [21] and MobileNet [36]), and later CNN/MLP variants (e.g., ConvNext [37] and MLPMixer [38]). Furthermore, we leverage a large-scale dataset, DOTA [39], and 3 variants (i.e., small, medium, large) of a practical DNN detector, YOLOv5 [40], for object detection tasks in orbit.

DNN engines. All evaluated DNN models are converted into TensorRT engines [20] and inferred on real satellite payloads, NVIDIA Jetson Xavier NX [28] with 16GB 128-bit LPDDR4 [29], using TensorRT runtime [20]. A prerequisite step of inference is to convert the trained DNN model from a DL framework (e.g., PyTorch) into optimized data flow graphs and build inference engines (in the main memory[3]). The TensorRT runtime then creates an execution context from the engine with input and output buffers in preparation for inference. Thus, the engines are crucial for the reliability of DNN inference. Table I shows the size of each engine allocated in memory in preparation for processing one input, which is a good indicator of the memory intensity of each DNN inference. Additionally, the accuracy of all DNN models' inferences without any error is listed in Table I. We further employ low-precision INT8 types to represent quantized floating point values (both parameters and activations) and utilize posting-training quantization to create quantized DNN engines. This can achieve faster execution speed ($\approx 4\times$) and lower memory usage ($\approx 1/4$) than full-precision, although incurring a slight accuracy degradation, which is not the priority of our study.

REMU configurations. We account for the bit error rate (BER) and the error model when injecting errors in runtime DNN inference via REMU. We set BER at 10^{-7} and 10^{-6} based on the statistics of existing real-world radiation tests [11], and the number of bit errors depends on the memory intensity of each inference, i.e., the size of each engine in this paper. The radiation-induced error model is a configurable input of REMU. In Sec.IV-B, we use the default one (Sec.III-D), to evaluate the error tolerance, which is a mixture of SEUs and MCUs. Fig. 5 shows the detailed proportion of SEUs/MCUs and multiplicity of MCUs for each BER and DNN model. Typically, the proportion of SEUs is higher than 90% and the proportion of $\geq$3-bit MCUs is significantly less than SEUs and 2-bit MCUs. In Sec.IV-C, we apply six additional error models to illustrate the impacts of MCUs. To better understand REMU's memory-aware error injections, we show a case in Fig. 4. We locate the emulated SEUs and MCUs in both the virtual and physical address space. Specifically, we visualize the lifecycle of two MCU examples during REMU's emulation from the DRAM cells to the inference process, which benefits the precise location of MCUs. Note that we use the default DRAM addressing without interleaved mode [41], thus the bits in an MCU will be kept in one contiguous physical address block. Other address mappings can be customized in REMU like [32].

TABLE I: DNN models used in our evaluations. The listed engines are optimized by TensorRT [20] for both FP32 full-precision and INT8 low-precision quantization.

Dataset	Model	Engine size		Baseline accuracy	
		FP32	INT8	FP32	INT8
RESISC45	ResNet-50	115MB	24MB	96.7%	96.6%
	DenseNet-161	144MB	32MB	96.0%	95.0%
	EfficientNet	18MB	6.9MB	91.8%	87.2%
	MobileNet	9.6MB	3.6MB	93.0%	92.2%
	ViT	328MB	107MB	97.1%	95.6%
	MLPMixer	227MB	63MB	95.7%	93.8%
	ConvNext	108MB	30MB	96.8%	93.1%
DOTA	YOLOv5s*	48MB	11MB	81.2%	76.2%
	YOLOv5m*	129MB	28MB	80.4%	75.8%
	YOLOv5l*	294MB	54MB	76.6%	77.3%

*Object detection uses mean average precision (mAP) instead of accuracy.

B. Error tolerance of DNN baselines

To better understand the error tolerance of each DNN model and explore the effectiveness of REMU in identifying sensitivities during each DNN inference, we conduct an extensive evaluation under millions of REMU's error injection. This shows us how differences in BERs, quantization, DNN models, error locations, and error models (Sec.IV-C) can potentially affect the reliability of DNN inference. To our knowledge, this is the first demonstration of DNN inference's error tolerance on runtime memory under near-to-real radiation-induced error emulation, which benefits from REMU's *practical error models* and *precise memory addressing*.

Avoid crashes. Fig. 7 illustrates the impact of SEU and MCU errors on the accuracy of DNN inference. We use REMU to execute 1000 error-injected inferences per BER per DNN model and show the average accuracy. However, we observe a high probability of crashes (e.g., aborts, CUDA failures) during runtime error injections when considering the entire space of the inference engine, which is not our concern. Our concern is a more stealthy error behavior with seamlessly completed inference but wrong results, making errors more difficult to discover. By dumping the engine, we find that errors at the end of the engine[4] are more likely to activate a crash. Therefore, we additionally omit these areas from the engine and evaluate the error tolerance of DNN inference when errors are located in the remaining space (the majority), denoted as *parameter space*

[3]In NVIDIA Jetson Xavier NX, the main memory and GPU memory are physically unified. So no typical data copy/transfer between them is needed.

[4]Engine's serialized format is close-sourced. By dumping the engine, We infer that its end likely contains network definition, engine metadata, executable code of CUDA kernel, etc., associated with inference.

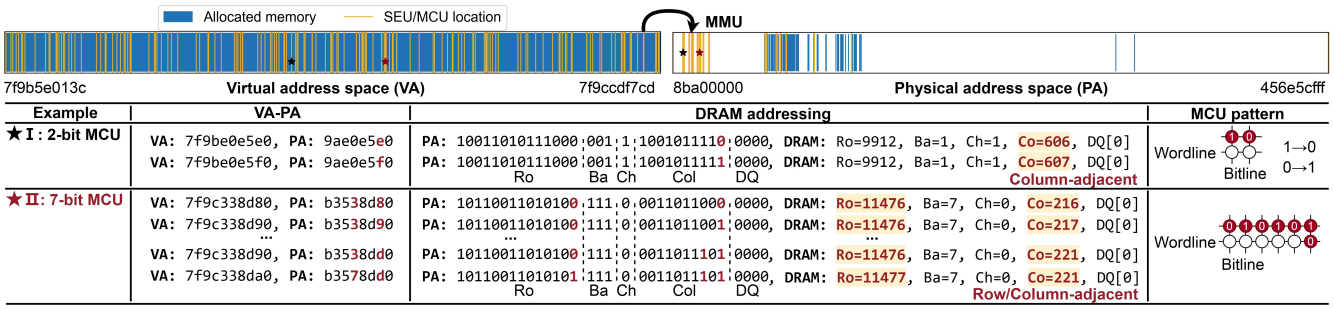


Fig. 4: A case of REMU's error injection. We use REMU to inject SEUs/MCUs in ResNet-50's runtime inference under 10^{-6} BER and the default error model. We show the continuous virtual address space of the inference engine allocated in the main memory and the corresponding discrete physical address space. The seemingly scattered SEUs/MCUs are more clustered in the physical address space. Besides, we give two MCU examples (e.g., 2-bit and 7-bit MCUs) to illustrate the REMU's precise memory addressing from the upper-level process space to the low-level DRAM cells in both row- and column-adjacency.

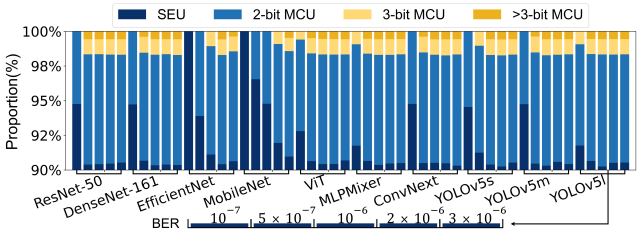


Fig. 5: The proportion of SEUs and the MCUs with different multiplicity in the default error model used for each DNN model and BER. The y-axis is the proportion and the x-axis corresponds to the DNN models (5 BER settings for each).

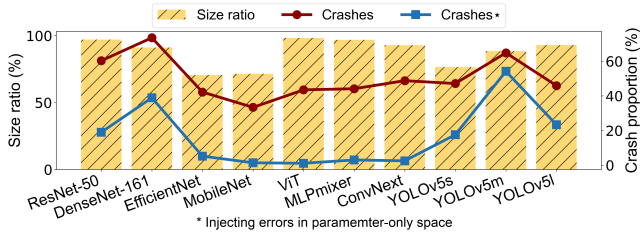


Fig. 6: The left y-axis represents the ratio of the parameter space size to the overall engine size. The right y-axis is the proportion of crashes observed in 1000 inferences. The x-axis corresponds to the DNN models.

(⋆), which results in fewer crashes. Fig. 6 lists the size and crash occurrence of the parameter space compared to the overall engine for each DNN model. Even with this additional evaluation, we underscore that injecting the overall engine is more realistic, as radiation can affect any memory space. When evaluating the overall engine, we skip the crashes and only record the completed inferences.

Quantization. Fig. 7a shows that precision affects the error tolerance of DNN inference, however, the results are not as expected when injecting errors solely in parameter space (⋆), indicating that the engine additionally holds critical data related to inference beyond the parameter space, which is overlooked in previous model-level error injection tools [17]–[19]. FP32 and FP16 precision exhibit a weaker error tolerance than INT8 precision due to the larger data range and fragile exponent bit. In the following, we focus on INT8 precision as a more challenging case for error attribution.

Accuracy collapse. Fig. 7 shows that the average accuracy declines across almost all DNN models as the BER increases. We observe a significant accuracy drop when injecting errors in the overall engine. For example, the average accuracy drops 18% in DenseNet-161 under 3×10^{-6} BER. If the BER continues increasing, the average accuracy will drop dramatically. Besides, Table II lists the minimum accuracy that occurred during tests, which means DNN inference may experience severe accuracy collapse under radiation-induced errors. Almost all DNN models may be degraded as random guesses (i.e., below 5%) even under a very low BER.

Takeaways. We demonstrate REMU 's effectiveness in robust model selection and sensitivity discovery, providing insights for radiation testing and hardening before in-orbit deployment.

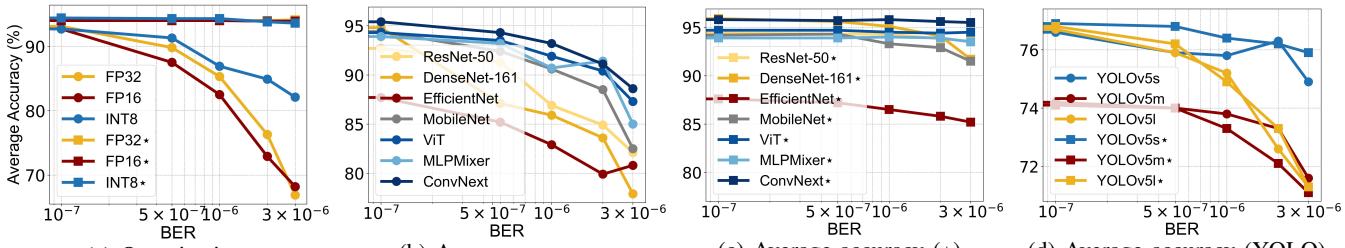
First, REMU can support all DNN models and select a better one balancing efficiency, accuracy, and tolerance. In Fig. 7b, REMU reveals that different DNN models exhibit varying sensitivity, e.g., DenseNet-161 (-18%) is more sensitive than ViT (-6%). We observe that the smaller the model, the more sensitive it is. ViT and MLPMixer are relatively more robust but less efficient than the lightweight EfficientNet and MobileNet. Conversely in YOLOv5, the small variant is less sensitive but may suffer a more severe accuracy collapse (i.e., 5.4%) than the medium/large. Therefore, when in-orbit edges lean towards smaller models due to resource constraints, we advocate greater efforts on the error tolerance of these models.

Second, REMU can effectively expose the undiscovered sensitivity in the engine. The accuracy collapse in Table II can be located in the engine for further protection. In the parameter space, different models have different locations. For instance, the collapse (27%) occurs in the No.4 layer of EfficientNet but in the No.2 layer of ResNet-50. Beyond the parameter space, the most severe collapse below 5% consistently occurs between two layers in the network definition at the end of the engine as illustrated in Table IIa. These special errors are not intuitively in parameters but auxiliary runtime logic related to the computation of inference. The collapse of 4% highlighted in blue results from the final output buffer not being processed and remaining initialized 0 due to errors. For YOLOv5, such collapse is rare because the post-processing in object detection will further filter out the abnormal output.

C. One MCU does not equal to multiple SEUs

To further investigate the superiority of REMU's memory awareness over previous random error injection [17]–[19], we evaluate the in-depth impact of MCUs that rely on precise addressing of adjacency. If MCUs are not considered (i.e., the proportion is 0), REMU will effectively degrade to random error injection. This is the first to exploit the MCU's impact on DNN inference.

The proportion of MCUs considered in the above default error model is 4%, among which 2% 2-bit MCUs, and 2% 3-bit MCUs. In this evaluation, we set six additional error models with different MCU multiplicities and proportions under the same BER: no MCUs, 50% 2-bit MCUs, 33% 3-bit MCUs, 25% 4-bit MCUs, 20% 5-bit MCUs, and 14% 7-bit MCUs. Fig. 8 shows that the increase in MCU proportion and multiplicity intensifies the reduction of average accuracy. These results are due to aggregated MCUs hitting nearly continuous sensitive bits of DNN inference, especially when the MCU



(a) Quantization. (b) Average accuracy. (c) Average accuracy (⋆). (d) Average accuracy (YOLO).

Fig. 7: Average accuracy (y-axis) under REMU's error injections. The x-axis represents BERs. (a) The average accuracy of different quantization levels (e.g., ResNet-50). (b)–(c) The average accuracy of seven classification models when injecting errors in the overall engine and parameter space (⋆). (d) The average accuracy of three variants of the detection model.

TABLE II: Accuracy collapse under REMU's error injections. (a) A case of collapse (highlighted in blue) in ResNet-50's inference engine. (b)–(c) Accuracy collapse of seven classification models when injecting errors in the overall engine and parameter space (⋆). (d) Accuracy collapse of three variants of the detection model.

(a) Collapse location.

```
7f82a37b80|.....(Unnamed La|
7f82a37b90|yer* 107) [Scale|
7f82a37ba0|]_output;.......|

...  ★7f82a37c14: fe➔7e
       Accuracy: 4%

7f82a37c70|..(Unnamed Layer|
7f82a37c80|* 145) [Activati|
7f82a37c90|on]_output;.....|
```

(b) Accuracy collapse.

Model	BER 10^{-7}	BER 10^{-6}
ResNet-50	1%	1%
DenseNet-161	2%	4%
EfficientNet	4%	1%
MobileNet	1%	1%
ViT	4%	4%
MLPmixer	4%	4%
ConvNext	4%	4%

(c) Accuracy collapse (⋆).

Model	BER 10^{-7}	BER 10^{-6}
ResNet-50⋆	79%	27%
DenseNet-161⋆	62%	9%
EfficientNet⋆	77%	27%
MobileNet⋆	60%	10%
ViT⋆	93%	41%
MLPmixer⋆	29%	10%
ConvNext⋆	64%	66%

(d) Accuracy collapse (YOLO).

Model	BER 10^{-7}	BER 10^{-6}
YOLOv5s	72.1%	43.9%
YOLOv5m	64.8%	50.4%
YOLOv5l	57.0%	57.2%
YOLOv5s⋆	10.0%	5.4%
YOLOv5m⋆	63.5%	53.7%
YOLOv5l⋆	51.6%	10.2%

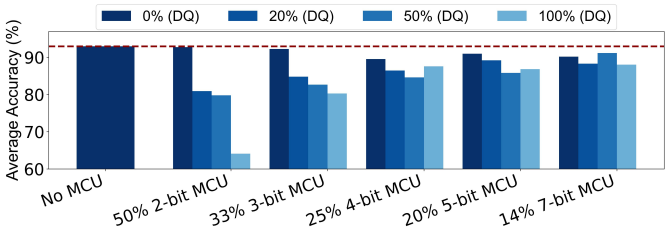


Fig. 8: MCUs' impact on ResNet-50's inference: x-axis is the 6 novel error models with different MCU multiplicities and proportions. For each error model, we set 4 probabilities of MCUs that occur along DQ. The y-axis is the 1000 inferences' average accuracy (10^{-7} BER).

TABLE III: Example of a byte in ResNet-50 inference engine: the accuracy under an SEU error at each bit and the accuracy under a 5-bit MCU error along the DQ-dimension.

Byte (01000001): Accuracy								
0	1	2	3	4	5	6	7	5-bit MCU
94%	42%	42%	42%	42%	42%	94%	94%	**9%**

diffuses along DQ, in addition to the DRAM rows and columns. We find that bits within a byte (i.e., along DQ) always exhibit identical high sensitivity. For example in Table III, almost every bit in some byte can lead to an accuracy drop, and together resulting in a more severe collapse 9%. To explore these sensitivities, we configure REMU to enable DQ-dimensional MCUs (i.e., set the probability of DQ-dimension greater than 0). Fig. 8 compares MCUs without DQ-dimension and with DQ-dimension of different probabilities for each novel error model. Please note that the proportion of MCU's DQ-dimension diffusion can be modeled via radiation tests. We propose preliminary insights and set the proportion to 0%, 20%, 50%, and 100% in this paper. DNN inference under increased errors along DQ generally shows a worse average accuracy. These results underscore that the introduction of memory addressing in REMU substantially supports the MCU's precise emulation, which is not equal to multiple SEUs and, therefore, can effectively expose more sensitive locations for further protection, such as aggregated sensitive bits.

D. Other results and discussions

Benefiting from the optimization via the bitmap-based tree, the efficiency of error injection is greatly improved. We test the completion time of DNN inference before/after error injection, also with/without the optimization. The result shows that error injection without optimization introduces a $67.1 - 99.8\times$ completion time increase while the overhead reduces to $3.1 - 3.5\times$ with the optimization. The above findings reveal that runtime DNN inference is highly sensitive to spatially correlated MCU errors and runtime logic. We hope our lessons can foster more academic and industrial efforts on reliable orbital DNN inference, such as allocating critical bits into different memory banks and leveraging detection on output buffers. Our future work will evaluate the sensitivity of DNN inference beyond DRAM and the engine, focusing on instructions and intermediate results.

V Conclusion

This paper designs a memory-aware emulator, REMU, to emulate radiation-induced bit-flip errors, i.e., SEUs and MCUs, and investigate their impact on in-orbit DNN inference. We demonstrate that existing emulation tools either lack easy usage or fail to precisely trace spatially correlated bit errors in memory across hierarchical systems. REMU achieves both correctness and efficiency by precise modeling of the memory system and leveraging the bitmap-based tree for dual addressing to reduce the complexity of large-scale error searching. Based on REMU, extensive experiments are conducted to understand the error behavior of runtime DNN inference, which can benefit the selection of robust model, and sensitivity discovery of the DNN before deployed in orbit.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB3105202), National Natural Science Foundation of China (62272266, 62132009, 62202261), and Center of High performance computing, Tsinghua University.

References

[1] B. Tao, O. Chabra, I. Janveja, I. Gupta, and D. Vasisht, "Known knowns and unknowns: Near-realtime earth observation via query bifurcation in serval," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 2024, pp. 809–824.

[2] R. Xing, M. Xu, A. Zhou, Q. Li, Y. Zhang, F. Qian, and S. Wang, "Deciphering the Enigma of Satellite Computing with COTS Devices: Measurement and Analysis," in *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2024.

[3] B. Denby, K. Chintalapudi, R. Chandra, B. Lucia, and S. Noghabi, "Kodan: Addressing the computational bottleneck in space," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023.

[4] B. Denby and B. Lucia, "Orbital edge computing: Machine inference in space," *IEEE Computer Architecture Letters*, pp. 59–62, 2019.

[5] D. Saaristo, "Shoot from the stars: Startup provides early detection of wildfires from space," https://blogs.nvidia.com/blog/ororatech-wildfires-from-space/.

[6] G. Choumos, K. Tsaprailis, V. Lappas, and C. Kontoes, "Artificial intelligence for a safe space: Data and model development trends in orbit prediction and collision avoidance," in *AIAA SCITECH 2024 Forum*, 2024, p. 2066.

[7] B. Denby and B. Lucia, "Orbital edge computing: Nanosatellite constellations as a new class of computer system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 939–954.

[8] M. Esposito, B. Dominguez, M. Pastena, N. Vercruyssen, S. Conticello, C. van Dijk, P. Manzillo, and R. Koeleman, "Highly integration of hyperspectral, thermal and artificial intelligence for the ESA PHISAT-1 mission," in *Proceedings of the International Astronautical Congress IAC, Washington, DC, USA*, 2019, pp. 21–25.

[9] A. Samaras, F. Bezerra, E. Lorfevre, and R. Ecoffet, "CARMEN-2: In flight observation of non destructive single event phenomena on memories," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 2011, pp. 839–848.

[10] J. P. Walters, K. M. Zick, and M. French, "A practical characterization of a NASA SpaceCube application through fault emulation and laser testing," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–8.

[11] J. C. Fabero, H. Mecha, F. J. Franco, J. A. Clemente, G. Korkian, S. Rey, B. Cheymol, M. Baylac, G. Hubert, and R. Velazco, "Single event upsets under 14-mev neutrons in a 28-nm SRAM-based FPGA in static mode," *IEEE Transactions on Nuclear Science*, pp. 1461–1469, 2020.

[12] F. Bezerra, R. Ecoffet, E. Lorfèvre, A. Samaras, and C. Deneau, "CARMEN2/MEX: An in-flight laboratory for the observation of radiation effects on electronic devices," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 2011, pp. 607–614.

[13] H. Wang, S. Myint, V. Verma, Y. Winetraub, J. Yang, and A. Cidon, "Mars attacks! software protection against space radiation," in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023.

[14] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "Llfi: An intermediate code-level fault injection tool for hardware faults," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 11–16.

[15] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 249–258.

[16] O. Chatzopoulos, G. Papadimitriou, V. Karakostas, and D. Gizopoulos, "Gem5-marvel: Microarchitecture-level resilience analysis of heterogeneous soc architectures," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 543–559.

[17] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 426–435.

[18] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 25–31.

[19] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[20] H. Vanholder, "Efficient inference with tensorrt," in *GPU Technology Conference*, vol. 1, no. 2, 2016, pp. 1–24.

[21] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[22] D. Alexey, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv: 2010.11929*, 2020.

[23] H. Su, T. Lu, C. Feng, and L. Chen, "Triple module redundancy reliability framework design based on heterogeneous multi-core processor," *Procedia Computer Science*, vol. 183, pp. 504–511, 2021.

[24] P. Caron, C. Inguimbert, L. Artola, R. Ecoffet, and F. Bezerra, "Physical mechanisms of proton-induced single-event upset in integrated memory devices," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1404–1409, 2019.

[25] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based sec-ded-daec code," in *25th IEEE VLSI Test Symposium (VTS'07)*. IEEE, 2007, pp. 349–354.

[26] D. Ma, F. Lin, A. Desmaison, J. Coburn, D. Moore, S. Sankar, and X. Jiao, "Dr. dna: Combating silent data corruptions in deep learning using distribution of neuron activations," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 239–252.

[27] N. Bleier, M. H. Mubarik, G. R. Swenson, and R. Kumar, "Space microdatacenters," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 900–915.

[28] NVIDIA, "Jetson Xavier NX Developer Kit," 2024, online; accessed 14-March-2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit

[29] J. S. S. T. Association *et al.*, "JEDEC standard: Low power double data rate 4 (LPDDR4)," *JEDEC Standard JESD209-4*, 2014.

[30] M. Abadi, "Tensorflow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, 2016, pp. 1–1.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[32] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[33] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *Proceedings of the IEEE*, vol. 105, no. 10, pp. 1865–1883, 2017.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[35] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[36] A. G. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[37] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[38] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit *et al.*, "Mlp-mixer: An all-mlp architecture for vision," *Advances in neural information processing systems*, vol. 34, pp. 24 261–24 272, 2021.

[39] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Dota: A large-scale dataset for object detection in aerial images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3974–3983.

[40] G. Jocher, "Yolov5 by ultralytics," 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[41] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks," in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.

Self-Attention to Operator Learning-based 3D-IC Thermal Simulation

Zhen Huang[1,2], Hong Wang[1], Wenkai Yang[3,5], Muxi Tang[4], Depeng Xie[5], Ting-Jung Lin[2], Yu Zhang[1], Wei W. Xing[6,†], Lei He[2,7,†]
[1] School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2] Ningbo Institute of Digital Twin, Eastern Institute of Technology, Ningbo, China
[3] ShanghaiTech University, Shanghai, China [4] Tsinghua University, Beijing, China
[5] BTD Technology, Ningbo, China [6] University of Sheffield, Sheffield, UK
[7] University of California, Los Angeles, USA
[†]E-mail: {w.xing@sheffield.ac.uk, lhe@ucla.edu}

***Abstract*—Thermal management in 3D ICs is increasingly challenging due to higher power densities. Traditional PDE-Solving based methods, while accurate, are too slow for iterative design. Machine learning approaches like FNO provide faster alternatives but suffer from high-frequency information loss and high-fidelity data dependency. We introduce Self-Attention U-Net Fourier Neural Operator (SAU-FNO), a novel framework combining self-attention and U-Net with FNO to capture long-range dependencies and model local high-frequency features effectively. Transfer learning is employed to fine-tune low-fidelity data, minimizing the need for extensive high-fidelity datasets and speeding up training. Experiments demonstrate that SAU-FNO achieves state-of-the-art thermal prediction accuracy and provides an 842× speedup over traditional FEM methods, making it an efficient tool for advanced 3D IC thermal simulations.**

***Index Terms*—Thermal modeling, Self-Attention, Fourier Neural Operator, Transfer Learning.**

I. Introduction

Thermal simulation in Electronic Design Automation (EDA) is essential as integrated circuit technology advances. Compared to traditional 2D and 2.5D designs, 3D integration has higher power density, making it more susceptible to overheating and hot spots, which can degrade performance and shorten chip lifespan [1]–[4]. Efficient thermal management and rapid simulation tools are therefore vital for layout optimization and reliability [5].

3D chip thermal simulation involves solving the 3D heat conduction PDE. Traditional methods like the Finite Element Method (FEM) and Finite Difference Method (FDM), as used in tools like Ansys Icepak, Celcius, COMSOL, and MTA, are accurate but often too slow due to the complexity of mesh partitioning and PDE solving, limiting their use in iterative design optimization [6]–[8].

With the rise of artificial intelligence, ML-based approaches have gained attention in thermal simulation [9]–[14]. For example, [15] introduced a CNN model for temperature field prediction, but purely data-driven methods lack physical interpretability and need extensive training data.

PINN [16], which parameterizes PDE solutions directly through neural networks, has been used for various heat transfer problems [17], [18]. Its main advantage is embedding physical laws into the learning process for consistent results. However, PINN models require retraining for different system instances due to their fixed geometry, limiting scalability in 3D IC thermal simulations. DeepOHeat [19] improved this by combining PINN with operator learning to map power distribution and heat transfer, though at the cost of long training times. Despite leveraging DeepONet's [20] grid-invariance, training duration remains a significant challenge.

FNO uses Fast Fourier Transform (FFT) [21] to approximate PDEs, making it more consistent with physical laws than CNNs. It performs well in the Darcy flow and the Navier-Stokes equation. The resolution-invariant capability ensures FNO's adaptability to different chip sizes, a key advantage in thermal simulations [22], [23]. Recent work, such as MLA-FNO [24], improves precision and speed by combining active learning and FNO. However, FNO methods are limited by two significant challenges. First, its tendency to neglect high-frequency components results in notable errors. Secondly, FNO still demands considerable high-fidelity simulation data for effective model training.

Inspired by U-Net's ability to preserve multi-scale features through skip connections [25], [26] and self-attention's [27] capacity to capture long-range dependencies, we integrated these mechanisms with the FNO for 3D IC thermal modeling. This combination significantly improves the accuracy of thermal predictions, particularly in capturing local temperature variations and high-frequency details, leading to more precise thermal management and design. Additionally, we leverage transfer learning [28] by fine-tuning the model trained with low-fidelity data, further improving performance and reducing the data requirements. This approach enhances the accuracy of thermal predictions, particularly in capturing local temperature variations and high-frequency details, and addresses the inherent challenge of data scarcity in machine learning applications. Our key contributions are as follows:

- We propose a novel operator learning architecture, **S**elf-**A**ttention **U**-Net **F**ourier **N**eural **O**perator (**SAU-FNO**), that integrates attention mechanisms and U-Net with FNO to enhance 3D IC thermal modeling, particularly in capturing high-frequency information, making it better suited for 3D IC thermal simulations.

[†] Corresponding authors. This work was partially supported by "Science and Technology Innovation in Yongjiang 2035" (2024Z283) and by research support from BTD Inc.

- We incorporate transfer learning to fine-tune the model with low-fidelity data, which avoids reliance on high-fidelity data while achieving competitive accuracy.
- Extensive experiments demonstrate that our method achieves state-of-the-art (SOTA) results, with metrics Mean Squared Error (MSE) reduced by over 50% compared to other approaches.
- SAU-FNO achieves significant computational acceleration, delivering over 842× speedup compared to mainstream commercial software such as COMSOL and MTA, while maintaining comparable accuracy.

The rest of the paper is structured as follows. Section II provides the background and preliminaries of thermal modeling and related technology. Section III describes the proposed SAU-FNO in detail. Section IV presents the experimental results, and Section V summarizes the paper.

II. Background

A. Thermal modeling of 3D IC

The core of 3D IC thermal analysis is simulating heat conduction. Thermal analysis tools compute temperature distribution across the chip based on power routing and key physical parameters like chip layout and stack geometry, enabling accurate heat distribution evaluation and thermal management optimization. The main challenge lies in solving the global heat conduction PDEs:

$$\sum_{i=1}^{3} \frac{\partial}{\partial y_i}\left(k\frac{\partial T}{\partial y_i}\right) + Q_g = \rho c_p \frac{\partial T}{\partial t}, \tag{1}$$

Where C_p, ρ, and k are the capacity, mass density, and thermal conductivity of the material, respectively. Q_g and T stand for the rate of internally generated energy per unit volume at any spatial-temporal location and the temperature.

For simplicity, we consider a steady temperature field of isotropic materials, simplifying the equation as follows:

$$C_v \frac{dT}{dt} = k\nabla^2 T + Q_g, \tag{2}$$

Where C_v represents the volume heat capacity of a substance, $\frac{dT}{dt}$ represents the rate of change of temperature with time, k represents the thermal conductivity, $\nabla^2 T$ represents the Laplacian operator, and Q_g the rate of production of heat inside the control volume represents the volume heat generation rate.

We consider the steady-state simulation problem and get the following simplified formula:

$$k \cdot \nabla^2 T + Q_g = 0 \tag{3}$$

the temperature distribution $t(x)$ of 3D IC is governed by the boundary condition, where Robin condition and Neumann condition are adopted:

$$\begin{aligned} \kappa(x)\nabla t(x) \cdot n &= g(x) \\ \kappa(x)\nabla t(x) \cdot n &= \eta(t_a - t(x)) \end{aligned} \tag{4}$$

where $t(x)$ represents the temperature at position x, η is the thermal transmissivity. $f(x)$ denotes power trace distribution.

B. Fourier Neural Operators

Neural networks excel at learning mappings in finite-dimensional spaces but struggle with parameter or boundary condition variations, limiting generalizability. Neural operators address this by learning mappings between function spaces. The Fourier Neural Operator (FNO) efficiently maps initial conditions to PDE solutions, making it valuable for complex physical problems.

The operator mapping is mathematically described as:

$$\mathcal{G} : \mathcal{U} \to \mathcal{V}, \tag{5}$$

where $\mathcal{G}$ maps inputs from the function space $\mathcal{U}$ to outputs in $\mathcal{V}$. This mapping generalizes across various input scenarios within these spaces, not confined to single instances.

FNOs leverage FFT to map PDE solutions into the frequency domain, enhancing generalization and computational speed compared to traditional models like CNNs. Their resolution invariance and scalability make them ideal for fluid mechanics (e.g., Navier-Stokes equations) and efficient for large-scale 3D IC thermal simulations across varying chip sizes and grid resolutions.

The core functionality of FNOs involves transforming the input function $u_\ell(x)$ through a series of layers, each performing the operation:

$$u_{\ell+1}(x) = \sigma\left(W u_\ell(x) + \mathcal{F}^{-1}\left(\rho(\xi) \cdot \mathcal{F}\left(u_\ell(x)\right)\right) + b\right). \tag{6}$$

Here, $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier and inverse Fourier transforms, respectively. The term $\rho(\xi)$ represents a learnable convolution kernel in the frequency domain, W is a linear operator (weight matrix), b is a bias term, and σ is the Gaussian Error Linear Unit (GELU).

By using Fourier transforms, FNOs capture global and local patterns critical for modeling complex PDE behaviors, enabling generalization across resolutions and geometries for various physical problems. In summary, FNOs advance neural network architectures for PDE modeling with their operator-learning capability, resolution invariance, and computational efficiency, making them powerful for complex simulations, including thermal modeling.

C. Self-attention

Self-attention, first popularized in NLP [27], captures global information and long-distance dependencies more effectively than traditional networks. By dynamically adjusting data relationships through queries, keys, and values, it excels with complex, high-dimensional data. Unlike CNNs focusing on local fields, self-attention learns global interdependencies, enhancing recognition of broad patterns and fine details.

In applications such as 3D IC thermal modeling, self-attention not only tracks regional temperature correlations but also identifies local temperature spikes, adapting seamlessly to large-scale data. This capability enhances detail capture, particularly of high-frequency components, thus boosting both the accuracy and efficiency of simulations.

D. Transfer learning

Transfer learning transfers knowledge from pre-trained models to solve new tasks, particularly useful when data is scarce or costly [28]. It leverages features learned from a related task and fine-tunes them for the target task, reducing reliance on large amounts of high-fidelity data while improving model performance and convergence speed.

In 3D IC thermal simulation, obtaining high-fidelity data is expensive and time-consuming. We addressed this by initially training a model with low-fidelity data to learn basic heat transfer patterns and global features quickly. The model was then fine-tuned with a small set of high-fidelity data to capture detailed features, like local hot spots and junction temperatures. This strategy enhances complex temperature distribution prediction, reduces high-fidelity data requirements, and accelerates convergence, offering an efficient solution for large-scale simulations.

III. Proposed Method

Our method **SAU-FNO** consists of three stages: lifting the input to a higher-dimensional space, passing it through iterative U-Fourier layers, and projecting the result back into the original space. Each U-Fourier layer employs a combination of kernel integral transformations, U-Net and CNN operators, and linear operators, enhancing feature extraction.

Additionally, we introduce a self-attention mechanism to capture long-distance dependencies, which is crucial for tasks like 3D thermal simulations. This setup also supports transfer learning, allowing training on low-resolution data and fine-tuning with high-resolution data for improved generalization.

A. Fundamental U-FNO Design

As described in Section 2, FNO consists of multiple superimposed Fourier layers. We enhance the original Fourier layers by integrating a U-Net bypass in each U-FNO layer, as illustrated in Figure 1. The U-Net processes local convolutions, enhancing the representation of high-frequency information. The specific steps of our proposed method are as follows:

1) **Lifting:** Lift the input observation $a(x)$ to high dimensional space using a neural network transformation P.
2) **Iterative Layers:** Apply iterative Fourier layers followed by iterative U-Fourier layers:

$$v_{l_0}(x) \to \cdots \to v_{l_L}(x) \to v_{m_0}(x) \to \cdots \to v_{m_M}(x), \tag{7}$$

where $v_{l_k}(x)$ for $k = 0, 1, \ldots, L$ and $v_{m_k}(x)$ for $k = 0, 1, \ldots, M$ are sequences of functions taking values in $\mathbb{R}^c$, with channel dimension c.
3) **Projection:** Project $v_{m_M}(x)$ back to the original space using a fully connected neural network transformation Q, yielding $z(x) = Q(v_{m_M}(x))$.

Within each newly proposed U-Fourier layer, we define:

$$v_{m_{k+1}}(x) = \sigma\left(\mathcal{K}v_{m_k}(x) + \mathcal{U}v_{m_k}(x) + Wv_{m_k}(x)\right), \forall x \in D, \tag{8}$$

where $\mathcal{K}$ is the kernel integral transformation, $\mathcal{U}$ is a U-Net CNN operator, and W is a linear operator; all are learnable parameters. The activation function σ is GELU, which introduces strong non-linearity into each U-Fourier layer. Other settings are detailed in Section IV, Experiment Settings.

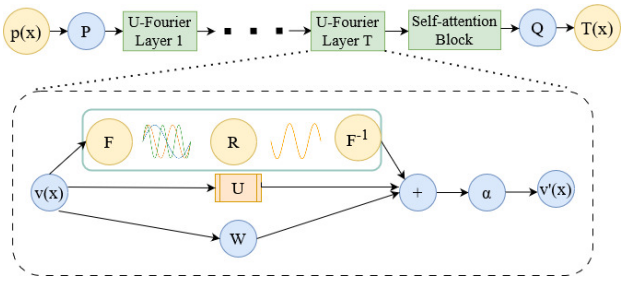


Fig. 1. Our SAU-FNO model architecture. $p(x)$ is the input, P and Q are fully connected neural networks, and $T(x)$ is the output. $\mathcal{F}$ denotes the Fourier transform, R is the parameterization in Fourier space, $\mathcal{F}^{-1}$ is the inverse Fourier transform, W is a linear bias term, and α is the activation function. U denotes a U-Net.

B. Attention-Enhanced U-FNO

Attention mechanisms, widely used in computer vision models like Vision Transformer (ViT), excel at capturing key spatial relationships. Similarly, thermal fields exhibit complex spatial temperature variations. By leveraging self-attention, models can capture long-range dependencies, enhancing the understanding of global heat conduction effects. Inspired by this, we integrate a self-attention module to improve feature extraction in 3D IC thermal simulation tasks.

The overall self-attention block architecture is shown in Figure 2. We perform a 1×1 convolution on the feature map $V_t(x)$ obtained from the U-FNO to compute the spatial attention map A_s and the channel attention map A_c sequentially. The computations are as follows:

$$\begin{aligned} A_c &= W_h V_t(x), \quad Q = W_q V_t(x), \\ K &= W_k V_t(x), \quad s_{ij} = Q_i^\top K_j, \\ A_s &= \text{softmax}(s_{ij}) = \frac{\exp(s_{ij})}{\sum_{k=1}^{N} \exp(s_{ik})}, \end{aligned} \tag{9}$$

where W_h, W_q, and W_k are learnable parameters representing the embeddings for the value, query, and key, respectively. The softmax function ensures that the attention weights are properly normalized. These parameters are trained jointly with the U-Fourier layer during training.

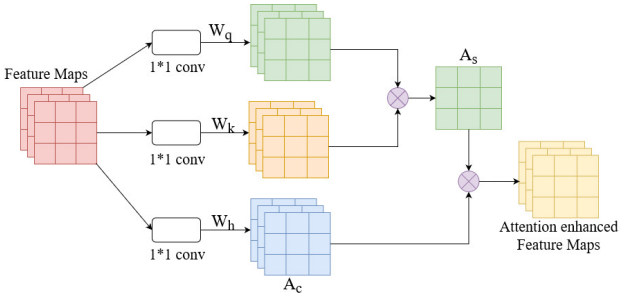


Fig. 2. The self-attention block architecture.

By using 1×1 convolutions in the attention block, we retain the mesh-invariant property of the original U-FNO, preserving its ability to generalize from coarse grid training to high-resolution grid inference. This approach also facilitates transfer learning using multi-fidelity data.

The final attention-enhanced feature map is obtained by combining the spatial and channel attention maps:

$$V_t'(x) = A_s \otimes A_c, \tag{10}$$

where $\otimes$ denotes element-wise multiplication.

As described in Section III-A, the architecture of our proposed U-FNO layer is:

$$v_0 \to v_1 \to v_2 \to \cdots \to v_t, \tag{11}$$

which directly affects the final thermal modeling performance. We refine the U-FNO layer by inputting the feature map $V_t(x)$ to obtain the attention-enhanced feature map $V_t'(x)$, as shown in Figure 2.

The experiments show that adding self-attention blocks after all U-FNO layers yields similar performance to adding them only after the last one or two layers. We attribute this to the multi-layer architecture of U-FNO, where features are gradually transmitted and accumulated. Moreover, the U-Net's skip connections facilitate the propagation of key information, even when attention is applied only in the last layer.

Therefore, to reduce computational complexity, we introduce the attention block only in the last layer, in the $V_t(x) \to V_t'(x)$ step. Refer to Section IV for other detailed settings.

C. Training with Transfer Learning

SAU-FNO effectively models 3D ICs and, with its mesh invariance, can train on low-resolution data and infer on high-resolution data, enabling better generalization than traditional networks. However, large grid resolution differences between training and testing may reduce performance, as low-resolution data often misses complex local and high-frequency details.

Training directly with high-resolution data is costly and impractical. Transfer learning addresses this by pre-training on low-resolution data and fine-tuning with a small amount of high-resolution data, boosting prediction accuracy and reducing reliance on extensive high-fidelity datasets, thus improving efficiency.

1) **Pre-training:** First, a large amount of low-resolution data $D_l = \{X_l, Y_l\}$ is used to train a low-fidelity model M_l. Here, Y_l is the prediction obtained by M_l to the input X_l. We use the L2 loss function to optimize the parameters of the model:

$$L_2 = \frac{1}{N} \sum_{i=1}^{N} \left(T_{\text{pred},i} - T_{\text{true},i}\right)^2, \tag{12}$$

where N is the number of samples, $T_{\text{true},i}$ represents the true value for sample i, and $T_{\text{pred},i}$ represents the model's prediction for sample i.

2) **Fine-tuning:** In the second stage, the pre-trained model M_l is fine-tuned using a relatively small amount of high-fidelity data $D_h = \{X_h, Y_h\}$. The training process is similar to the pre-training stage. The learning rate lr_2 during fine-tuning is about an order of magnitude smaller than that used in the pre-training stage. The rest of the settings are detailed in the experimental section.

IV. EXPERIMENTAL RESULTS

A. Experiment Settings

Experimental Setup: We adopted the Alpha 21264 EV6 microprocessor architecture [29] as the core structure and selected three different 3D ICs for our experiments. These ICs share the same face-to-back stacking structure, including device layers, Thermal Interface Material (TIM), Through-Silicon Vias (TSVs), packaging, and a heatsink. Specifically, Chip1 is a single-core processor consisting of two device layers: one layer contains the core, two L1 caches, and one L2 cache, while the other layer comprises three L2 caches. Chip2 is a three-layer quad-core processor, with the top layer closest to the heatsink consisting of four cores, and the remaining two layers identical, each containing two L2 caches. Chip3 is an octa-core two-layer microprocessor with an architecture similar to Chip1, where the upper layer is composed of eight cores and L1 caches, and the lower layer consists of four L2 caches. In all chips, the address and data bus between the L2 cache and the processor core are connected via TSVs. The 3D structure is illustrated in Figure 3. To simplify the figure of the floorplan, TSVs passing through each layer have been omitted. Additionally, Table I provides detailed structures of the 3D ICs.

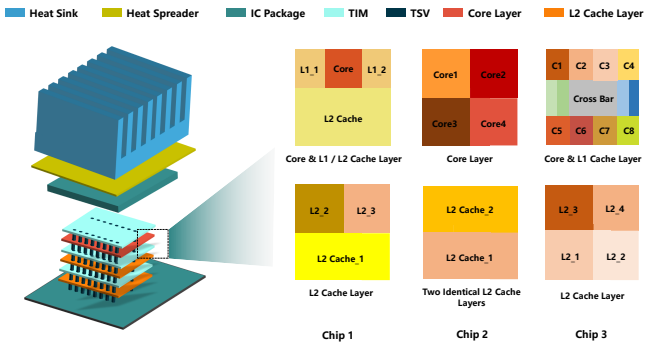


Fig. 3. The stacking structures and floorplan of three 3D chips examined in this work.

Data Generation: We randomly assigned power levels to different functional blocks while ensuring the total power remained within an appropriate range. By varying the power distribution, we aimed to achieve significant and reasonable temperature differences in the simulation results, reflecting real-world scenarios. The model output is a three-dimensional temperature distribution; due to the slow processing speed of commercial software such as Celsius, Icepak, and COMSOL, we adopted MTA [30] as the solver to derive the training data outputs. For each of the three types of chips, we generated

5,000 sets of distinct power distributions. Unless otherwise specified, the data was divided into training and testing sets at a ratio of 4:1.

Model Setting: We referenced the settings from previous work and used Optuna and W&B for hyperparameter tuning. For Chip1 and Chip2, the SAU-FNO was configured with a width of 2 and a model structure of [12, 12, 2]. For Chip3, the width was doubled to 64, resulting in a model structure of [24, 24, 2], to accommodate its larger data scale and grid size. The U-Net encoder has 4 layers of 3×3 convolution with ReLU activations, producing feature maps of [64, 128, 256, 512], while the decoder restores them to [512, 256, 128, 64] using bilinear interpolation and 3×3 convolutions. The self-attention module processes inputs of shape [Bs, 64, 64, 64] into Q, K, and V matrices with dimensions [Bs, 64, 64, d] (d=64). Further settings follow [31] and [32].

Training and Testing: We utilized a decaying learning rate with the Adam optimizer. The initial learning rate was set to 1e-4, and a weight decay of 1e-5 was applied to mitigate model overfitting. For fine-tuning, an initial learning rate of 1e-5 was used to ensure training stability. Each training process ran for over 200 epochs to guarantee convergence. All experiments were conducted on an RTX 3090 GPU with CUDA 11.3, using PyTorch version 1.10.1 and Python 3.7.

B. Compared to Latest ML methods

Recently, numerous studies have applied machine learning to 3D IC thermal modeling, but these methods have various limitations. Our approach, as a neural operator method, offers unique properties like mesh invariance not found in traditional neural networks. Direct comparisons of accuracy or time are not entirely fair, so we only benchmarked against recent neural operator methods, including FNO, U-FNO, and DeepOHeat.

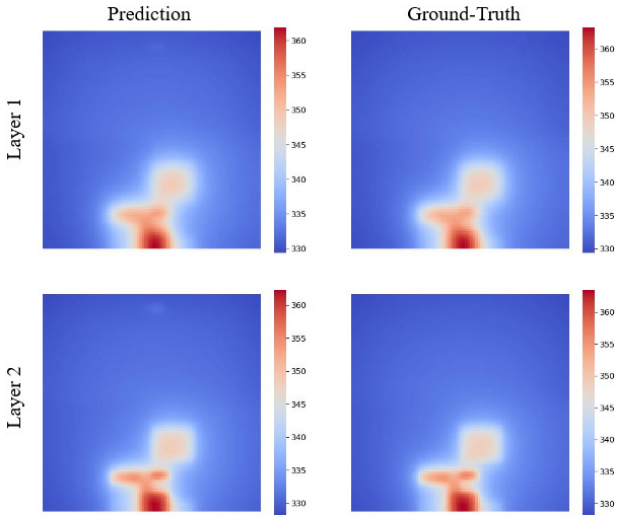


Fig. 4. Comparisons between our SAU-FNO predictions and Ground-Truth on Case 1 of Chip1.

We selected two representative cases with significant power distribution variations from Chip1 for visualization, as shown

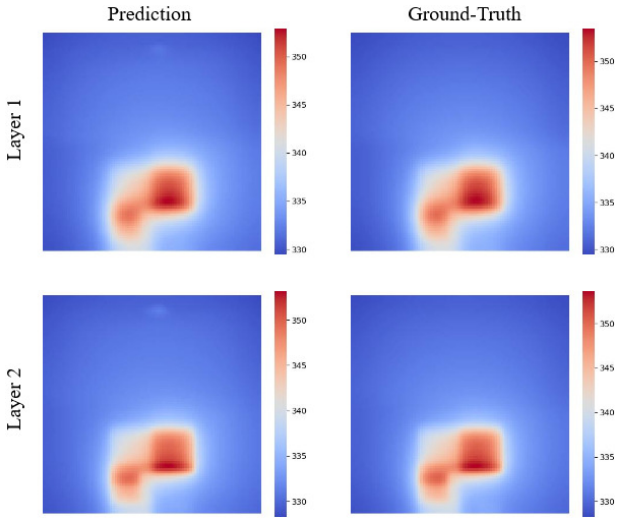


Fig. 5. Comparisons between our SAU-FNO predictions and Ground-Truth on Case 2 of Chip1.

in the heatmaps Figure 4 and Figure 5. The chips have two heating layers, labeled as layer 1 and layer 2. The visualizations show that our thermal simulation results closely match the ground truth, demonstrating strong performance in average temperature difference and excelling in junction temperature.

For quantitative evaluation, we used metrics such as RMSE, average temperature error (Mean), and maximum temperature error (Max), along with MAPE and PAPE. As shown in Table II, SAU-FNO significantly outperforms the commonly used DeepOHeat and FNO, with RMSE and junction temperature reductions of over 50% compared to FNO. While methods like U-Net lack resolution invariance and were not extensively compared for fairness, our results demonstrate clear advantages, including significant improvements over GAR, highlighting the superior performance and robustness of our approach. Additionally, the comparison between our method and U-FNO demonstrates the effectiveness of the self-attention module, while the improvements from FNO to U-FNO further highlight the contribution of the U-Net architecture. These comparisons effectively serve as an ablation study validating the importance of each component in our method.

We found that the structural differences among chips lead to variations in absolute accuracy. However, our method consistently achieves significant improvements and reaches SOTA performance across all chips. Therefore, we only report the average quantitative results for Chip2, with results for other chips provided in the supplementary material.

C. Validation of Transfer Learning

We implemented transfer learning by pre-training the model on extensive low-fidelity data and fine-tuning it with a small amount of high-fidelity data. This approach achieves nearly the same effectiveness as training from scratch with large-scale high-fidelity data, while significantly reducing the need for such data and accelerating training.

TABLE I
GEOMETRIC STRUCTURES AND THERMAL PARAMETERS OF 3D-ICS

	size (mm)			Thermal Attributes	
	Single-Core	Quad-Core	Octa-Core	Thermal Conductivity (W/m·K)	Volumetric Heat Capacity (J/m^3K)
L2 Cache Layers	16×16×0.15	12.4×12.76×0.15	10×10×0.1	100	1.75×10^6
Core layer	16×16×0.15	12.4×12.76×0.15	10×10×0.1	100	1.75×10^6
TIM	16×16×0.02	12.4×12.76×0.02	10×10×0.052	4	4.00×10^6
Heat Spreader	30×30×1	30×30×1	30×30×1	400	3.55×10^6
Heat Sink (Base)	60×60×6.9	60×60×6.9	60×60×6.9	400	3.55×10^6
Heat Sink (Fins×21)	1×60×50	1×60×50	1×60×50	400	3.55×10^6
TSV	diameter:0.01 ; pitch:0.01			100	1.75×10^6

TABLE II
COMPARISON OF OUR METHOD WITH OTHER ML METHODS, MAX STANDS FOR JUNCTION TEMPERATURE AND MEAN STANDS FOR AVERAGE TEMPERATURE.

Method	Resolution	RMSE	MAPE	PAPE	Max	Mean
DeepOHeat [19]	40∗40	0.457	0.093	0.811	2.936	0.297
	64∗64	0.473	0.097	0.914	4.355	0.312
FNO [21]	40∗40	0.438	0.086	0.730	2.774	0.329
	64∗64	0.409	0.077	1.073	4.081	0.292
U-FNO [31]	40∗40	0.221	0.049	0.195	0.741	0.185
	64∗64	0.278	0.065	0.275	1.046	0.253
GAR [33]	40∗40	0.576	0.127	0.893	4.639	0.153
	64∗64	0.598	0.145	0.917	5.132	0.170
Ours	40∗40	0.197	0.041	0.168	0.650	0.146
	64∗64	0.203	0.043	0.179	0.773	0.162

As shown in Table III, RMSE and junction temperature metrics showed minimal accuracy loss with fine-tuning compared to full high-fidelity training. The low-fidelity model used 4,000 cases, while fine-tuning the high-fidelity model required only 1,000 cases, achieving a 4:1 ratio of low-resolution to high-resolution data, which we determined to be optimal after multiple experiments. A high-fidelity model trained from scratch with 4,000 cases served as a benchmark. Given that high-resolution data collection is 4 to 6 times slower, total training time was reduced by about 2.5 times. Increasing fine-tuning data to 4,000 cases slightly improved accuracy. Transfer learning was also effective on other ML methods like FNO and U-FNO, proving its suitability for efficiency-critical 3D IC thermal simulations.

TABLE III
THE COMPARISON BEFORE AND AFTER TRANSFER LEARNING ON CHIP1, TRANSFER MEANS TRANSFER LEARNING.

Method	Transfer	RMSE	MAPE	PAPE	Max	Mean
FNO [21]	-	0.177	0.026	0.581	2.140	0.089
	✓	0.185	0.031	0.615	2.245	0.097
U-FNO [31]	-	0.130	0.018	0.298	1.115	0.061
	✓	0.141	0.023	0.309	1.224	0.073
Ours	-	0.090	0.014	0.232	0.871	0.049
	✓	0.097	0.015	0.241	0.897	0.058

D. Compared to Professional Thermal Simulation Tool

We compared the accuracy and speed of our model with mainstream 3D IC thermal simulation tools, including commercial software like COMSOL and open-source tools such as MTA [30] and Hotspot [34]. COMSOL, based on FEM, is suited for complex engineering, while MTA is an academic FEM tool, and Hotspot uses fast thermal network analysis based on FDM and empirical models. For simplicity, we only compared junction temperature and minimum temperature here, while other metrics such as MAPE are provided in the supplementary material. To achieve a certain degree of fairness in comparison, we conducted simulation experiments with them using the finest mesh and maximum resolution available.

Using the method in Section IV, we generated 20 power distributions for chip1-chip3 high-fidelity data and ran simulations to measure the maximum temperature and distribution, comparing average errors with our approach. Junction temperature was the primary comparison metric. Table IV shows that SAU-FNO predictions are close to baseline methods, with a maximum error within 0.25K, proving its practical usability. While COMSOL is slower, MTA and Hotspot performed faster. Our SAU-FNO method averaged 0.27s per prediction, compared to 227.31s for MTA and 98.47s for Hotspot, achieving speedups of 842x and 365x, respectively.

TABLE IV
MAXIMUM AND MINIMUM TEMPERATURE COMPARISONS AMONG COMSOL, MTA, HOTSPOT AND SAU-FNO ON FIVE STEADY-STATE SAMPLES.

		COMSOL	MTA	Hotspot	Ours	Error*
$chip_1$	Max(K)	381.257	381.173	391.051	381.159	-0.098
	Min(K)	328.032	328.107	334.970	328.139	+0.107
$chip_2$	Max(K)	380.065	380.134	392.015	380.195	+0.130
	Min(K)	325.259	325.450	336.290	325.473	+0.214
$chip_3$	Max(K)	422.207	422.079	432.017	422.091	-0.108
	Min(K)	347.496	347.330	348.019	347.353	-0.143

* Error refers to the difference between SAU-FNO and COMSOL.

V. CONCLUSION

We present SAU-FNO, an advanced thermal modeling framework that addresses key challenges in 3D IC thermal management. By integrating self-attention and U-Net architectures into the FNO, SAU-FNO effectively captures both global and local thermal dynamics. Our results demonstrate significant accuracy improvements and an 842× reduction in computation time compared to traditional methods. Transfer learning further boosts performance, reducing reliance on extensive high-fidelity datasets. Future work will expand SAU-FNO's capabilities to support a broader range of thermal analysis tasks, promoting more efficient design processes in advanced semiconductor technologies.

REFERENCES

[1] H. Delaram, A. Dastfan, and M. Norouzi, "Optimal thermal placement and loss estimation for power electronic modules," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 8, no. 2, pp. 236–243, 2018.

[2] H. Sultan, A. Chauhan, and S. R. Sarangi, "A survey of chip-level thermal simulators," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–35, 2019.

[3] K. Cao, J. Zhou, T. Wei, M. Chen, S. Hu, and K. Li, "A survey of optimization techniques for thermal-aware 3D processors," *Journal of Systems Architecture*, vol. 97, pp. 397–415, 2019.

[4] M. Neisser, "International roadmap for devices and systems lithography roadmap," *Journal of Micro/Nanopatterning, Materials, and Metrology*, vol. 20, no. 4, pp. 044 601–044 601, 2021.

[5] J.-M. Lin, W.-Y. Chang, H.-Y. Hsieh, Y.-T. Shyu, Y.-J. Chang, and J.-M. Lu, "Thermal-aware floorplanning and TSV-planning for mixed-type modules in a fixed-outline 3-d ic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 9, pp. 1652–1664, 2021.

[6] Y.-C. Chen, S. Ladenheim, H. Kalargaris, M. Mihajlović, and V. F. Pavlidis, "Computationally efficient standard-cell fem-based thermal analysis," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 490–495.

[7] P. Li, L. T. Pileggi, M. Asheghi, and R. Chandra, "Efficient full-chip thermal modeling and analysis," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 2004, pp. 319–326.

[8] H. Sultan, A. Chauhan, and S. R. Sarangi, "A survey of chip-level thermal simulators," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–35, 2019.

[9] L. Yin, A. Wang, W. Zhu, A. Guo, J. Liu, M. Tang, L. Chen, and J. Zhang, "A stepwise integration separation of variables solver for full-chip thermal uncertainty analysis," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2024.

[10] W. Jin, S. Sadiqbatcha, J. Zhang, and S. X.-D. Tan, "Full-chip thermal map estimation for commercial multi-core cpus with generative adversarial learning," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[11] J. Lu, J. Zhang, and S. X.-D. Tan, "Real-time thermal map estimation for amd multi-core cpus using transformer," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–7.

[12] J. Wen, S. Pan, N. Chang, W.-T. Chuang, W. Xia, D. Zhu, A. Kumar, E.-C. Yang, K. Srinivasan, and Y.-S. Li, "Dnn-based fast static on-chip thermal solver," in *2020 36th Semiconductor Thermal Measurement, Modeling & Management Symposium (SEMI-THERM)*. IEEE, 2020, pp. 65–75.

[13] A. Kumar, N. Chang, D. Geb, H. He, S. Pan, J. Wen, S. Asgari, M. Abarham, and C. Ortiz, "Ml-based fast on-chip transient thermal simulation for heterogeneous 2.5 d/3d ic designs," in *2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 2022, pp. 1–8.

[14] R. Ranade, H. He, J. Pathak, N. Chang, A. Kumar, and J. Wen, "A thermal machine learning solver for chip simulation," in *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, 2022, pp. 111–117.

[15] Y. Hua, Z.-Q. Wang, X.-Y. Yuan, Y. B. Li, W.-T. Wu, and N. Aubry, "Estimation of steady-state temperature field in multichip modules using deep convolutional neural network," *Thermal Science and Engineering Progress*, vol. 40, p. 101755, 2023.

[16] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[17] L. Chen, J. Lu, W. Jin, and S. X.-D. Tan, "Fast full-chip parametric thermal analysis based on enhanced physics enforced neural networks," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[18] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *Journal of Heat Transfer*, vol. 143, no. 6, p. 060801, 2021.

[19] Z. Liu, Y. Li, J. Hu, X. Yu, S. Shiau, X. Ai, Z. Zeng, and Z. Zhang, "Deepoheat: operator learning-based ultra-fast thermal simulation in 3d-ic design," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[20] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv preprint arXiv:1910.03193*, 2019.

[21] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

[22] Y. Zhang, Z. Gong, W. Zhou, X. Zhao, X. Zheng, and W. Yao, "Multi-fidelity surrogate modeling for temperature field prediction using deep convolution neural network," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106354, 2023.

[23] Y. Lyu, X. Zhao, Z. Gong, X. Kang, and W. Yao, "Multi-fidelity prediction of fluid flow based on transfer learning using fourier neural operator," *Physics of Fluids*, vol. 35, no. 7, 2023.

[24] S. Li, X. Yu, W. Xing, R. Kirby, A. Narayan, and S. Zhe, "Multi-resolution active learning of fourier neural operators," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024, pp. 2440–2448.

[25] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[26] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[27] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[28] Z.-Q. Wang, Y. Hua, H.-R. Xie, Z.-F. Zhou, Y.-B. Li, and W.-T. Wu, "Transfer learning of convolutional neural network model for thermal estimation of multichip modules," *Case Studies in Thermal Engineering*, vol. 59, p. 104576, 2024.

[29] R. E. Kessler, "The alpha 21264 microprocessor," *IEEE micro*, vol. 19, no. 2, pp. 24–36, 1999.

[30] S. Ladenheim, Y.-C. Chen, M. Mihajlović, and V. F. Pavlidis, "The mta: An advanced and versatile thermal simulator for integrated systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3123–3136, 2018.

[31] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, "U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow," *Advances in Water Resources*, vol. 163, p. 104180, 2022.

[32] W. Peng, Z. Yuan, Z. Li, and J. Wang, "Linear attention coupled fourier neural operator for simulation of three-dimensional turbulence," *Physics of Fluids*, vol. 35, no. 1, 2023.

[33] Y. Wang, Z. Xing, and W. XING, "Gar: generalized autoregression for multi-fidelity fusion," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8441–8455, 2022.

[34] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 14, no. 5, pp. 501–513, 2006.

TransRoute: A Novel Hierarchical Transistor-Level Routing Framework Beyond Standard-Cell Methodology

Chen-Hao Hsu[1], David Z. Pan[1], Laurent Perron[2], Frédéric Didier[2], Xiaoqing Xu[3], and Hao Chen[3]
[1]*ECE Department, The University of Texas at Austin, Austin, TX, USA*
[2]*Google Research, Paris, France*
[3]*Google DeepMind, Mountain View, CA, USA*
chenhaoh@utexas.edu, dpan@ece.utexas.edu, {lperron, fdid, xiaoqingxu, haojc}@google.com

***Abstract*—In advanced technology nodes, benefits from scaling have become limited in terms of power, performance, and area (PPA), necessitating improvements through design-technology co-optimization (DTCO). While the standard-cell methodology is widely used in modern VLSI design, it inherently constrains the potential of DTCO due to its abstraction at the logic level, limiting optimization at the physical level. To bridge this gap, transistor-level design methodology is desired to break the abstraction of standard cells. Existing research has explored large-scale transistor placement, but a gap remains in developing a routing framework that efficiently addresses transistor-level routing challenges. This paper proposes a pioneering transistor-level routing framework for large-scale transistor placements, along with an efficient CP-SAT (Constraint Programming-SATisfiability) formulation for routing in lower layers. Experimental results demonstrate that the proposed routing framework enables transistor-level designs to achieve significantly reduced total wirelength and high utilization rates for designs with hundreds to thousands of transistors, outperforming traditional standard-cell-based approaches in advanced technology nodes.**

I. Introduction

In advanced technology nodes, the PPA benefits from scaling have become limited, highlighting the growing importance of advancements through DTCO. DTCO involves three critical stages: technology, design enablement, and design [1]. Among these, the design enablement stage is critical for bridging technology and design by providing essential inputs to the design phase, including standard cell libraries, intellectual properties (IPs), and signoff environments. Standard cell libraries are fundamental to the VLSI design process due to their scalability, reusability, and seamless integration with electronic design automation (EDA) tools. Each library comprises pre-defined logic functions with fixed physical implementations that standardize the design framework. However, while the standard-cell methodology simplifies design and supports PPA improvement, the fixed physical layouts of standard cells impose constraints on further optimizing the physical design and achieving maximum PPA potential.

To fully unleash the potential of DTCO, it is crucial to move beyond the abstraction of standard cells by adopting a transistor-level physical design approach [2], rather than relying on pre-defined physical implementations in standard cells. Eliminating the abstraction provides greater optimization flexibility, leading to a smaller design area and reduced total wirelength compared to traditional standard-cell-based design. Furthermore, achieving a high utilization rate is challenging with the standard-cell-based approach, as it often struggles to produce LVS/DRC-clean results within a compact design area. A key limitation arises from the fixed physical implementations of standard cells, which hinder the full utilization of lower-layer routing resources (e.g., Metal0) that follow the "use it or waste it" principle. Breaking the standard-cell abstraction allows these lower-layer routing resources to be fully explored and utilized, further enhancing design optimization.

Existing work on transistor-level placement and routing primarily focuses on automating standard cell design. As design rule complexity continues to grow in advanced nodes, manual standard cell design has become increasingly challenging, necessitating advanced automation to address these complexities. Standard cell design automation is categorized into two major approaches: concurrent and sequential. Concurrent approaches [3]–[5] address transistor placement and routing simultaneously for a cell. In contrast, sequential approaches [6]–[10] divide the process into two stages: transistor placement and in-cell routing. Transistor placement [11]–[14] minimizes cell area and enhances routability by leveraging diffusion sharing, while in-cell routing [15]–[17] ensures net connectivity without design rule violations and guarantees pin accessibility. However, existing in-cell routing research primarily focuses on standard-cell-scale transistor-level routing and faces scalability challenges when applied to large-scale placements with hundreds or thousands of transistors.

Recently, TransPlace [18] proposed a transistor-level placement framework that directly places all transistors of a design without standard-cell abstraction, achieving a compact design area and reducing total half-perimeter wirelength. However, existing transistor-level routing approaches are limited to standard-cell scales, typically fewer than 50 transistors, due to scalability issues. To the best of our knowledge, no prior work has addressed the challenges of transistor-level routing for transistor placements with hundreds or even thousands of transistors. Therefore, a new transistor-level routing framework is needed to efficiently route large-scale placements.

In this paper, we present a novel hierarchical transistor-level routing framework capable of routing large-scale transistor placements with high utilization, while optimizing the total wirelength and via count, and handling various complex design rules. The main contributions of this paper are summarized as follows:

- We present the first transistor-level routing framework for a large-scale transistor placement, consisting of three main steps: partitioning, lower-layer routing, and upper-layer routing.
- We propose a transistor partitioning technique that leverages transistor placement results. This approach prioritizes the physical locations of transistors over their logical relationships (those within a standard cell), potentially yielding greater enhancements in PPA.
- We propose an efficient CP-SAT connectivity formulation for transistor-level routing, which controls the flow of each net on a grid-based routing graph while enforcing node exclusiveness through integer encoding at each vertex.
- Experimental results show that the proposed routing framework achieves LVS/DRC-clean transistor-level designs, with an average 22% reduction in wirelength and 14% reduction in via count compared to standard-cell-based designs, while maintaining a high utilization rate of around 90%.

The remainder of this paper is organized as follows. Section II gives the preliminaries and the problem formulation. Section III illustrates our proposed transistor-level routing framework, and Section IV details the proposed CP-SAT formulation for lower-layer routing within a partition. Section V shows the experimental results, and Section VI concludes the paper.

II. Preliminaries and Problem Formulation

This section presents an overview of transistor placement. Then, we formulate the transistor-level routing problem.

A. Transistor Placement

A legal transistor placement requires all transistors of a given transistor-level netlist are placed within a design canvas without overlap. When two transistors that are placed adjacently in the same row, the nets of their shared diffusion must be the same, known as diffusion sharing. Figure 1 shows a legal transistor placement with four standard-cell rows.

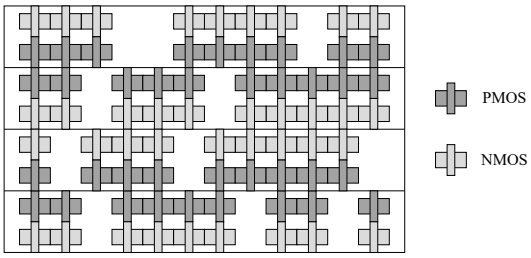


Fig. 1. A transistor placement.

B. Problem Formulation

Problem 1 (Transistor-Level Routing): Given a transistor placement and its corresponding transistor-level netlist, generate a routing solution such that all nets are routed without design rule violations, and the total wirelength and via count are minimized.

III. Transistor-Level Routing Framework

Large-scale transistor-level routing is challenging since we need to consider numerous complex design rules while optimizing total wirelength and via count. The overall flow of our proposed transistor routing consists of three stages as shown in Fig. 2: (1) partitioning, (2) lower-layer routing, and (3) upper-layer routing. First, the layout is partitioned into several partitions based on the transistor placement. From these partitions, we categorize all nets into two types: intra-partition nets and inter-partition nets. An intra-partition net has all its terminals within a single partition, while an inter-partition net has terminals across multiple partitions. Intra-partition nets are routed entirely within their corresponding partitions in the lower routing layers (typically Metal1 and below). For an inter-partition net, we connect its terminals in the lower routing layers and also create an *escape pin* in each of its partitions. These escape pins are then used to connect the inter-partition nets in the upper routing layers (typically Metal1 and above).

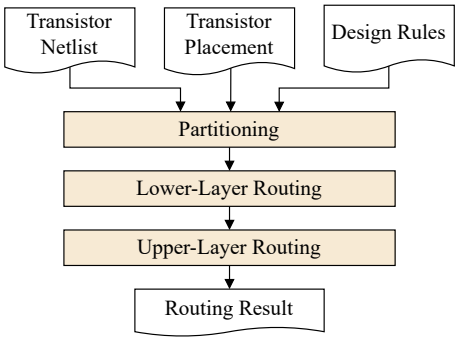


Fig. 2. The overall flow of our proposed transistor-level routing framework.

A. Partitioning

Routing all transistors directly within a macro design can lead to extremely long runtime. To address this challenge, we propose a layout partitioning strategy, where the layout is divided into partitions, each containing a set of transistors. Based on transistor placement, the layout is divided into several partitions according to transistor locations, with each partition being of similar size. Partition size plays a crucial role. If the partitions are too small, they can lead to severe routing congestion in the upper layers. On the other hand, if a partition is too large, connections within the partition may fail. In practice, we use partitions that are 2–4 standard-cell rows by 15–25 contacted poly pitches (CPPs), which are larger than individual standard cells. Furthermore, we use an Euler path (a sequence of diffusion-shared transistors without breaks) as a fundamental component for partitioning, meaning that an Euler path cannot be split across multiple partitions. As a result, partitions are not necessarily rectangular and can take on any rectilinear shape.

Fig. 3 shows the partitioning of the transistor placement in Fig. 1. It is important to emphasize that the partitioning of transistors within a design occurs at the physical level, unlike

the standard-cell methodology, which partitions transistors at the logic level. Such a distinction is crucial as partitioning at the logic level may limit PPA optimization.

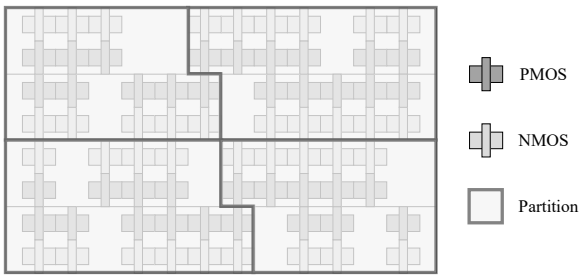


Fig. 3. Four partitions of the transistor placement.

B. Lower-Layer Routing

Lower-layer routing involves net routing for all nets and the creation of escape pins for inter-partition nets within a partition. This stage is particularly challenging due to limited routing resources, the high density of terminals, and complex design rules. Therefore, we formulate lower-layer routing of each partition into a CP-SAT problem and solve the problem by the CP-SAT solver [19], which is detailed in Section IV. Fig. 4 shows the lower-layer routing results of the four partitions in Fig. 3.

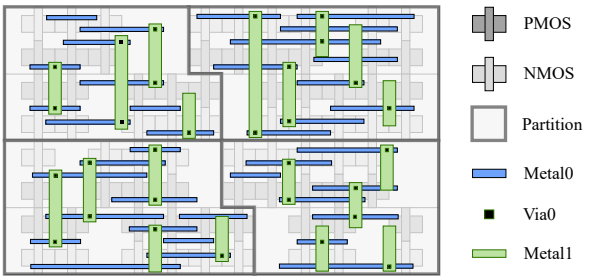


Fig. 4. Lower-layer routing for the four partitions.

C. Upper-Layer Routing

For upper-layer routing, we need to connect all inter-partition nets through their escape pins, with each net guaranteed to have one escape pin in each of its partitions. Since routing resources are more abundant in upper layers, efficiency becomes our primary concern. Connecting all escape pins is similar to the traditional routing problem for standard-cell-based design. Therefore, we could use a conventional router for standard cells to complete the upper-layer routing across all partitions.

IV. CP-SAT Formulation for Lower-Layer Routing

This section details our proposed CP-SAT formulation for lower-layer routing for each partition. We first introduce the grid-based routing graph, followed by the construction of the corresponding flow network. Based on the flow network, we then formulate the CP-SAT constraints and define the objective function.

A. Grid-Based Routing Graph

For lower-layer routing, we use a grid-based routing graph [20]. In advanced technology nodes, routing on each layer is unidirectional with routing directions alternating between layers. Each layer contains multiple tracks, and the grid points on the routing graph represent the intersection points projected from both the layer above and the layer below. Furthermore, due to the non-uniform configuration of tracks, edges in a grid graph can have varying physical lengths on each layer. In this paper, we allow the escape pins created only on the Metal1 layer. Fig. 5 shows a grid-based routing graph from the diffusion/poly layers to the Metal1 layer.

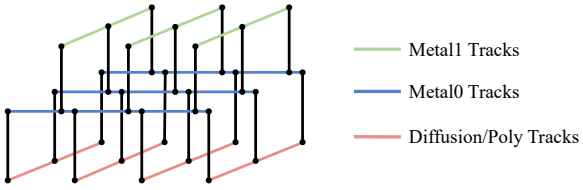


Fig. 5. A grid-based routing graph.

B. Flow Network Construction

To ensure connectivity, we construct a flow network from the grid-based routing graph, with each grid point as a vertex and each edge forming a pair of bidirectional arcs [20]. A terminal pin (diffusion/gate pin) typically has multiple access grid points. A *super-node* is created for each terminal pin. For each net, one of its terminal pins is selected as the source, while the remaining pins are designated as sinks. For the source terminal pin, arcs are constructed from its super-node to each of its access grid points. Conversely, for sink terminal pins, arcs are created from each access grid point to their super-node.

For an inter-partition net, it is necessary to create an escape pin on the escape layer (e.g., Metal1), enabling connections to pins in other partitions. To achieve this, an *escape node* is constructed, and an arc is constructed from each of the grid points on the escape layer to the escape node. Additionally, the escape node is also a sink to each inter-partition net. Fig. 6 shows the flow network based on a routing grid graph.

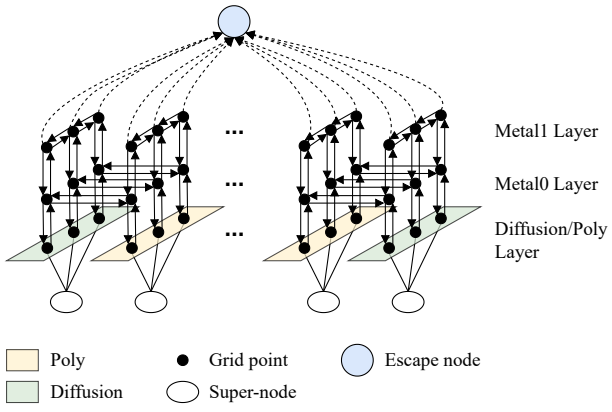


Fig. 6. The flow network based on a grid-based routing graph.

C. Constraints

The CP-SAT constraints for lower-layer routing are divided into two categories: connectivity and design rules. Connectivity constraints ensure the connection of all nets and the generation of an escape pin for each inter-partition net, with these constraints formulated based on a flow network. On the other hand, the design rule constraints are dedicated to preventing any violations of design rules, which are based on an undirected graph derived from the flow network by removing arc directions. Table I summarizes the notations used in this paper.

TABLE I
NOTATIONS USED IN THIS PAPER

Symbol	Description
N	The set of nets in a partition.
N_{inter}	The set of inter-partition nets in a partition.
V/A	The set of vertices/arcs.
V_u^i/V_u^o	The set of vertices where vertex u has an in/out-arc.
s_n	The source node for net n.
T_n	The set of sink nodes for net n.
$t_{n,i}$	The i^{th} sink node for net n.
v_e	The escape node.
$a_{u,v}$	Boolean. Indicate if arc (u,v) is selected.
$f_{u,v}$	Integer. Flow on arc (u,v).
c_u	Integer. The encoding of vertex u.

1) Connectivity: The net routing problem can be formulated as a terminal Steiner forest problem. To construct terminal Steiner forest, we can leverage network flow by controlling the flow between nodes and the flow directions. For a net to be fully connected, there must be a flow from the source node to each of its sink nodes without overlapping with the flows of other nets.

- *Flow Conservation Constraint*: For each vertex in the flow network, the inflow and outflow must be the same for flow conservation:

$$\forall u \in V, \text{inflow}(u) = \text{outflow}(u), \tag{1}$$

where

$$\text{inflow}(u) = \begin{cases} |T_n|, & \text{if } u = s_n, n \in N \\ \sum_{v \in V_u^i} f_{v,u}, & \text{otherwise,} \end{cases}$$
$$\text{outflow}(u) = \begin{cases} 1, & \text{if } u \in T_n \setminus \{v_e\}, n \in N \\ |N_{inter}|, & \text{if } u = v_e \\ \sum_{v \in V_u^o} f_{u,v}, & \text{otherwise.} \end{cases} \tag{2}$$

The inflow of a source node is fixed to the number of sinks, the outflow of each sink node (except the escape node) is fixed to one, and the outflow of the escape node is fixed to the number of inter-partition nets. Note that the domain of each flow variable $f_{u,v}$ is $[0, \max_{n \in N} \deg(n) - 1]$.

- *Selected Arc Constraint*: Arc (u,v) is selected if and only if it has non-zero flow:

$$\forall (u,v) \in A, a_{u,v} \leftrightarrow (f_{u,v} \geq 1). \tag{3}$$

- *In/Out-Arc Count Constraint*: To ensure that each net is connected by a tree, at most one of the in-arcs can be selected for each node, except the escape node:

$$\forall u \in V \setminus \{v_e\}, \sum_{v \in V_u^i} a_{v,u} \leq 1. \tag{4}$$

For each source node, at most one out-arc can be selected to ensure that the source node is a leaf node of a tree:

$$\forall n \in N, \sum_{v \in V_{s_n}^o} a_{s_n,v} \leq 1. \tag{5}$$

- *Vertex and Edge Exclusiveness Constraint*: The flow conservation ensures the connectivity of all nets, but each node or edge can be assigned to at most one net. To enforce vertex exclusiveness, each vertex u has an integer encoding c_u, typically set to the net index (i.e., $1, \ldots, |N|$). The encoding of all the source and sink nodes (except the escape node) is fixed to their net index. Moreover, if an arc (u,v) is selected, then its two vertices must have the same encoding:

$$\forall (u,v) \in A, v \neq v_e, a_{u,v} \rightarrow (c_u = c_v). \tag{6}$$

Note that only node exclusiveness needs to be constrained since edge exclusiveness is implicitly ensured.

Fig. 7 shows an example of terminal Steiner forest construction for two nets, n_1 and n_2, using network flow. Net n_1 includes a source s_1 with three sinks $t_{1,1}$, $t_{1,2}$, and $t_{1,3}$. Similarly, net n_2 has a source s_2 and four sinks $t_{2,1}$, $t_{2,2}$, $t_{2,3}$, and $t_{2,4}$. The inflows of s_1 and s_2 are fixed to 3 and 4 respectively, while the outflow of each sink is fixed to 1.

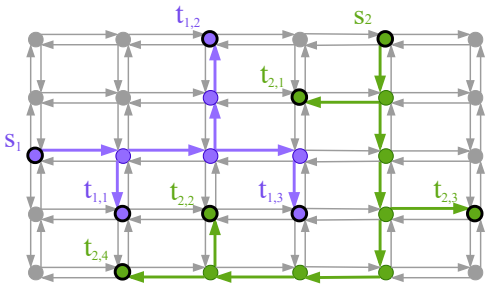


Fig. 7. Terminal Steiner forest construction using network flow.

Our proposed CP-SAT formulation is more efficient than the SAT-friendly integer linear programming (ILP) formulation in [20], which uses only 0-1 integer variables. Unlike [20], which decomposes a k-pin net into $(k-1)$ source-to-sink pairs connected by independent commodity flows, our approach directly generates a Steiner tree by controlling the flow of a net. This allows implicit edge exclusiveness through a single integer variable for node encoding, whereas [20] requires n 0-1 variables per edge, with constraints ensuring at most one is active for edge exclusiveness, where n is the net count.

2) Design Rules: Connectivity constraints ensure all nets are connected with node and edge exclusiveness, but design rules are not addressed. To handle design rules, we add CP-SAT constraints based on the undirected graph derived from a directed graph used for connectivity. Several common design rules are considered, such as minimum area, end-of-line spacing, via spacing, etc. Based on the undirected graph, we can formulate CP-SAT constraints for the design rules, following a similar approach to that in the works [3], [20].

D. Objective

The objective of lower-layer routing is to minimize the total wirelength and via costs. Each arc in the routing graph, excluding those connected to super-nodes or the escape node, represents either a wire or a via. Wire costs are calculated as the product of wirelength and a layer-specific unit cost, while via costs are determined by a user-defined parameter. Thus, the objective in the CP-SAT formulation is $\sum_{a \in A} \text{cost}(a)$.

V. Experimental Results

The proposed partitioning and lower-layer routing methods were implemented in the C++ programming language with the CP-SAT solver [19]. For upper-layer routing, we used the OpenROAD project [21] and the Cadence Innovus tool [22]. This section describes the benchmark generation process and provides a detailed comparison of standard-cell-based designs versus transistor-level designs, along with an evaluation of the efficiency of the proposed CP-SAT connectivity formulation.

A. Benchmarks

Our transistor-level netlists are generated from standard-cell-based designs. First, we synthesize an RTL design using a standard-cell library to obtain a gate-level netlist. The gate-level netlist is then flattened by replacing each standard cell with its corresponding SPICE transistor-level netlist. The resulting transistor-level netlist includes all transistors from the original standard-cell-based design, but with the hierarchy fully eliminated.

Table II gives the benchmark statistics, where "Tech.", "#Trans.", "#Net", "#SDC", and "SDC Area" denote the technology, transistor count, signal net count, standard cell (SDC) count, and total standard cell area (μm^2), respectively. To evaluate the proposed transistor-level routing framework, there are totally eight designs across two advanced industry technology nodes: Tech1 (Design1 to Design4) and Tech2 (Design5 to Design8).

TABLE II
Benchmark statistics

Benchmark	Tech.	#Trans.	#Net	#SDC	SDC Area
Design1	Tech1	72	40	15	0.41
Design2	Tech1	162	90	23	0.87
Design3	Tech1	666	300	76	3.38
Design4	Tech1	684	314	84	3.51
Design5	Tech2	286	129	28	1.14
Design6	Tech2	633	282	55	2.49
Design7	Tech2	656	336	77	2.68
Design8	Tech2	1969	936	180	7.72

B. Overall Quality Comparison

To evaluate transistor-level routing, we compare the post-routing results of transistor-level designs with their corresponding standard-cell-based designs. Two different placement and routing flows are used for the standard-cell-based designs, depending on the technology node. The open-source OpenROAD [21] design flow is used for Tech1 (Design1 to Design4), while the commercial Cadence Innovus flow is employed for Tech2 (Design5 to Design8). Furthermore, transistor-level placements for each design are generated using a placement framework based on TransPlace [18]. All the transistor-level designs are LVS/DRC-clean.

Table III summarizes a comprehensive comparison of the post-routing results, where "Total WL", "Total Via", "Design Area", and "Util." represent the total wirelength (μm), via count, design area (μm^2), and utilization rate, respectively. The design area refers to the bounding box of all standard cells or transistors, and the utilization rate, equivalent to that in standard-cell-based designs, is calculated by dividing the total SDC area by the design area. The transistor-level designs consistently exhibit significantly less total wirelength and via count compared to the standard-cell-based designs, with an average wirelength reduction of 23% for Tech1 and 21% for Tech2, and via count reductions of 25% for Tech1 and 2% for Tech2. These improvements can be attributed to two key factors. First, transistor-level placement, without the constraints of standard-cell abstraction, allows for more diffusion sharing opportunities, enabling designs to fit into a smaller area and reducing the overall HPWL. Second, the absence of standard-cell abstraction enables more nets to be routed in the lower layers, minimizing the need for detours into upper layers. Furthermore, transistor-level designs achieve significantly smaller design areas, with reductions of 29% for Tech1 and 31% for Tech2. As a result, these designs reach a high utilization rate of around 90%, which is challenging to achieve with DRC-clean results in standard-cell-based designs.

C. Wirelength Breakdown

Fig. 8 presents the breakdown of wirelength by layers for both the standard-cell-based design and transistor-level (T-level) design of Design3, Design4, Design7, and Design8. For Design3 and Design4, the transistor-level designs show significantly reduced wirelength in the upper layers (Metal2 and above), while the lower layers have slightly more wirelength compared to the standard-cell-based designs. However, the total wirelength in the transistor-level designs remains lower. It is important to note that lower-layer routing resources follow the "use it or waste it" principle. In standard-cell-based designs, routing on Metal0 is restricted due to fixed routing within standard cells. In contrast, transistor-level designs can fully leverage the lower-layer routing, as they are not constrained by the fixed physical implementations of standard cells. For Design7 and Design8, the transistor-level designs have significantly less wirelength across all layers, except for a slight increase in Metal3. This minor increase helps avoid congestion in lower layers due to the smaller design area compared to standard-cell-based designs.

TABLE III
OVERALL COMPARISON BETWEEN STANDARD-CELL-BASED DESIGNS AND TRANSISTOR-LEVEL DESIGNS

Benchmark	Standard-Cell-Based Design				Transistor-Level Design			
	Total WL	Total Via	Design Area	Util.	Total WL	Total Via	Design Area	Util.
Design1	15.71	67	0.81	0.51	8.79	26	0.41	1.00
Design2	36.37	150	1.38	0.63	26.50	100	0.97	0.89
Design3	167.76	618	4.70	0.72	155.40	609	3.85	0.88
Design4	181.20	646	4.79	0.73	153.80	611	3.76	0.93
Avg. Ratio	1.00	1.00	1.00	-	0.77	0.75	0.71	-
Design5	54.26	229	1.75	0.65	37.70	191	1.18	0.97
Design6	115.11	500	4.09	0.61	86.21	473	2.76	0.90
Design7	160.74	683	4.19	0.64	129.84	652	2.92	0.92
Design8	506.49	1907	11.83	0.65	451.83	2263	8.61	0.90
Avg. Ratio	1.00	1.00	1.00	-	0.79	0.98	0.69	-

TABLE IV
COMPARISON OF CONNECTIVITY FORMULATIONS

Benchmark	Grid Sizes	#Nets	#Nodes	Max Deg.	Time Limit (s)	[20]		Ours	
						Feasible Time (s)	Length	Feasible Time (s)	Length
Graph1	8×8×3	10	27	5	60	4.20	84.30	1.03	81.70
Graph2	10×10×4	10	32	7	120	16.52	140.20	14.67	118.90
Graph3	10×10×3	15	35	3	600	236.61	144.00	30.37	130.60
Graph4	15×15×4	12	37	7	900	260.99	295.50	181.58	196.20
Graph5	12×12×3	20	40	2	1200	1055.76	279.00	8.71	203.20
Avg. Ratio						1.00	1.00	0.39	0.82

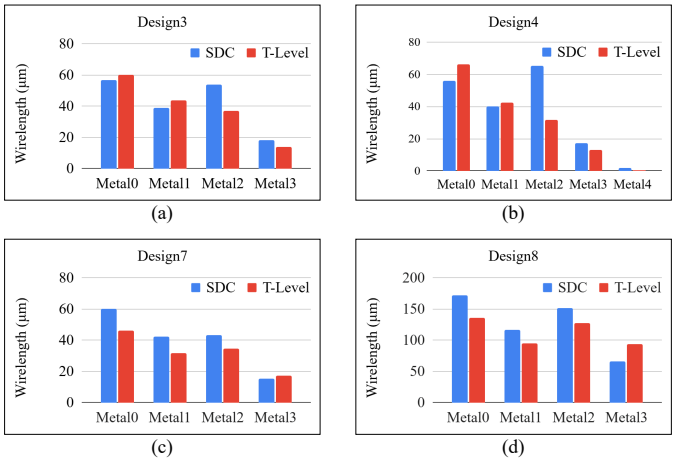


Fig. 8. Wirelength breakdown by layers: (a) Design3, (b) Design4, (c) Design7, and (d) Design8.

D. Connectivity Formulation Comparison

To evaluate the efficiency of our proposed CP-SAT connectivity formulation, we implemented both the SAT-friendly ILP formulation [20] and our CP-SAT formulation using the CP-SAT solver. We generated five synthetic graphs with various grid sizes and net counts for terminal Steiner forest construction, as given in Table IV, where "#Nets", "#Nodes", and "Max Deg." represent the net count, total node count, and maximum net degree, respectively. A time limit was set for each graph based on the problem size. The CP-SAT solver finds an initial feasible solution and continues optimizing until it reaches an optimal solution or the time limit. Due to the solver's non-deterministic nature, each formulation was run 10 times per graph to measure the average time to find the first feasible solution and the average total length.

Table IV compares the performance of the SAT-friendly ILP and CP-SAT formulations. For graph4 and graph5, the solver timed out in 4 and 6 out of 10 runs, respectively, without finding a feasible solution when using the SAT-friendly ILP formulation. On average, the proposed formulation finds the first feasible solution 61% faster and reduces total net length by 18% within the time limit compared to the SAT-friendly ILP formulation. The results demonstrate that the proposed CP-SAT formulation allows the solver to find a feasible solution more efficiently, enabling it to achieve a better-optimized solution within the given time limit.

VI. CONCLUSION

In this paper, we have presented an innovative transistor-level routing framework capable of routing all nets within large-scale transistor placements. The layout partitioning method has been proposed to accurately reflect the proximity of transistors at the physical level, rather than relying on logic-level relationships, facilitating the optimization of the ultimate layout. Furthermore, our proposed CP-SAT-based lower-layer routing algorithm efficiently routes all nets with a novel connectivity formulation, optimizing both wirelength and via costs while handling complex design rules in cutting-edge technology nodes. Experimental results have demonstrated the transistor-level design's significant improvements over standard-cell-based designs in wirelength, via count, and design area. Transistor-level routing is the final step toward achieving complete transistor-level design for macro designs, and this work marks the first attempt to fully realize this goal. As DTCO becomes increasingly important in advanced technology nodes, we believe that the transistor-level design approach, beyond standard-cell methodology, can fully exploit the potential of DTCO.

REFERENCES

[1] S. Choi, J. Jung, A. B. Kahng, M. Kim, C.-H. Park, B. Pramanik, and D. Yoon, "PROBE3.0: A systematic framework for design-technology pathfinding with improved design enablement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[2] R. Hentschke, "Physical design at the transistor level beyond standard-cell methodology," in *Proceedings of ACM International Symposium on Physical Design (ISPD)*, 2022, pp. 141–143.

[3] D. Lee, D. Park, C.-T. Ho, I. Kang, H. Kim, S. Gao, B. Lin, and C.-K. Cheng, "SP&R: SMT-based simultaneous place-and-route for standard cell synthesis of advanced nodes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 10, pp. 2142–2155, 2020.

[4] C.-K. Cheng, C.-T. Ho, D. Lee, and B. Lin, "Multirow complementary-FET (CFET) standard cell synthesis framework using satisfiability modulo theories (SMTs)," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 7, no. 1, pp. 43–51, 2021.

[5] D. Lee, C.-T. Ho, I. Kang, S. Gao, B. Lin, and C.-K. Cheng, "Many-tier vertical gate-all-around nanowire FET standard cell synthesis for advanced technology nodes," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 7, no. 1, pp. 52–60, 2021.

[6] P. Van Cleeff, S. Hougardy, J. Silvanus, and T. Werner, "BonnCell: Automatic cell layout in the 7-nm era," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 10, pp. 2872–2885, 2019.

[7] Y.-L. Li, S.-T. Lin, S. Nishizawa, H.-Y. Su, M.-J. Fong, O. Chen, and H. Onodera, "NCTUcell: A DDA-aware cell library generator for FinFET structure with implicitly adjustable grid map," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[8] H. Ren and M. Fojtik, "NVCell: Standard cell layout in advanced technology nodes with reinforcement learning," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1291–1294.

[9] C.-T. Ho, A. Ho, M. Fojtik, M. Kim, S. Wei, Y. Li, B. Khailany, and H. Ren, "NVCell 2: Routability-driven standard cell layout in advanced nodes with lattice graph routability model," in *Proceedings of ACM International Symposium on Physical Design (ISPD)*, 2023, pp. 44–52.

[10] C.-T. Ho, A. Chandna, D. Guan, A. Ho, M. Kim, Y. Li, and H. Ren, "Novel transformer model based clustering method for standard cell design automation," in *Proceedings of ACM International Symposium on Physical Design (ISPD)*, 2024, pp. 195–203.

[11] A. Sorokin and N. Ryzhenko, "SAT-based placement adjustment of FinFETs inside unroutable standard cells targeting feasible DRC-clean routing," in *Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 159–164.

[12] M. Cardoso, A. Bubolz, J. Cortadella, L. Rosa, and F. Marques, "Transistor placement for automatic cell synthesis through boolean satisfiability," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[13] K. Jo and T. Kim, "Optimal transistor placement combined with global in-cell routing in standard cell layout synthesis," in *Proceedings of IEEE International Conference on Computer Design (ICCD)*, 2021, pp. 517–524.

[14] K. Baek and T. Kim, "CSyn-fp: Standard cell synthesis of advanced nodes with simultaneous transistor folding and placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[15] N. Ryzhenko and S. Burns, "Standard cell routing via boolean satisfiability," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2012, pp. 603–612.

[16] J. Cortadella, J. Petit, S. Gomez, and F. Moll, "A boolean rule-based approach for manufacturability-aware cell routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 3, pp. 409–422, 2014.

[17] Y.-L. Li, S.-T. Lin, S. Nishizawa, and H. Onodera, "Mcell: multi-row cell layout synthesis with resource constrained max-sat based detailed routing," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–8.

[18] C.-H. Hsu, X. Xu, H. Chen, D. Ruic, and D. Z. Pan, "TransPlace: A scalable transistor-level placer for VLSI beyond standard-cell-based design," in *Proceedings of IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2024, pp. 312–318.

[19] L. Perron and F. Didier. CP-SAT. Google. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver/

[20] D. Park, D. Lee, I. Kang, C. Holtz, S. Gao, B. Lin, and C.-K. Cheng, "Grid-based framework for routability analysis and diagnosis with conditional design rules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 12, pp. 5097–5110, 2020.

[21] T. Ajayi and D. Blaauw, "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019, pp. 1–6.

[22] Innovus implementation system. Cadence. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html

Finding the Pareto Frontier of Low-Precision Data Formats and MAC Architecture for LLM Inference

Brian Crafton[1], Xiaochen Peng[1], Xiaoyu Sun[1], Ashwin Lele[1], Bo Zhang[1], Win-San Khwa[2], and Kerem Akarvardar[1]
[1]TSMC Corporate Research, San Jose, USA. [2]TSMC Corporate Research, Hsinchu, Taiwan.

***Abstract*—To accelerate AI applications, numerous data formats and physical implementations of matrix multiplication have been proposed, creating a complex design space. This paper studies the efficient MAC implementation of the integer, floating-point, posit, and logarithmic number system (LNS) data formats and Microscaling (MX) and Vector-Scaled Quantization (VSQ) block data formats. We evaluate the area, power, and numerical accuracy (evaluated as signal-to-quantization noise ratio) of >25,000 MAC designs spanning each data format and several key design parameters such as the inner product size and accumulation width. We find that for the same numerical accuracy, pareto optimal MAC designs with emerging data formats (LNS16, MXINT8, VSQINT4) achieve 1.8×, 2.2×, and 1.9× TOPs/W improvement compared to FP16, FP8, and FP4 dot product implementations.**

I. Introduction

The computational requirements of modern artificial intelligence (AI) applications have given rise to an entire industry for hardware acceleration. While there exist several different system-level architectures for processing these applications, all leading AI processors leverage quantization and custom hardware for matrix multiplication.

Quantization enables narrower data formats by casting higher precision (FP32) weights and activations to lower precision data formats such as INT8 or FP8 [10], [13], [26]. Using fewer bits per weight or activation yields a linear improvement in effective memory capacity and bandwidth and a quadratic improvement in area and energy efficiency. From these improvements, quantization has achieved fast and widespread adoption in commercial designs. NVIDIA's recent addition of FP4 and FP6 to Blackwell is one such example [37]. With both commercial interest and success, a myriad of new data formats and so-called *block data formats* [7], [9] have been proposed. Together, these data formats form a rich design space with improved dynamic range at low area and power overhead.

A second factor contributing to improved performance is custom matrix multiplication hardware. Most modern commercial AI accelerators feature some type of matrix multiplication accelerator such as the Tensor Core in NVIDIA's GPUs [1] or Google's Tensor Processing Unit (TPU) [20]. Within these matrix multiplication accelerators, special fused dot product units can be used that are more efficient than conventional multipliers and adders [16]. In addition to reduced logic complexity, custom matrix multiplication units process more operations in parallel reducing the processor datapath overhead.

The combination of new data formats and physical implementations of matrix multiplication creates a complex design space warranting exploration. The trend towards narrower data formats and reduced precision hardware is ultimately a trade-off between numerical accuracy and hardware efficiency. Therefore, to effectively evaluate the design space, it is necessary to compare data formats and their hardware implementations based on both numerical accuracy and area and energy efficiency.

In this work, we evaluate the efficient MAC implementation of the integer, floating-point, posit [15], and logarithmic number system (LNS) data formats and Microscaling (MX) [9] and Vector-Scaled Quantization (VSQ) [7] block data formats. We design, implement, verify, and evaluate over 25,000 MAC implementations across these data formats and various MAC design parmeters: inner product size, outer product size, accumulation width, etc. Next, we present the pareto optimal MAC designs considering power, performance, and numerical accuracy across several large language model (LLM) benchmarks. First, we find that at the same numerical accuracy, LNS, MX, and VSQ achieve higher area and energy efficiency than integer and floating-point. Second, we find that choosing the optimal MAC design parameters, especially inner product size and accumulation width, is essential to strong area and energy efficiency. Lastly, we show that at the same numerical accuracy, pareto optimal MAC designs with emerging formats at different bit-widths (LNS16, MXINT8, VSQINT4) achieve 1.8×, 2.2×, and 1.9× TOPs/W improvement compared to FP16, FP8, and FP4 dot product implementations.

II. Background and Related Work

A. Data Formats for AI

To evaluate the design space of data formats, we study and evaluate four common data formats with unique properties: integer, floating-point, posit, and logarithmic number system (LNS). To introduce the key differences between the data formats, we visualize the 16 real values (representations) of each data format with 4-bit precision in Fig. 1. Each data format yields a different dynamic range (maximum value divided by minimum value) and step sizes. A large dynamic range is necessary for when the distribution of weights or activations has a large variance, however when the data format's dynamic range is unnecessarily large, many bit patterns will be wasted. Therefore, it is ideal for the target data format's dynamic range to match the variance of the data distribution as closely as possible.

The **integer** (INT) and **floating-point** (FP) data formats are the commonly used data formats in AI models. For both integer and floating-point, we follow the specification in [32]. For integer, an implicit scale factor and symmetric range is used such that the raw value range is: $[-1\frac{63}{64}, +1\frac{63}{64}]$. For floating-point, we follow [32] for E5M2, E4M3, E3M2, and E2M1 and [2] for FP32 and FP16.

The **posit** data format [15] is a relatively new alternative to floating-point with a larger dynamic range, fewer wasted bit patterns, and a tapered distribution to allocate more bit patterns near zero. Similar to floating-point, posit has a sign, exponent (of size "*es*"), and fraction - which operates similar to a mantissa but has no subnormal behavior. Posit also adds a regime field that is run-length encoded. This means the field sizes and locations are dynamic, requiring complex decoding logic [24]. However, these dynamic fields create a *tapered* distribution with higher dynamic range and more bit patterns allocated near zero. Fig. 1 includes a 4-bit posit with a 1-bit exponent field (E1).

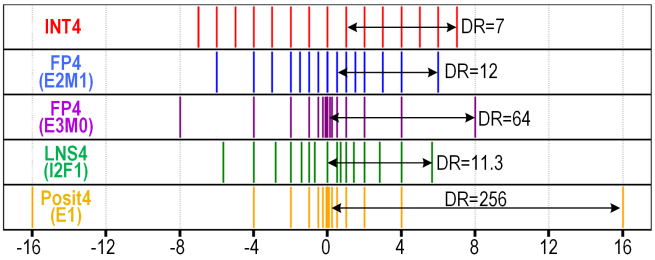


Fig. 1. The 16 real values for INT4, FP4 E2M1 & E3M0, LNS4, and posit4. Dynamic range (DR) varies between data formats.

The **logarithmic number system** (LNS) was originally used to simplify multiplications by moving them to the logarithm domain where they become additions [35]. Since then, several works [19], [29], [41] have applied the same idea to reduce the cost of matrix multiplication in AI applications. In floating-point terms, LNS uses a sign bit (*s*) and exponent to encode a number and neglects having a mantissa or fraction field. To improve the precision of LNS, recent work proposes the multi-dimensional logarithmic number system (MDLNS) [41] or multi-base logarithmic number system [19]. For this work, we follow a special case of MDLNS demonstrated in [41], where the exponent is replaced with a fixed-point number containing an integer and fraction component. For example, the 8-bit (I4F3) LNS with an exponent of 4 integer bits (*int*) and 3 fraction bits (*frac*) can be modeled as a two-base MDLNS of the following form:

$$(-1)^s \cdot 2^{int} \cdot \sqrt[8]{2}^{frac}$$

B. Block Data Formats

A relatively new type of data formats proposed to improve accuracy at low-precision are called block data formats [9]. At a high level, block data formats use a shared scale or exponent for a *block* of data, where a block is typically 32 to 64 adjacent weights or activations from the same matrix or vector. The advantage of a shared scale is that it gives a higher effective dynamic range for a block of data at a low memory, area, and power overhead. The two block data formats we evaluate are **Microscaling** (MX) [33] and **Vector-Scaled Quantization** (VSQ) [7]. MX defines a standard where an 8-bit exponential scale factor (E8M0) is shared amongst a block of 32 weights or activations. VSQ applies a linear scale for a vector of 64 weights or activations [7], [22]. Although prior work uses FP32 [7] and INT8 [22] scales, we use an FP8 (E5M2) scale because it achieves the best tradeoff between hardware cost and numerical accuracy in our evaluations. The key advantage of VSQ, over MX, is that the scale is not constrained to a power of 2 and thereby offers higher precision. However, re-scaling the data is more expensive and requires multiplication instead of shift operations. Although several other works fit the criteria of block data formats [8], [12], [23], [36], we conclude each is similar in design to either MX and VSQ in that they use a shared exponent or linear scale for a block of data.

C. Matrix Multiplication Unit Design

Modern AI processors utilize specialized matrix multiplication hardware to benefit from both instruction-level parallelism and improved area and energy efficiency. Traditionally, GPUs perform single-instruction multiple-data (SIMD) operations on a *warp* or 32 threads executing the same operation on different data. In this execution model, 32 independent multiply-and-accumulate instructions operate on 32 different data. While this model provides the programmer with generality, it fails to exploit the area and energy efficiency advantage of a fused operation. The simplest example of this is the DP4 (4-input dot product) instruction, that implements: $C = \sum_{n=1}^{4} A_i \cdot B_i$, where A and B are 4-input vectors.

The implementation of the dot product and how various factors contribute to power, performance, area, and accuracy has been the subject of several studies [14], [17], [21], [27], [30], [34]. The dot product unit was conventionally implemented as a network of multipliers and adders, but the initial implementation of the so-called "fused" dot product demonstrated how area and power could be reduced by up 40% by fusing the network of multipliers and adders [34]. In Fig. 2, we illustrate the design of the 4-input FP8 (E4M3) dot product unit with MX scaling and FP32 (or FP16) accumulation. In the fused dot product, multiplication is performed similar to the baseline approach, but the alignment and accumulation of all four products is done together before any normalization or rounding. Following accumulation, only a single normalization and rounding step is performed, thereby reducing area, power, latency. Two designs from Intel [17], [21] extend this idea to 32 input terms which further amortizes the cost of alignment and normalization, effectively improving the area and energy efficiency.

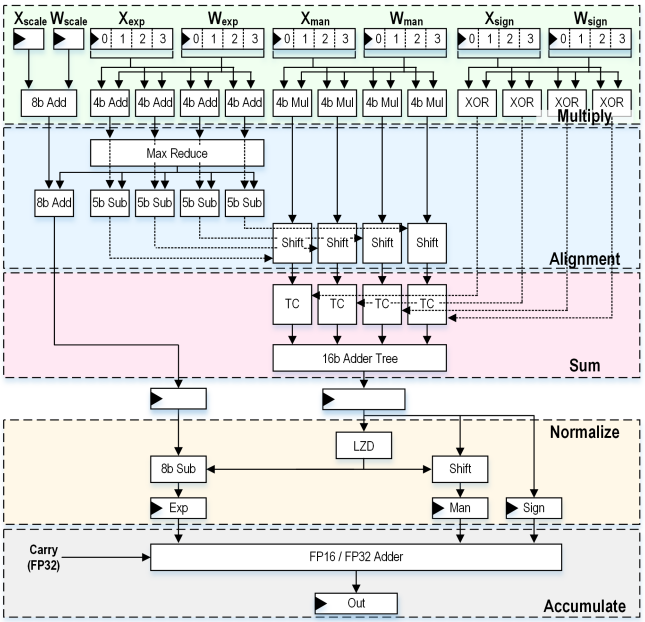


Fig. 2. Schematic of a 4-input fused FP8 (E4M3) dot product unit. Color coding corresponds to area and power breakdown in Fig. 4.

As an extension of the dot product operation (vector-vector multiplication), several commercial designs have introduced dedicated hardware to execute vector-matrix multiplication and matrix-matrix multiplication as a single operation. The Tensor Core, introduced in NVIDIA's Volta architecture [1], is the best known example. The first version of the Tensor Core executes a single 4×4×4 matrix multiplication per clock and improves FP16 throughput by 6× (21.2→125 TOPs) over the previous generation (Pascal) that used only CUDA cores. Subsequent versions of the Tensor Core increased the dimensions from 4×4×4 (Volta) to 8×8×4 (Ampere) to 16×8×4 (Hopper), resulting in throughput improvement. While the primary advantage of performing fused matrix-matrix multiplication is instruction-level parallelism, both pre-processing logic and input registers can be shared resulting in modest area and power savings.

The fused dot product can achieve higher numerical accuracy than the conventional design because it avoids truncation between each operation. However, most implementations of the fused dot product truncate the products following alignment to avoid an unnecessarily large internal datapath (over 90-bits for FP16). There exists a wide range of accumulation widths across commercial designs ranging from 22-bits in Samsung's mobile processor [31], 24-bits in NVIDIA's Tensor Core [16], 37-bits [17] in Intel's accelerator, and 69-bits in ARM's DP4 IP [27]. However, for machine learning applications, [4] demonstrates that truncation as low as 16-bits (for FP16) avoids any significant numerical accuracy loss.

With the emergence of block data formats such as MX and VSQ, several designs [9], [27], [37] have implemented block scaling as part of the MAC design. Conventional layer-wise or channel-wise quantization, where one scale may be shared by thousands of values

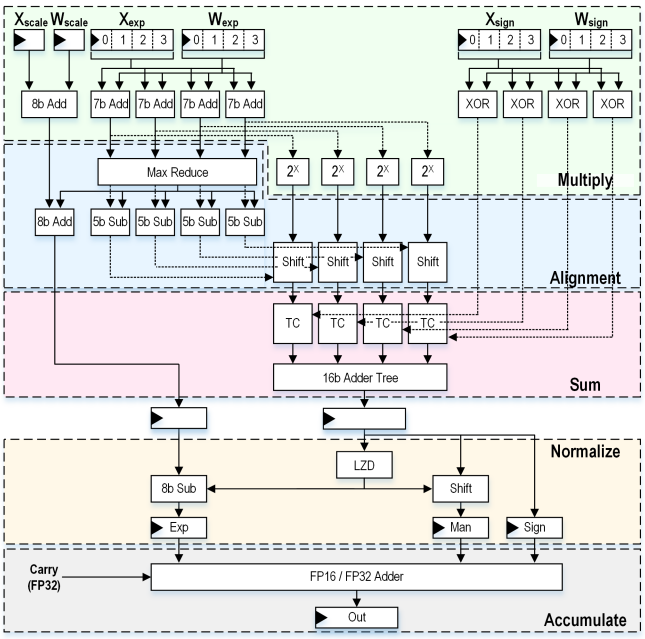


Fig. 3. Schematic of a 4-input fused LNS8 (I4F3) dot product unit. Color coding corresponds to area and power breakdown in Fig. 4.

in a tensor, can be executed efficiently in software. However, the fine-grained scaling requirements of block data formats necessitate integration of the scale inside the MAC operation to maximize efficiency [9], [39]. In Fig. 2, we show how MX scaling can be integrated into the dot product by first adding the scales in parallel, and then adding them to the maximum exponent prior to normalization.

III. MAC Design and Evaluation

A. MAC Architecture

In Fig. 2 we presented the architecture for a configurable 4-input floating-point dot product unit. For posit and LNS, we adapt this design so we can fairly evaluate the design optimizations applied to each data format. Following prior work [24], [30], we find that the posit dot product is most efficiently implemented by first converting posit to floating-point, and then performing the dot product in floating-point. It should be noted that the posit to floating-point conversion is costly in terms of both area and power, and recent work redefines posit such that the decoding step is not as expensive [18]. Posit also has dynamic field sizes requiring higher precision multipliers and adders.

Fig. 3 shows our 4-input dot product design for LNS. To implement 8-bit (1-bit sign, 4-bit integer, 3-bit fraction or I4F3) LNS, 7-bit addition replaces the mantissa multiplication and exponent addition blocks. Next, the fraction component of the result (3 LSBs) is converted from the log domain using a base-2 fixed point exponential function. For our Verilog implementation, we use the associated fixed-point Synopsys Designware IP (DW_exp2). The integer component of the result (5 MSBs) can be processed by shifting the fraction component. The summation, normalize, and accumulate portion of the dot product remains mostly similar to floating-point.

B. Evaluation Methodology

To evaluate the MAC implementations of the various data formats, we consider three key figures of merit: area efficiency, energy efficiency, and numerical accuracy. To evaluate area and energy efficiency, we design, implement, verify, and optimize Verilog models for each data format at different bit precisions. The Verilog model for each data format is parameterized so we can sweep the various optimization parameters studied in the following section (inner product size, accumulation width, pipeline stages, etc). To verify each MAC design, we create a Python emulator and verify against 10,000 test vectors with special attention to special cases such as NaN, infinity, sub-normal, and zero. We synthesize each MAC design in 5nm CMOS at 1 GHz using 750mV VDD (TT corners) and collect area (um^2) and critical path (ns) results. Next, we simulate the synthesized MAC designs using 100 vectors sampled from 4 LLM benchmarks and extract power (mW) using Synopsys PrimePower. In Fig. 5 to Fig. 10 we alternate between TOPs/W and TOPs/mm^2 to address both figures of merit. We find improvements in TOPs/W and TOPs/mm^2 have strong correlation, especially at fixed VDD.

While area efficiency and energy efficiency are common figures of merit for evaluating AI hardware, numerical accuracy is equally important for a fair comparison. This is because the major trade-off when moving to lower precision data formats or efficient MAC designs is numerical accuracy. Following [9], we evaluate numerical accuracy using signal-to-quantization noise ratio (SQNR). However, we compute SQNR using the result of quantized matrix multiplication to account for numerical error introduced by the MAC design. We compute SQNR of each MAC design based on the following equation:

$$SQNR = -10\log(\frac{||Q(X)\cdot Q(W) - X\cdot W||^2}{||X\cdot W||^2})$$

where X and W are activations and weights in FP32, Q is the quantization function that maps FP32 to the quantized data format (ex. FP8), and $||X|| = (\sum_{i=1}^{N} x_i^2)^{1/2}$.

SQNR is a simple yet powerful figure of merit because it scales linearly with the number of bits in the data format and allows us to compare the data formats and their MAC implementation fairly. However, the relationship between SQNR and application accuracy is model and task dependent and typically non-linear. But if we assume the relationship between SQNR and application accuracy is mostly monotonic, then we can effectively compare data formats achieving similar SQNR. To evaluate the SQNR of each MAC design, we sample random vector-vector products from each layer in OpenELM-1B [28] and Llama3.2-1B [38] on the Hellaswag dataset [40] and BERT-Large [25] and RoBERTa-Large [25] on the STSB dataset [6]. We first compute the FP32 result of the vector-vector products as reference. Next, we scale the vectors by the ratio of the max value in the target data format and the max value in the tensor they are sampled from. This ensures the full range of the target data format is used. Then, we quantize the vectors to the target data format and compute the vector-vector product with each MAC design. To execute a large vector-vector product with a smaller MAC design, we break it down into several operations and accumulate the partial sums in FP32 precision. Lastly, we compute the SQNR against the FP32 reference.

C. MAC Design Characterization

In Fig. 4, we present a baseline evaluation (area, power, and SQNR) of each data format with a nominal MAC design to serve as reference for the design optimizations in the following section. For each data format we use a 16-input MAC, with an accumulation width of 16, and 1 pipeline stage. To characterize area and power, we show the block-level breakdown of each MAC implementation in 5nm CMOS. Based on the results in Fig. 4, we make the following initial observations: ① INT8, FP8 (E4M3), and LNS8 (I4F3) consume similar area and power because of the design parameters. INT4 is visibly more power and area efficient than FP4 (E2M1) and LNS4 (I2F1) because accumulation width is fixed at 16-bits. In the

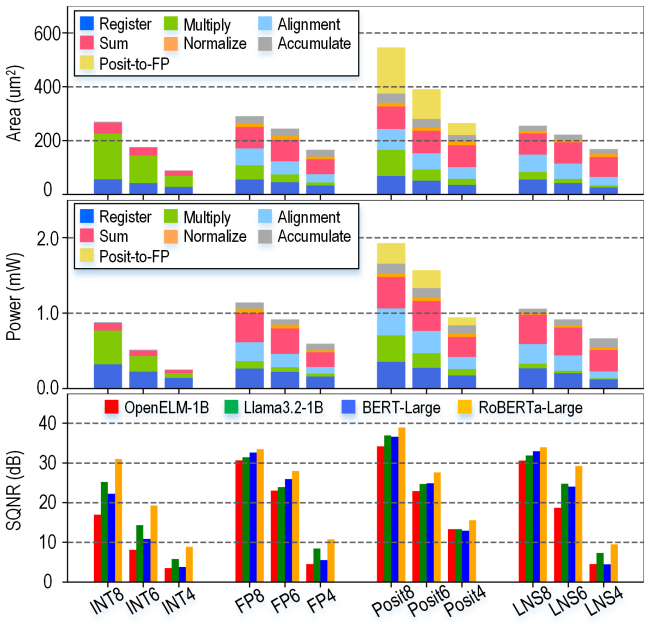


Fig. 4. Area, power, and SQNR of 8-bit, 6-bit, and 4-bit MAC designs for integer, floating-point, posit, and LNS.

following sections, we show how changing design parameters will cause large variations in area and power. ② Posit (E1) consumes nearly 2× more area and power than the other data formats because of the FP conversion overhead. However, it also achieves the best SQNR because of the higher dynamic range. ③ Multiplication makes up a significant fraction of the INT design, but FP (E4M3) only needs a 4-bit multiplier and the base-2 exponent logic in 4/6/8-bit LNS is negligible. ④ A single pipeline stage (register component in Fig. 4) consumes significant area and power overhead for 4/6/8-bit data formats. ⑤ On average the complex data formats (FP, posit, LNS) achieve higher SQNR than INT because they have higher dynamic range which is necessary for the considered transformer models.

D. MAC Design Optimization

Inner Product Size: In section II, we discussed how the inner product size (number of input terms) of the fused dot product can have a significant impact on area and energy efficiency. To understand how each data format benefits from a larger inner product size, we sweep the inner product size of a nominal MAC design and evaluate the energy efficiency (TOPs/W) in Fig. 5. First, we find that increasing the inner product size of the MAC improves the energy efficiency of all data formats. To better understand this observation, we plot the normalized per-MAC area breakdown for the FP8 MAC in Fig. 6. The improvement comes from amortizing the cost of normalization, accumulation, and registers after accumulation. However, the improvement saturates when normalization and accumulation become

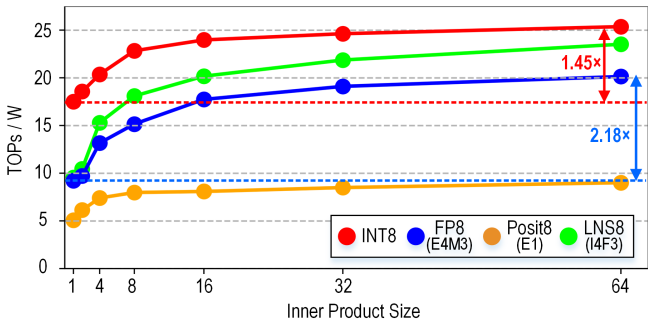


Fig. 5. TOPs/W versus the inner product size for 8-bit data formats.

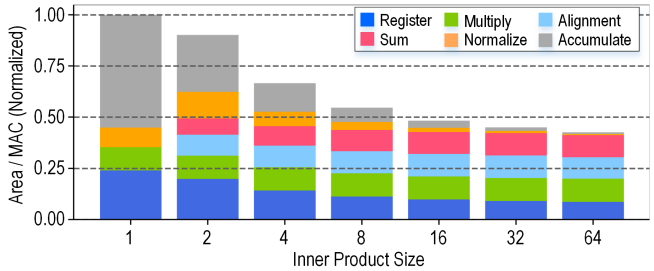


Fig. 6. Normalized area breakdown for the FP8 (E4M3) fused dot product as a function of inner product size.

a negligible fraction of the MAC area. Second, we find that the complex data formats (FP8, LNS8, posit8) benefit more from the increased number of input terms than INT8 because normalization is not performed in the integer dot product and accumulation is significantly cheaper (FP32 vs INT24 addition). Lastly, it should be noted that although peak TOPs/W can be improved by increasing the inner product size, it does not necessarily mean application performance will improve because under-utilization is more likely to occur depending on the specific layer size.

Outer Product Size: In section II, we identified how several commercial designs have extended the dot product instruction (vector-vector multiplication) to both vector-matrix and matrix-matrix multiplication. While this is done primarily to improve instruction-level parallelism, it also has logic-level implications for area and energy efficiency. To understand how each data format benefits from performing matrix-matrix multiplication (as opposed to vector-vector multiplication), we synthesize a 16-input dot product with outer product shapes of 1×1, 2×2, and 4×4 and evaluate the area efficiency (TOPs/mm^2) in Fig. 7. First, we find that increasing the outer product size for FP8, INT8, and LNS8 gives a modest improvement of ∼11% because only the input registers and pre-processing logic (sub-normal, zero, infinity, NaN) are amortized. Second, we find that posit improves visibly more than the other formats, because it's costly decoding logic can be shared across outer products. Posit8 achieves 1.35× TOPs/mm^2 for an outer product of 4×4 versus 1×1.

Accumulation Width: In section II, we identified how floating-point dot products implemented in several AI accelerators truncate the aligned products prior to addition in order to improve area and energy efficiency. To understand how each data format benefits from a reduced accumulation bit-width, we sweep the accumulation width of a nominal MAC design and evaluate the energy efficiency (TOPs/W) in Fig. 8. Because reducing the accumulation bit-width negatively impacts numerical accuracy, we also evaluate SQNR in Fig. 8. First, we find that FP8 and LNS8 achieve similar energy efficiency improvement before SQNR begins to reduce. Second, we find that INT8 is more sensitive to reducing the accumulation bit-width because it is fixed-point (not floating-point). For this reason, the accumulation bit-width of integer is rarely (if ever) truncated, but it is included here for completeness.

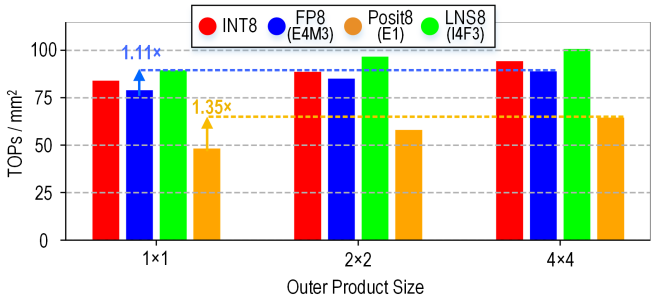


Fig. 7. TOPs/mm^2 versus the outer-product size for 8-bit data formats.

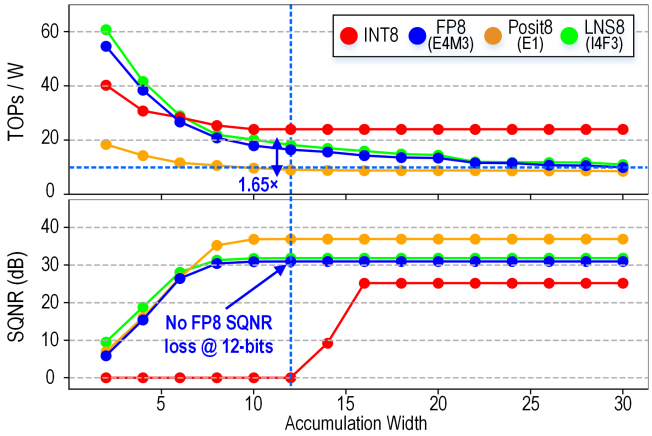


Fig. 8. TOPs/W versus the accumulation width for 8-bit data formats.

Block Data Formats: In section II, we identified an industry trend towards block data formats for effective low-precision quantization. To understand how MX and VSQ can impact the tradeoff space between area efficiency and numerical accuracy, we design and simulate their pairings with the integer, floating-point, posit, and LNS data formats. In Fig. 9, we compare a design without block scaling versus designs with a block size of 32 and 64. First, we find that block scaling yields a significant improvement in SQNR for INT4, FP4, and LNS4 while the area overhead is negligible. However, because posit4 has a large dynamic range it may not need block scaling even at 4 bits. Second, we find that decreasing the block size (64→32) improves SQNR and reduces area efficiency marginally. Third, we observe that VSQ has slightly higher overhead than MX and yields slightly higher SQNR. Because VSQ sets the scale more precisely, it achieves ∼3dB higher SQNR corresponding to $\frac{1}{2}$ bit precision.

Fixed-Point Accumulation: In section II, we described how a fused dot product aligns all mantissa products to the maximum exponent prior to accumulation. Under this topology, the products of floating-point multiplication are accumulated in floating-point; they are represented using the maximum exponent and N mantissa products, where N is the inner product size. If instead, we were to represent the products of floating-point multiplication as fixed-point numbers it would require >90 bits in FP16 [5]. However, for FP4, the products of floating-point multiplication can be represented in fixed-point using only 8-bits, and the cost of global alignment outweighs the benefits of truncation. In Fig. 10, we plot energy efficiency for both floating-point and fixed-point accumulation on FP16, FP12, FP8, FP6, FP4, and FP3. We find that floating-point accumulation is significantly better for FP16, FP12, and FP8. However, we find

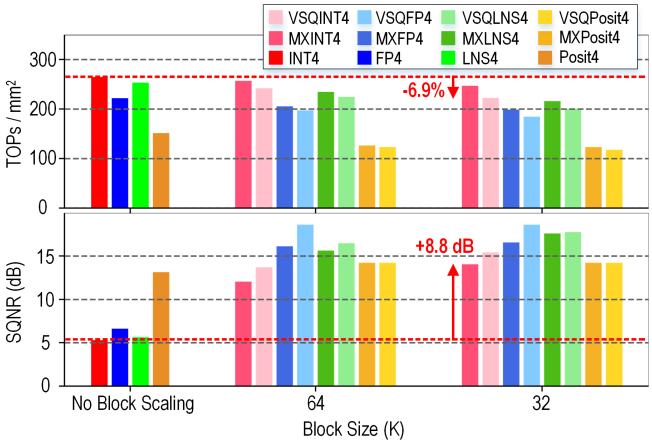


Fig. 9. TOPs/mm^2 versus block size for MX & VSQ 4-bit data formats.

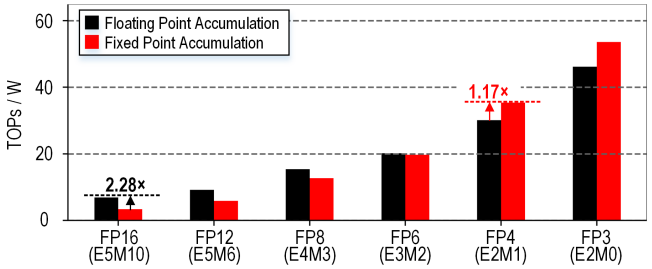


Fig. 10. TOPs/W versus floating-point and fixed point accumulation.

that FP4 with fixed-point accumulation achieves 17% better energy efficiency than the floating-point accumulation implementation.

Pipeline Stages: The last design parameter we evaluate is the number (and location) of pipeline stages. In our floating-point Verilog model we have 6 pipeline stages that can be enabled or disabled giving a total of 64 (2^6) permutations. For each design, we find the subset of pipeline configurations that yields the best area and energy efficiency. Ultimately, we find that the optimal pipeline configuration changes based on the data format and other design parameters (inner product size, accumulation width, etc).

IV. Design Space Exploration

A. Experimental Results

In Fig. 11, we plot the numerical accuracy versus energy efficiency design space for INT, FP, LNS, posit and the MX and VSQ variants of INT and FP. For each data format, we evaluate MAC implementations for 4-bit, 6-bit, 8-bit, 12-bit, and 16-bit variants. We evaluate several configurations of the FP (ex. E5M2, E4M3), LNS (ex. I5F2, I4F3), and posit (ex. E0, E1) data formats at each bit-width. We sweep a wide range of the MAC design parameters presented in section III, resulting in over 25,000 total designs. We color only the pareto optimal designs for each data format, and the gray points represent the sub-optimal designs. It should be noted that we do not consider the area and power overhead of quantization in our evaluation.

In Fig. 11, we inspect three sets of data points corresponding to three different levels of SQNR. The first set (A1, B1, C1) are baseline 1-input MAC designs that resemble the implementation of the fused-multiply add (FMA) instruction in a CPU or NVIDIA CUDA cores. The second set (A2, B2, C2) are fused dot product designs with an outer product size of 4×8 to model the Tensor Core in NVIDIA's H100 [3]. The inner product size increases from 16 (FP16) to 32 (FP8) to 64 (FP4) to model the Tensor Core in NVIDIA's H100 [3]. The third set (A3, B3, C3) are the pareto optimal designs resulting from our design space evaluation. It should be noted we limit the inner product size to 64 and outer product size to 4×4 (except for A2, B2, C2) in our evaluation to constrain the design space. It should also be noted that not all pareto optimal design points may be suitable for commercial implementation due to other factors. Instead, this design space exploration is intended to show the current upper bound that can be achieved with new data formats and custom MAC design. Starting with the 16-bit (∼70dB) data formats, we find that the FP16 dot product design (A2) achieves 2.0× TOPs/mm^2 and 1.9× TOPs/W over the baseline 1-input MAC (A1). The pareto-optimal 16-bit LNS data point (A3) achieves 1.8× TOPs/mm^2 and 1.8× TOPs/W over the FP16 dot product (A2) at nearly the same SQNR. These improvements come from the larger inner product size, smaller sum (accumulation) width, and area savings LNS obtains by avoiding multiplication. Next, we evaluate 8-bit (∼30dB) data formats. We find that the pareto-optimal MXINT8 data point (B3) can achieve 1.8× TOPs/mm^2 and 2.2× TOPs/W over the FP8 dot product (B2) while achieving higher SQNR. Lastly, we evaluate 4-bit (∼15dB)

ID	MAC Design						Figure of Merit			
	Data Format	Inner Product	Outer Product	Sum Width	# Pipeline	Block Size	#Bits	TOPs/W	TOPs/mm^2	SQNR (dB)
A1	FP16	1	-	37	2	-	16	3.1	22.0	70.8
A2	FP16	16	4×8	24	2	-	16	5.8	43.9	70.8
A3	LNS16	64	4×4	16	2	-	16	**10.3**	**78.1**	70.2
B1	FP8 (E4M3)	1	-	37	2	-	8	7.1	43.7	30.3
B2	FP8 (E4M3)	32	4×8	24	2	-	8	15.6	90.8	30.3
B3	MXINT8	64	4×4	16	1	64	8.125	**33.8**	**164.7**	**34.2**
C1	FP4 (E2M1)	1	-	37	2	-	4	7.9	59.1	4.6
C2	FP4 (E2M1)	64	4×8	24	2	-	4	43.6	238.7	4.6
C3	VSQINT4	64	4×4	8	1	32	4.25	**82.2**	**413.9**	**14.6**

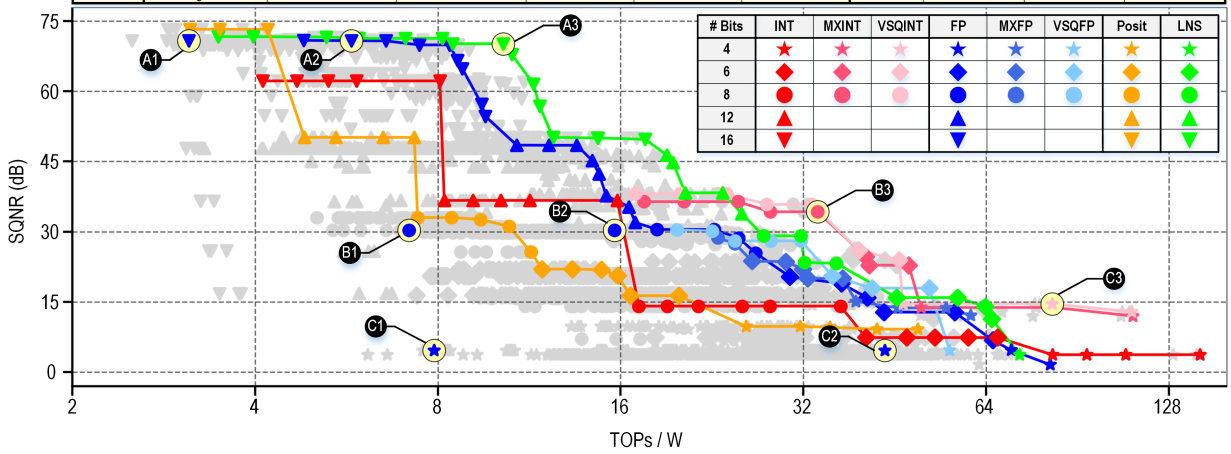


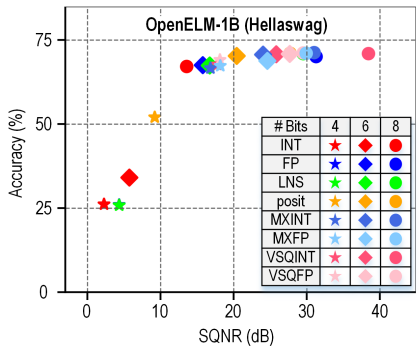
Fig. 11. Energy efficiency versus numerical accuracy design space for INT, MXINT, VSQINT, FP, MXFP, VSQFP, posit, and LNS.

data formats. We find that the pareto-optimal VSQINT4 data point (C3) can achieve 1.7× TOPs/mm^2 and 1.9× TOPs/W over the FP4 dot product (C2) while achieving higher SQNR.

From Fig. 11 we can also make several other observations. ① The pareto optimal data format for >8-bit is LNS. LNS has a marginal advantage over FP is because it avoids multiplication which is a large fraction of the energy consumed by FP when the number of mantissa bits is large (FP16, FP12). ② The pareto optimal data formats for ≤8-bit are MXINT and VSQINT. Both MX and VSQ boost the accuracy of low precision data formats at low hardware overhead. ③ Posit appears sub-optimal despite having a high SQNR compared to data formats with a similar number of bits (ex. posit8 vs FP8). This is because posit requires hardware overhead for decoding and dynamic field sizing. ④ INT has sub-optimal SQNR because it has insufficient dynamic range for the data distribution in the LLMs. This challenge is resolved by pairing INT with MX or VSQ. ⑤ At various points in the scatter plot, we see designs that improve in TOPs/W with no cost to SQNR. This observation occurs because the inner product size, outer product size, and pipeline configuration can improve TOPs/W at no impact to SQNR. ⑥ For each data format there are *steps* at various points in the pareto curve. These steps correspond to transitions from a higher precision format to a lower precision format (ex. INT12→INT8) or changes in parameters like the accumulation width and block size that trade TOPs/W for SQNR.

B. Application Benchmarks

To correlate the numerical accuracy with application accuracy, we benchmark on four representative LLMs. We evaluate two decoder-only transformer models, OpenELM-1B [28] and Llama-3.2-1B [38], and two encoder-only transformer models, BERT-Large [11] and RoBERTa-Large [25]. For OpenELM-1B and Llama-3.2-1B, we run inference on the HellaSwag dataset [40] where the model selects one of four answers. For BERT-Large and RoBERTa-Large, we run inference on the STSB dataset [6] where the model predicts a similarity score between 0 and 1. To evaluate the different data formats on each model, we implement custom PyTorch kernels to quantize FP32 weights and activations of each matrix multiplication operation to the target data format. The actual matrix multiplication executes in FP32, but the underlying weights and activations are constrained to the target data format such that it emulates custom hardware. Fig. 12 presents the accuracy achieved across our set of transformer benchmarks on the target data formats at 4-bit, 6-bit, and 8-bit precision. The results are green if <1% accuracy loss is achieved from the FP32 baseline, otherwise red. We observe a positive trend towards MX and VSQ, similar to results in Fig. 9 and Fig. 11. In Fig. 12, we also plot SQNR versus accuracy for OpenELM-1B on Hellaswag and empirically confirm a mostly monotonic (and non-linear) relationship between SQNR and application-level accuracy. A similar relationship occurs for the other benchmarks.



Task	Language Decoder						Language Encoder					
Dataset	Hellaswag						STSB					
Model	OpenELM-1B			Llama3.2-1B			BERT-Large			RoBERTa-Large		
FP32 Baseline	71.26			63.66			90.22			90.57		
Data Format	4-bit	6-bit	8-bit	4-bit	6-bit	8-bit	4-bit	6-bit	8-bit	4-bit	6-bit	8-bit
INT	26.19	34.14	67.31	25.27	26.41	27.74	0.00	0.00	67.72	0.00	52.18	90.38
FP	25.95	67.55	70.14	26.10	39.22	41.21	0.00	80.87	89.72	1.46	90.29	90.55
Posit	52.12	70.37	71.10	27.63	60.51	63.58	39.77	89.74	90.25	37.08	86.99	88.18
LNS	25.89	67.49	70.97	26.28	34.50	44.06	3.39	82.28	90.17	0.00	90.47	90.55
$MXINT_{K=32}$	67.77	70.80	71.06	49.74	63.50	63.56	88.09	89.98	90.17	88.80	90.35	90.60
$MXFP_{K=32}$	66.65	70.69	71.26	45.14	62.38	63.12	89.28	90.15	90.33	86.90	90.45	90.49
$MXLNS_{K=32}$	63.31	70.84	71.10	36.60	62.55	63.54	88.86	90.02	90.06	81.19	90.38	90.61
$VSQINT_{K=32}$	69.12	70.88	71.22	56.73	63.30	63.63	88.97	90.06	90.19	90.01	90.57	90.59
$VSQFP_{K=32}$	67.37	68.89	71.14	57.85	62.67	63.77	89.34	90.19	90.22	90.12	90.44	90.63

Fig. 12. Accuracy results (%) for target data formats on LLM benchmarks (OpenELM-1B, Llama3.2-1B, BERT-Large, and RoBERTa-Large)

REFERENCES

[1] (2017) Nvidia tesla v100 gpu architecture. [Online]. Available: http://tinyurl.com/volta-architecture-whitepaper

[2] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[3] (2022) Nvidia h100 tensor core gpu architecture. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

[4] H. Abdelaziz, J. H. Shin, A. Pedram, J. Hassoun *et al.*, "Rethinking floating point overheads for mixed precision dnn accelerators," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 223–239, 2021.

[5] N. Brunie, "Modified fused multiply and add for exact low precision product accumulation," in *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*, 2017, pp. 106–113.

[6] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics, 2017. [Online]. Available: http://dx.doi.org/10.18653/v1/S17-2001

[7] S. Dai, R. Venkatesan, M. Ren, B. Zimmer, W. Dally, and B. Khailany, "Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 873–884, 2021.

[8] B. Darvish Rouhani *et al.*, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," *Advances in neural information processing systems*, vol. 33, pp. 10 271–10 281, 2020.

[9] B. Darvish Rouhani *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[10] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3.int8(): 8-bit matrix multiplication for transformers at scale," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 30 318–30 332.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[12] M. Drumond, T. Lin, M. Jaggi, and B. Falsafi, "Training dnns with hybrid block floating point," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[13] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2023. [Online]. Available: https://arxiv.org/abs/2210.17323

[14] S. Galal and M. Horowitz, "Energy-efficient floating-point unit design," *IEEE Transactions on computers*, vol. 60, no. 7, pp. 913–922, 2010.

[15] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing frontiers and innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[16] B. Hickmann and D. Bradford, "Experimental analysis of matrix multiplication functional units," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 116–119.

[17] B. Hickmann, J. Chen, M. Rotzin, A. Yang, M. Urbanski, and S. Avancha, "Intel nervana neural network processor-t (nnp-t) fused floating point many-term dot product," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2020, pp. 133–136.

[18] N.-D. Ho, G. Jeong, C.-M. Kang, S. Choi, and I.-J. Chang, "Mersit: A hardware-efficient 8-bit data format with enhanced post-training quantization dnn accuracy," in *Proceedings of the 61th ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[19] J. Johnson, "Efficient, arbitrarily high precision hardware logarithmic arithmetic for linear algebra," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2020, pp. 25–32.

[20] N. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[21] H. Kaul, M. Anders, S. Mathew, S. Kim, and R. Krishnamurthy, "Optimized fused floating-point many-term dot-product hardware for machine learning accelerators," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 84–87.

[22] B. Keller, R. Venkatesan, S. Dai, S. G. Tell, B. Zimmer, C. Sakr, W. J. Dally, C. T. Gray, and B. Khailany, "A 95.6-tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1129–1141, 2023.

[23] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[24] Q. Li, C. Fang, and Z. Wang, "Pdpu: An open-source posit dot-product unit for deep learning applications," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[26] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra, "Llm-qat: Data-free quantization aware training for large language models," 2023. [Online]. Available: https://arxiv.org/abs/2305.17888

[27] D. R. Lutz *et al.*, "Fused fp8 4-way dot product with scaling and fp32 accumulation," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. IEEE, 2024, pp. 40–47.

[28] S. Mehta, M. H. Sekhavat, Q. Cao, M. Horton, Y. Jin, C. Sun, I. Mirzadeh, M. Najibi, D. Belenko, P. Zatloukal, and M. Rastegari, "Openelm: An efficient language model family with open training and inference framework," 2024. [Online]. Available: https://arxiv.org/abs/2404.14619

[29] D. Miyashita, E. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation. arxiv 2016," *arXiv preprint arXiv:1603.01025*.

[30] R. Murillo, D. Mallasén, A. A. Del Barrio, and G. Botella, "Energy-efficient mac units for fused posit arithmetic," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 2021, pp. 138–145.

[31] J.-S. Park *et al.*, "A multi-mode 8k-mac hw-utilization-aware neural processing unit with a unified multi-precision datapath in 4-nm flagship mobile soc," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 189–202, 2022.

[32] B. D. Rouhani, N. Garegrat, T. Savell, A. More, K.-N. Han, M. Zhao, R. Hall, J. Klar, E. Chung, Y. Yu, M. Schulte, R. Wittig, I. Bratt, N. Stephens, J. Milanovic, J. Brothers, P. Dubey, M. Cornea, A. Heinecke, A. Rodriguez, M. Langhammer, S. Deng, M. Naumov, P. Micikevicius, M. Siu, , and C. Verrilli, "Ocp microscaling (mx) specification," *Open Compute Project*, 2023.

[33] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf *et al.*, "Microscaling data formats for deep learning," *arXiv preprint arXiv:2310.10537*, 2023.

[34] J. Sohn and E. E. Swartzlander, "A fused floating-point four-term dot product unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 3, pp. 370–378, 2016.

[35] E. E. Swartzlander and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1238–1242, 1975.

[36] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei, "Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[37] A. Tirumala and R. Wong, " NVIDIA Blackwell Platform: Advancing Generative AI and Accelerated Computing ," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2024, pp. 1–33. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HCS61935.2024.10665247

[38] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[39] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[40] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[41] J. Zhao, S. Dai, R. Venkatesan, B. Zimmer, M. Ali, M.-Y. Liu, B. Khailany, W. J. Dally, and A. Anandkumar, "Lns-madam: Low-precision training in logarithmic number system using multiplicative weight update," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3179–3190, 2022.

Property-driven Parallel Symbolic Model Checking of LTL

Yuheng Su[†‡], Yingcheng Li[†‡], Qiusong Yang[‡§], Yiwei Ci[‡], Ziyu Huang[¶]
[†]University of Chinese Academy of Sciences
[‡]Institute of Software, Chinese Academy of Sciences
[§]Laboratory for Advanced Computing and Intelligence Engineering, Wuxi, China
[¶]Beijing Forestry University

***Abstract*—Model checking is an automated method used to formally verify systems by checking them against properties. However, a major problem in model checking is the state explosion. To overcome this challenge, one approach is to utilize parallel processing capabilities to either speed up computations or handle larger-scale problems. Explicit model checking has lower computational complexity and can be easily parallelized. There are numerous parallel explicit model checking algorithms available in the literature. Symbolic model checking offers significant advantages over explicit model checking in terms of problem scalability and verification speed. However, treating states encountered during the search as sets poses a challenge in devising efficient parallel algorithms. As a result, current research on parallelizing symbolic model checking has primarily focused on reachability analysis or safety properties, rather than attempting to parallelize the nested fixpoint calculations. In this paper, we propose a novel property-driven approach for parallel symbolic model checking of full LTL. Our algorithm introduces a fair model state labelling function that forms a partition of the nested fixpoint across the product combining the model and the property Büchi automaton. The experimental results demonstrate significant speedup, ranging from 2.81 to 17.19 times compared to sequential approaches on a 32-core machine. Moreover, in comparison to existing parallel model checking methods, our approach not only surpasses those relying on BDD libraries with a maximum improvement of up to 134% and an average improvement of 33.1% but also demonstrates significant superiority over the state-of-the-art parallel explicit model checker.**

I. Introduction

Model checking is an automated method for formally verifying systems against temporal properties [1]. However, the state space must be exhaustively explored either explicitly or symbolically, leading to the state explosion problem that has plagued model checking since its inception [2]. To address this issue, researchers have developed clever methods such as abstractions [3] and symbolic representations [4]. Another approach is to leverage the rapid development of computing, storage, and communication technologies by parallelizing sequential model checking using multi-core [5], distributed systems [6], GPGPU [7], and other modes of computation.

Explicit model checking is amenable to parallelization, and extensive parallel algorithms exist in the literature. In explicit model checking, each state encountered during BFS, DFS, or some variants is explicitly stored and individually manipulated. Explicit model checking algorithms based on BFS or the detection of strongly connected components (SCC) belong to the Nick class [8], which theoretically have highly parallel algorithms [9]. Although DFS is inherently sequential, and thus the classic nested DFS model checking algorithm [10] based on finding an accepting and fair cycle does not belong to the Nick class, there are also many efficient parallel explicit model checking algorithms based on DFS [8].

Symbolic model checking offers significant advantages, compared to explicit model checking, in terms of the scale of problems and verification speed. Symbolic model checking uses characteristic functions to represent a set of states. Symbolic representations, such as BDDs [11], are normally more compact than explicit representations, making it possible to verify problems with larger scales. Modern symbolic model checking algorithms are variants of the Emerson-Lei algorithm [12], which includes nested fixpoint calculations.

However, treating states encountered as sets poses a challenge in devising efficient parallel symbolic algorithms. The most challenging task lies in devising an efficient algorithm for computing the nested fixpoint when the symbolic state space is divided into slices. As a result, current research on parallelizing symbolic model checking has primarily focused on reachability analysis or safety properties, avoiding the parallelization of nested fixpoint calculations. Building upon distributedly partitioned BDDs, a parallel symbolic model checking algorithm for safety properties was introduced in [13]. There have been also various studies on parallel symbolic model checking of systems with specific structures. For instance, research has been conducted on the reachability analysis of security protocols with a bounded number of sessions [14] and the verification of split-invariants for asynchronously composed processes communicating through shared-memory [15]. Unlike these studies, which do not focus on full model checking, [16] proposed a symbolic algorithm for the model checking of propositional μ-calculus based on distributively evaluating subformulas. However, formulas to be verified are generally not large in scale, the degree of parallelism is limited.

In this paper, we present property-driven parallel symbolic model checking of full LTL. The contributions is summarized as follows:

- We propose a property-driven state space partition method for nested fixpoint computation. This algorithm utilizes a fair model state labelling function that ensures the labeling of a model state s in an automoton state q if and only if there exists a fair path starting from (s, q) in the product. The fair model state labelling function naturally forms a partition of the nested fixpoint. Therefore, by computing the labelling function, we effectively implement the symbolic model checking of LTL.
- We propose a parallelize algorithm to compute the state labelling function. This algorithm takes advantage of the labelling function can be computed using a backward BFS manner, which is well-suited for parallelization. In our approach, the computation is performed by a group workers. The number of workers used in this process is equal to the number of states in the automaton, denoted as n. However, we have also devised mechanisms to increase the degree of parallelization beyond n, allowing for more workers to be utilized. To ensure efficient usage of resources, we have also introduced load balancing techniques.
- We conduct a comprehensive evaluation of our parallel algorithm and compared it to existing approaches. The results highlight the effectiveness and efficiency of our proposed method. The experimental results on a 32-core machine demonstrate significant speedup, ranging from 2.81 to 17.19 times compared to sequential approaches. Moreover, in comparison to existing parallel methods, our approach not only surpasses those relying on BDD libraries with a maximum improvement of up to 134%

* Qiusong Yang is the corresponding author.

and an average of 33.1% but also demonstrates superiority over the parallel explicit model checker LTSMin [5].

II. Preliminaries

Kripke structure: Without loss of generality, we assume the system is modelled as Kripke structure. Specifically, a Kripke structure is represented by a tuple $M = (S, S_0, R, AP, L)$ in which S denotes the set of states, $S_0 \subseteq S$ represents the set of initial states, $R \subseteq S \times S$ denotes the transition relation, AP is the set of atomic propositions, and $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions that are true in that state [1]. A path of length l is defined as a sequence of states $s_0, \cdots, s_{l+1}$ where $s_0 \in S_0$ and for all $0 \leq i \leq l$, $(s_i, s_{i+1}) \in R$.

Linear temporal logic (LTL) is a formalism for specifying temporal properties derived from Kripke structures. It extends propositional logic with temporal operators such as $\mathbf{X}$ (next), $\mathbf{F}$ (eventually), $\mathbf{G}$ (globally), and $\mathbf{U}$ (until), allowing the expression of time-dependent behaviors [1]. The properties expressed in LTL can be classified into safety and liveness.

Büchi automaton: LTL model checking typically involves negating the formula, converting it to a Büchi automaton, and checking the emptiness of its product with the Kripke model. As the Büchi automaton generated from an LTL formula normally has an alphabet over the power set of AP and includes atomic propositions as labels in each state, in addition to the standard definition, we have incorporated a state labeling function in the following definition. A Büchi automaton is defined as $\mathcal{B} = (\Sigma, Q, \Delta, Q_0, \mathcal{L}, F)$, where $\Sigma = 2^{AP}$ is the finite alphabet, Q is the finite set of states, $\mathcal{L} : Q \to 2^{AP}$ is the state labelling function, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation such that if $(q, \sigma, q') \in \Delta$ then $\sigma = \mathcal{L}(q)$, Q_0 is the set of initial state, and $F = \{P_1, P_2, \cdots, P_k\}$ where for every $1 \leq i \leq k$, $P_i \subseteq Q$. Similarly, an infinite path ρ is a sequence states $q_0, \cdots, q_i, \cdots$ such that $q_0 \in Q_0$ and, for all $i \geq 0$, $(q_i, \nu(i), q_{i+1}) \in \Delta$, where $\nu(i)$ denotes the i-th input letter of the word sequence $\nu \in \Sigma^*$. A Büchi automaton accepts only infinite ***illustrate*** paths in which there exist at least one state from each P_i that appears infinitely often. Given a Kripke structure $M = (S, S_0, R, AP, L)$ and a Büchi automaton $\mathcal{B} = (2^{AP}, Q, \Delta, Q_0, \mathcal{L}, F)$, their synchronized product is denoted as $M \otimes \mathcal{B} = (2^{AP}, Q_\otimes, \Delta_\otimes, Q_{\otimes 0}, F_\otimes)$. Here, $Q_\otimes = S \times Q$, $Q_{\otimes 0} = S_0 \times Q_0$, and $((s_1, q_1), \sigma, (s_2, q_2)) \in \Delta_\otimes$ if and only if $(s_1, s_2) \in R$, $L(s_1) = \mathcal{L}(q_1)$, and $(q_1, \sigma, q_2) \in \Delta$. Furthermore, $F_\otimes(s_1, q_1) = F(q_1)$.

III. Problem Formulation

The formula $\mathbf{EG}\varphi$ expresses the existence of a path in which φ holds globally. When fairness is considered, the set of states satisfying $\mathbf{EG}\varphi$ can be obtained through the following nested fixpoint:

$$[\mathbf{EG}\varphi]_M^{Fair} = \nu Z.\varphi \wedge \bigwedge_{k=1}^{n} \mathbf{EXE}(\varphi \mathbf{U}(Z \wedge P_k)) \tag{1}$$

Here, ν represents the greatest fixpoint, and $\{P_1, \cdots, P_n\}$ represents the fairness. If any of the initial states of M belongs to $[\![\mathbf{EG}\varphi]\!]_M^{Fair}$, then $M \models \mathbf{EG}\varphi$. Consequently, the formula $\mathbf{EG}true$ characterizes the set of states from each of which there exists a fair path.

The symbolic model checking of LTL based on the computation of $\mathbf{EG}true$, assuming $F = P_1$ without loss of generality, is presented in Algorithm 1. When the fixpoint is reached, only the states from which there exists a fair path are kept. The procedure above heavily relies on the use of symbolic state representation, utilizing sets of states in every iteration. This approach significantly enhances its effectiveness and clarity. However, it also makes decision procedures employed in explicit parallel model checking, such as loop detection based on nested DFS [10], or loop detection based on BFS [17], impractical.

Algorithm 1: Symbolic algorithm for LTL model checking

Input: a Kripke structure M, a Büchi $\mathcal{B}_{\neg\varphi}$ automaton corresponding to the LTL formula to be verified.
Output: a boolean result indicating whether the language accepted by $M \otimes \mathcal{B}_{\neg\varphi}$ is empty.

```
1  begin  /* Compute EGtrue = νZ.EXEF(Z ∧ F⊗) */
2      S1 ← S × Q, Q' ← S1;
3      do
4          Q ← Q';
5          begin /* EF(Q ∧ F⊗) = μZ.(Q ∧ F⊗) ∨ EXZ */
6              W' ← ∅;
7              do
8                  W ← W';
9                  W' ← (Q ∧ F⊗) ∨ Pre⊗(W);
10             while W ≠ W';
11         begin                          /* Compute EX */
12             Q' ← Pre⊗(w);
13     while Q ≠ Q';
14 begin                                  /* Check emptiness */
15     if ∃(s0, q0) ∈ S0 ⊗ Q0 such that (s0, q0) ∈ Q then
16         return false
17     else
18         return true
```

As discussed previously, the current research on parallelizing symbolic model checking primarily focuses on reachability analysis and safety properties, while neglecting the parallelization of nested fixpoint calculations. Therefore, there is a gap in this research area as no investigations have been conducted on parallelizing the nested fixpoint calculation process. Our paper aims to present an effective approach for parallelizing the nested fixpoint computation process, enabling parallel symbolic model checking of full LTL.

IV. Property-driven Parallel Symbolic Model Checking

The term *property-driven* refers to the transformation of nested fixpoint computation into the parallel computation of n smaller fixpoints, where n is the number of states of the property automaton.

A. Sequential algorithm

Given a Kripke structure $M = (S, S_0, R, AP, L)$, a Büchi automaton $\mathcal{B} = (2^{AP}, Q, \Delta, Q_0, \mathcal{L}, F)$ and their synchronous product, we have the following definitions:

Definition IV.1 (Model state labelling function)**.** *Let $\mathcal{L}_\mathcal{M}$ be a function: $Q \to 2^S$. It should be noted that any subset of S can be characterized by a predicate over AP.*

Definition IV.2 (Projection)**.** *For a subset $Q' \subseteq Q$ and a model state labelling function $\mathcal{L}_\mathcal{M}$, the projection of $\mathcal{L}_\mathcal{M}$ over Q' is denoted as $\mathcal{L}_{\mathcal{M}\restriction Q'}(q) = \mathcal{L}_\mathcal{M}(q)$ if $q \in Q'$, and $\emptyset$ otherwise.*

Definition IV.3 (Fair model state labelling function)**.** *A model state labelling function $\mathcal{L}_\mathcal{M}$ is considered fair if and only if for any state s in the set $\mathcal{L}_\mathcal{M}(q)$, there exists a fair path starting from (s, q) in the product.*

Theorem IV.1. *For a state labeling function, the set $\{\mathcal{L}_\mathcal{M}(q) |$ for all $q \in Q\}$ forms a partition of the nested fixpoint $\mathbf{EG}true$.*

Algorithm 2: Sequential property-driven symbolic algorithm

Input: a Kripke structure M; a Büchi $\mathcal{B}_{\neg\varphi}$ automaton corresponding to the LTL formula to be verified.
Output: a boolean result indicating whether the language accepted by $M \otimes \mathcal{B}_{\neg\varphi}$ is empty.

/* Compute fair $\mathcal{L}_\mathcal{M}$ */
1 $\mathcal{L}^0_\mathcal{M}(q) \leftarrow true,\ for\ all\ q \in Q$;
2 **do**
3 $\quad \mathcal{L}^1_\mathcal{M} \leftarrow \mathcal{L}^0_\mathcal{M}$;
4 $\quad \mathcal{L}^0_\mathcal{M}(q) \leftarrow \mathcal{L}(q) \cap \mathcal{L}^1_\mathcal{M}(q)$ if $q \in P_1$, $\emptyset$ otherwise;
/* $s \in \mathcal{L}^0_\mathcal{M}(q)$ *if and only if* $(s,q) \in$ **EF**$(\mathcal{L}(q) \cap \mathcal{L}^1_\mathcal{M}(q))$ */
5 $\quad FIFO\ queue \leftarrow P_1$;
6 $\quad$ **while** $queue \neq \emptyset$ **do**
7 $\quad\quad q \leftarrow queue.remove$;
8 $\quad\quad$ **for** *each* $q' \in Pre_\mathcal{B}(q)$ **do**
9 $\quad\quad\quad update \leftarrow Pre_M(\mathcal{L}^0_\mathcal{M}(q)) \cap \mathcal{L}(q') \cap \overline{\mathcal{L}^0_\mathcal{M}(q')}$;
10 $\quad\quad\quad$ **if** $update \neq \emptyset$ **then**
11 $\quad\quad\quad\quad \mathcal{L}^0_\mathcal{M}(q') = \mathcal{L}^0_\mathcal{M}(q') \cup update$;
12 $\quad\quad\quad\quad$ *add* q' *to queue*;
/* *Compute* **EX**$(\mathcal{L}^0_\mathcal{M})$ */
13 $\quad \mathcal{L}^{EX}_\mathcal{M}(q) \leftarrow false,\ for\ all\ q \in Q$;
14 $\quad FIFO\ queue \leftarrow Q$;
15 $\quad$ **while** $queue \neq \emptyset$ **do**
16 $\quad\quad q \leftarrow queue.remove$;
17 $\quad\quad$ **for** *each* $q' \in Pre_\mathcal{B}(q)$ **do**
18 $\quad\quad\quad \mathcal{L}^{EX}_\mathcal{M}(q') = \mathcal{L}^{EX}_\mathcal{M}(q') \cup (Pre_M(\mathcal{L}^0_\mathcal{M}(q)) \cap \mathcal{L}(q'))$;
19 $\quad \mathcal{L}^0_\mathcal{M} = \mathcal{L}^{EX}_\mathcal{M}$;
20 **while** $\mathcal{L}^1_\mathcal{M} \neq \mathcal{L}^0_\mathcal{M}$;
/* Check emptiness */
21 **if** $\exists s_0 \in S_0, q_0 \in Q_0\ such\ that\ s_0 \in \mathcal{L}^0_\mathcal{M}(q_0)$ **then**
22 $\quad$ **return** *false*
23 **else**
24 $\quad$ **return** *true*

In Algorithm 2, we propose a sequential property-driven symbolic algorithm for model checking LTL. The main objective of the algorithm is to compute the fair model state labeling function. When line 4 is first executed, each accepting state q is labeled with the model states that satisfy $\mathcal{L}(q)$. The code between line 5 and line 12 performs backward labeling on the Büchi automaton's states. This ensures that a model state s is labeled in some state q if and only if $(s,q) \in$ **EF**$(\mathcal{L}(q) \cap \mathcal{L}^1_\mathcal{M}(q))$. In line 11, the code ensures that if a model state s' is added to the labeling of q', there must exist a state $s \in \mathcal{L}^0_\mathcal{M}(q)$ such that $((s',q'),\mathcal{L}(q'),(s,q)) \in \Delta_\otimes$. The subsequent steps in lines 13 to 19 calculate the **EX** operation. This operation involves removing a state's label s from some accepting state q under the following conditions: either (s,q) is not part of any loop, or it cannot reach any other accepting states.

A Kripke structure and an automaton are presented in Figure 1. Initially, $\mathcal{L}_\mathcal{M}(d) = \mathcal{L}(d) = \{x,y,z\}$, $\mathcal{L}_\mathcal{M}(a) = \mathcal{L}_\mathcal{M}(b) = \mathcal{L}_\mathcal{M}(c) = \emptyset$, and $queue = \{d\}$. Then get d from the $queue$, and calculate the pre-image of $\mathcal{L}_\mathcal{M}(d)$, $Pre_M(\mathcal{L}_\mathcal{M}(d)) = \{x,y,z\}$. And then transfer it to the predecessor states in the automaton, $Pre_\mathcal{B}(d) = \{b,c,d\}$. For state b, calculate all states that belong to $Pre_M(\mathcal{L}_\mathcal{M}(d)) \cap \mathcal{L}(b)$ and do not belong to $\mathcal{L}_\mathcal{M}(b)$, $Pre_M(\mathcal{L}_\mathcal{M}(d)) \cap \mathcal{L}(b) \cap \overline{\mathcal{L}_\mathcal{M}(b)} = \{x,z\}$. If it is not empty, add these states to $\mathcal{L}_\mathcal{M}(b)$ and enqueue b. The same for c and d, then $\mathcal{L}_\mathcal{M}(a) = \emptyset$, $\mathcal{L}_\mathcal{M}(b) = \{x,z\}$, $\mathcal{L}_\mathcal{M}(c) = \{y,z\}$, $\mathcal{L}_\mathcal{M}(d) = \{x,y,z\}$ and $queue = \{b,c\}$. Continue this process until the $queue$ becomes empty, indicating the occurrence of the fixpoint of **EF**. Then, proceed to calculate **EX** and repeat the entire process until $\mathcal{L}_\mathcal{M}$ is no longer updated(line 30). Finally, $\mathcal{L}_\mathcal{M}(a) = \{y\}$, $\mathcal{L}_\mathcal{M}(b) = \{x,z\}$, $\mathcal{L}_\mathcal{M}(c) = \{y,z\}$, $\mathcal{L}_\mathcal{M}(d) = \{x,y,z\}$, which is the fixpoint of **EG**$true$. The property is safe because the only initial state $x \notin \mathcal{L}_\mathcal{M}(a)$.

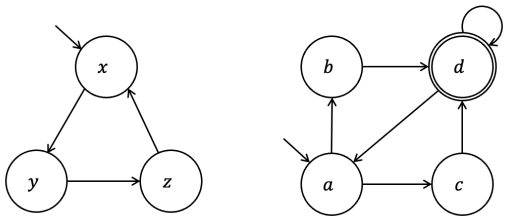


Fig. 1: A Kripke structure $S_0 = \{x\}$, and a Büchi automaton, $Q_0 = \{a\}$, $P_1 = \{d\}$, $\mathcal{L}(a) = \{y\}$, $\mathcal{L}(b) = \{x,z\}$, $\mathcal{L}(c) = \{y,z\}$, $\mathcal{L}(d) = \{x,y,z\}$

Algorithm 3: Parallel property-driven symbolic algorithm

Input: a Kripke structure M; the Büchi $\mathcal{B}_{\neg\varphi}$ automaton corresponding to $\neg\varphi$, where φ is the LTL formula to be verified. Let $n = |Q|$.
Output: a boolean result indicating whether the language accepted by $M \otimes \mathcal{B}_{\neg\varphi}$ is empty
Data: $\mathcal{L}^0_\mathcal{M}(q)$ with each element initialized to true, $\mathcal{L}^1_\mathcal{M}(q)$.

1 **do**
2 $\quad \mathcal{L}^1_\mathcal{M} = \mathcal{L}^0_\mathcal{M}$;
3 $\quad$ **for** *each* $q \in Q$ **do** /* Compute EXEF */
4 $\quad\quad Spawn\ Labelling_q(\mathcal{L}^0_{\mathcal{M}\uparrow P_1}(q))$;
5 $\quad$ **for** *each* $q \in Q$ **do**
6 $\quad\quad \mathcal{L}^0_\mathcal{M}(q) \leftarrow Sync\ Labelling_q$;
7 **while** $\mathcal{L}^0_\mathcal{M} \neq \mathcal{L}^1_\mathcal{M}$;
/* Check emptiness */)
8 **if** $\exists s_0 \in S_0, q_0 \in Q_0\ such\ that\ s_0 \in \mathcal{L}^0_\mathcal{M}(q_0)$ **then**
9 $\quad$ **return** *false*
10 **else**
11 $\quad$ **return** *true*

Theorem IV.2. *Algorithm 2 computes the fair model state labelling function.*

Proof. In the algorithm, a model state s is marked in a particular state q if and only if (s,q) is an accepting state of the product or can reach an accepting state within a finite number of steps. It is important to note that the algorithm will eventually terminate since the **EX** operations cannot run indefinitely. Upon termination, the nested fixpoint computation guarantees that s is marked in a state q if and only if there exists a fair path that starts from (s,q). □

B. Parallel algorithm

The algorithm 2 effectively labels each Büchi state in a backward BFS manner, making it well-suited for parallelization. The parallel version is presented in Algorithm 3 and Algorithm 4.

In Algorithm 3, lines 1-7 compute the **EXEF** formula in parallel, continuing in an iterative manner until a fixed point is reached. More specifically, line 4 of the code spawns a separate worker for each state in the Büchi automaton, and each worker executes Algorithm 4. The projection of $\mathcal{L}^0_\mathcal{M}$ onto P_1 in line 4 will assign labels to each

state q in P_1 based on the modeling states that satisfy q, leaving all other states labeled with the empty set. In line 6, the algorithm is blocked until all workers have completed their tasks, subsequently retrieving the results of their computations.

Algorithm 4: Labelling$_q$

Input: $\mathcal{L}^0_{\mathcal{M}}(q), \mathcal{L}(q)$.
Output: a set of states indicating fair model state labelling function over q

1 $\mathcal{L}^{EF}_{\mathcal{M}}(q) = \mathcal{L}(q) \cap \mathcal{L}^0_{\mathcal{M}}(q)$;
2 **if** $\mathcal{L}^{EF}_{\mathcal{M}}(q) \neq false$ **then**
3 **for** *each* $q' \in Pre_{\mathcal{B}}(q)$ **do**
4 *Send EF message* $\mathcal{L}^{EF}_{\mathcal{M}}(q)$ *to Labelling*$_{q'}$;
5 **do**
6 **if** *all other Labelling*$_q$ *blocked in receiving* **then**
7 *Notify other Labelling*$_q$ *stopping receiving and breaking*;
8 **Break**;
9 *Wait until an EF message arrives*;
10 *Receive a Set message ef_message*;
11 $update = Pre_M(ef_message) \cap \mathcal{L}(q) \cap \overline{\mathcal{L}^{EF}_{\mathcal{M}}(q)}$;
12 $\mathcal{L}^{EF}_{\mathcal{M}}(q) = \mathcal{L}^{EF}_{\mathcal{M}}(q) \cup update$;
13 **if** $update \neq false$ **then**
14 **for** *each* $q' \in Pre_{\mathcal{B}}(q)$ **do**
15 *Send EF message update to Labelling*$_{q'}$;
16 **while** *true*;
17 **for** *each* $q' \in Pre_{\mathcal{B}}(q)$ **do**
18 *Send EX message* $\mathcal{L}^{EF}_{\mathcal{M}}(q)$ *to Labelling*$_{q'}$;
19 $\mathcal{L}^{EXEF}_{\mathcal{M}}(q) = false$;
20 **do**
21 **if** *all other Labelling*$_q$ *blocked in receiving* **then**
22 *Notify other Labelling*$_q$ *stopping receiving and breaking*;
23 **Break**;
24 *Wait until an EX message arrives*;
25 *Receive a Set message ex_message*;
26 $\mathcal{L}^{EXEF}_{\mathcal{M}}(q) = \mathcal{L}^{EXEF}_{\mathcal{M}}(q) \cup (Pre_M(ex_message) \cap \mathcal{L}(q))$;
27 **while** *true*;
28 **Return** $\mathcal{L}^{EXEF}_{\mathcal{M}}(q)$;

In Algorithm 4, lines 5-20 collaboratively compute the **EF** with other workers. At each stage, the algorithm continuously processes incoming messages from workers associated with its successors in the Büchi property automaton, updating its own set of labeled fair states. Upon receiving a message containing the newly labeled fair states from one of the successors, the algorithm labels state q with the predecessors of those fair states in the Kripke structure, ensuring that these predecessors also satisfy the state q, as shown in lines 12-13. If the set of newly discovered states, denoted as $update$, is not empty, the algorithm sends them to its predecessors as indicated by lines 15-16 of the code. Lines 21-30 compute the **EX** in the same way. After *EX* is computed, the result is returned to Algorithm 3. Before waiting for new messages, the algorithm checks whether all other workers are also waiting for incoming messages. If so, it indicates that there are no more messages being sent, and the worker notifies others to stop receiving messages and exit.

Algorithm 5: Increase Parallelism

Input: $desired_parallelism$, $LTL_original$
Output: an automaton with desired size

1 $LTL = LTL_original$;
2 $automaton$ = $generate_automaton_from_ltl(\neg LTL)$;
3 **while** $automaton.size < desired_parallelism$ **do**
4 $R_s \leftarrow$ *select a sub relation of transition*;
5 $R_{sr} \leftarrow$ *replace primed states with* **X** *operator in* R_s;
6 $LTL_{sr} \leftarrow \bigwedge_{r \in R_{sr}} r$;
7 $LTL = \mathbf{G}(LTL_{sr}) \rightarrow LTL$;
8 $automaton$ = $generate_automaton_from_ltl(\neg LTL)$;
9 **Return** $automaton$;

C. Optimizations

1) Increase parallelism: According to Algorithm 3, the number of workers is set equal to the number of states of the automaton. However, it is worth noting that the size of the automaton exhibits exponential worst-case complexity relative to the size of the formula itself [18]. Despite this exponential complexity, it does not pose a disadvantage for our approach. Despite the exponential worst-case complexity, our experiments with the algorithms described in [18] have demonstrated that they often yield automaton that are smaller in size. Nevertheless, to overcome this limitation, we have developed a solution that involves expanding the size of the automaton beyond the number of states in the original automaton. This enhancement improves the parallelism of our algorithm. Intuitively, by augmenting the complexity of the LTL formula, we can also amplify the size of the corresponding automaton.

Usually, the transition relation R of a *Kripke structure* is represented as a conjunction of some sub-relations $R_1, ..., R_n$ such that:

$$R = \bigwedge_{i=1}^{n} R_i, \ where \ R_i \ has \ the \ form \ R_i \ \equiv v' = f(\overline{v}) \quad (2)$$

We can select a set of sub-relations, denoted as $R_s = \{R_i | R_i \ is \ selected\}$, and incorporated them into the LTL formula, which has led to an increase in its size. However, a problem arises due to the presence of primed variables v' in R_i, as these are not valid in LTL expressions. To address this issue, we need to first replace the primed variables v' with the LTL next operator $\mathbf{X}\ v$, resulting in a new set of sub-relations denoted as R_{sr}. Let $LTL_{sr} = \bigwedge_{r \in R_{sr}} r$. Consequently, the new LTL formula will be formulated as follows:

$$LTL_{new} = (\mathbf{G}\ LTL_{sr}) \rightarrow LTL_{original} \quad (3)$$

To establish the relationship between the conjunction of the selected relations and the original LTL, an implication is used. This implies that if a model satisfies the selected transition relations globally, then LTL_{new} will be equisatisfiable with $LTL_{original}$.

Algorithm 5 outlines the procedure for increasing parallelism. It begins by checking if the size of the generated automaton is smaller than the desired parallelism. If it is, another sub-relation is added to construct a new LTL formula, and this process is repeated until the size reaches or exceeds the desired value.

2) Load balancing: As described in Algorithm 2, the workers collaboratively calculate the fair model state labeling function using a backward BFS approach. Generally, parallelizing a BFS-based algorithm provides inherent load balancing advantages. However, it is inevitable that load imbalance may occur in parallel algorithms due

to various factors. For example, in our approach, if only a few threads are performing BDD computation while other threads are blocked, waiting for incoming messages, load imbalance may arise.

To achieve load balancing in a scenario where some workers are blocked, waiting for incoming messages, the work-stealing strategy [19] allows these workers to assist other workers by stealing BDD operations and adding them to their own queue. Instead of developing a proprietary system to support this work-stealing approach, we utilize Sylvan [20], which is a multi-core BDD library. In Sylvan, the work-stealing strategy is employed to facilitate the stealing of BDD operation tasks from other workers when a worker becomes blocked. By leveraging the capabilities of Sylvan, the BDD operations in lines 12-13 and 28 are inputted into the library for effective load balancing.

V. Experimental Results

A. Experiment Setup

We implemented the proposed property-driven parallel algorithm using Sylvan for BDD operations and ltl2tgba for automaton generation. The artifact is available at [21]. Experiments were conducted on a 32-core AMD EPYC 7532 machine with 128GB RAM.

The available benchmarks of model checking are mostly focused on invariant properties, whereas our approach focuses on LTL. We evaluated our method using cases from the examples provided in the NuSMV tool. All of the models and properties were represented in SMV format. The key parameters of the cases are outlined in Table III, where the column *Hold* signifies whether the model checking problem holds, *Variables* represents the count of boolean state variables, and *Assignments* indicates the number of next-state variable assignments, as illustrated in Equation (2).

B. General Observation

Baseline Table I presents the experimental results obtained from evaluating our proposed property-driven algorithm. The sequential property-driven algorithm, outlined in Algorithm 2 and denoted as PSMC, is employed as the baseline for parallel performance evaluation. To establish its comparability to state-of-the-art symbolic model checkers, the column *NuSMV* showcases the performance data of NuSMV on the cases. It becomes evident that the baseline algorithm outperforms NuSMV in nearly all cases.

Scalability The performance of the parallel version of PSMC on various numbers of cores, denoted as $PPSMC_{XX}$, is detailed in Table I. In particular, PPSMC achieves a parallel speedup ranging from 2.81 to 17.19 times compared to PSMC in a 32-core environment, with an average speedup of 8.18 times. Figure 2 further illustrates the speedup of PPSMC at various core counts on average. It is evident that as the number of cores increases, there is no significant slowdown in the upward trend. This observation validates the near-linear scalability of PPSMC, affirming its efficient handling of larger workloads.

Increase parallelism Table II presents the sizes of automaton generated using different numbers of sub-relations on specific models. As the number of sub-relations increases, more intricate LTL formulas are constructed, leading to larger automaton. This size increase is instrumental in enhancing the parallelism of our method. The results show that our method effectively improves parallelism by constructing complex automaton through sub-relation selection.

Load balancing In Table I, we also presents the runtime of PPSMC without load balancing. The result shows that load balancing leads to a significant improvement in performance, with an average improvement of 61%. This result clearly demonstrates the effectiveness of load balancing in our method.

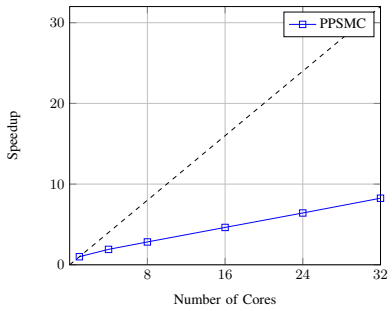


Fig. 2: PPSMC Average Speedups

TABLE II: Parallelism Increase

Model	$\lvert R_s \rvert$	0	1	2	3	4
phils p1	$\lvert Q_{\mathcal{B}} \rvert$	2	5	9	17	33
elevator	$\lvert Q_{\mathcal{B}} \rvert$	10	18	35	69	
syncarb	$\lvert Q_{\mathcal{B}} \rvert$	3	7	13	25	49
prod-cons-p0	$\lvert Q_{\mathcal{B}} \rvert$	4	19	39	71	

C. Comparison

Compare to Sylvan To our knowledge, the only existing parallel symbolic model checking approach for full LTL relies solely on parallel BDD libraries like Sylvan [20]. The advantage of this approach is that the high-level model checking algorithm remains unchanged, and all that needs to be done is to invoke the Sylvan library. The strength of our approach lies in an additional level of property-driven parallelization, which not only enhances the overall level of parallelism but also contributes to improved load balancing. We evaluated Algorithm 1 implemented by Sylvan in a 32-core environment. The results are presented in the column labeled $Sylvan_{32}$ of Table III. It is evident from the column labeled *Improv.*, when comparing the column $Sylvan_{32}$ against the column $PPSMC_{32}$, that PPSMC consistently outperforms parallel Sylvan SMC in most cases. The improvement reaches a maximum of up to 134%, with an average improvement of 33.1%.

Compare to LTSmin We also evaluated LTSmin [22], which is a state-of-the-art parallel explicit model checker. We input SMV models into LTSmin through the PINS interface since LTSmin does not support SMV frontend. The results are shown in the column $LTSmin_{32}$. It is apparent that LTSmin has encountered difficulties in solving numerous instances, highlighting the superior scalability of symbolic model checking over explicit methods.

Compare to NuXmv We conducted an evaluation of the k-liveness algorithm [23] implemented in NuXmv [24], a variant of the IC3 algorithm designed for checking full LTL properties. The experimental results are presented in Table III. While it may be unfair to directly compare PPSMC32 with NuXmv, it's noteworthy that PPSMC32 outperforms NuXmv in ten out of twenty cases. In particular, in the case of *cuhanoi10ro*, NuXmv experienced a timeout, whereas BDD-based methods successfully solved it. Despite the prevalent advantages of SAT-based methods in recent years, certain problem classes still rely on BDD-based techniques as the sole method for achieving automated solutions. BDD-based techniques are often considered as part of a diverse range of methods.

VI. Related Work

Explicit parallel model checking individually stores and processes each state encountered during the state space exploration. This approach facilitates the design and implementation of various parallel algorithms. In the case of DFS, the existing research mainly revolves around variants of the classical nested DFS, where a back edge indicates the presence of a loop [25]. On the other hand, in BFS, loop detection is achieved using methods such as back-level edges [17]. The issue with explicit parallel model checking is that, despite the involvement of additional resources

TABLE I: PPSMC Experimental Result

Model	NuSMV	PSMC	$PPSMC_4$	$PPSMC_8$	$PPSMC_{16}$	$PPSMC_{32}$	Speedup	W/O LB	LB Improv.
abp8 p0	6.28	1.13	0.67	0.42	0.29	0.16	x7.06	0.28	75%
abp8 p1	7.86	2.01	1.39	0.83	0.52	0.29	x6.93	0.59	103%
prod-cons p0	36.17	4.03	2.97	1.59	0.89	0.46	x8.76	0.56	22%
prod-cons p1	18.34	7.21	4.79	2.57	1.31	0.79	x9.13	1.21	53%
prod-cons p2	5.57	2.35	1.82	1.08	0.54	0.31	x7.58	0.44	42%
bc57-sensors p0	32.42	21.58	14.51	9.1	5.73	3.01	x7.17	4.59	52%
bc57-sensors p1	560.11	189.19	91.26	55.94	32.39	19.87	x9.52	33.13	67%
dme5	15.88	23.42	10.83	7.83	4.74	3.18	x7.36	5.45	71%
dme6	36.69	59.07	34.44	20.8	12.01	7.98	x7.4	15.47	94%
viscoherence p0	12.1	4.76	3.24	2.83	2.32	1.68	x2.83	2.32	38%
viscoherence p1	9.56	5.68	4.35	3.67	2.63	2.02	x2.81	3.11	54%
cunim1ro	209.44	306.19	120.21	69.92	41.45	24.11	x12.7	34.85	45%
cuhanoi7ro	2.19	2.78	1.33	0.78	0.5	0.29	x9.59	0.57	97%
cuhanoi10ro	210.69	234.43	84.25	46.34	25.09	13.64	x17.19	17.47	28%
cuabq2mfro	173.69	233.59	154.21	92.04	54.71	26.74	x8.74	46.31	73%
phils p0	91.37	46.64	33.93	28.04	13.59	5.73	x8.14	10.59	84%
phils p1	140.18	36.76	24.09	18.35	10.31	5.96	x6.17	8.02	35%
msi-wtrans	41.36	21.33	12.15	6.73	3.72	1.74	x12.26	2.54	46%
syncarb	35.5	29.5	22.38	14.66	9.49	5.93	x4.97	9.43	59%
elevator	5.58	2.09	1.16	0.75	0.47	0.29	x7.21	0.5	72%
Average							x8.18		61%

TABLE III: Experimental Result of Different Tools

Model	Hold	Variables	Assignments	NuXmv	$PPSMC_{32}$	$Sylvan_{32}$	Improv.	$LTSmin_{32}$
abp8 p0	yes	49	13	4.32	0.16	0.26	63%	MO
abp8 p1	yes	49	13	9.93	0.29	0.5	72%	MO
prod-cons p0	no	28	14	1.01	0.46	0.58	26%	1.71
prod-cons p1	no	28	14	1.56	0.79	1.21	53%	1.64
prod-cons p2	yes	28	14	1.22	0.31	0.41	32%	45.26
bc57-sensors p0	yes	78	69	20.42	3.01	2.18	-28%	MO
bc57-sensors p1	no	78	69	116.75	19.87	31.41	58%	MO
dme5	no	93	96	0.59	3.18	2.76	-13%	TO
dme6	no	111	115	0.76	7.98	8.04	1%	TO
viscoherence p0	no	45	37	1.31	1.68	1.72	2%	2.35
viscoherence p1	no	45	37	1.88	2.02	1.96	-3%	3.02
cunim1ro	yes	38	32	0.77	24.11	26.06	8%	TO
cuhanoi7ro	no	18	13	87.86	0.29	0.32	10%	0.19
cuhanoi10ro	no	24	19	TO	13.64	15.61	14%	15.37
cuabq2mfro	yes	41	35	0.42	26.74	TO	-	MO
phils p0	no	32	29	0.14	5.73	6.21	8%	4.19
phils p1	no	32	29	0.19	5.96	9.30	56%	2.31
msi-wtrans	yes	45	21	0.32	1.74	2.15	24%	MO
syncarb	yes	90	60	2.07	5.93	12.77	115%	MO
elevator	yes	39	20	58.94	0.29	0.68	134%	MO
Average							33.1%	

in the parallel environment, its capability to handle large-scale problems remains limited [8].

Symbolic parallel model checking involves processing states as sets. However, treating states encountered during the search as sets poses a challenge in devising efficient nested fixpoint computing algorithms. As a result, current research primarily focuses on reachability analysis or safety properties. For instance, some studies have focused on verification of split-invariants for asynchronously processes communicating through shared-memory [15], and reachability analysis based on partitioned BDDs [26]. In contrast to these studies that primarily focus on safety properties or invariants, [16] proposed a symbolic algorithm for propositional μ-calculus by distributively evaluating subformulas. However, this approach is limited in terms of parallelism due to the relatively small scale of the formulas. The work presented in [27] concentrates on a fragment of CTL utilizing partitioned BDDs and fixpoint computation, as opposed to nested fixpoint computation. Moreover, there exist symbolic parallel techniques that exclusively focus on parallelizing supporting BDD libraries [20]. Nevertheless, these approaches overlook the parallelization of the high-level model checking itself. Our previous experimental results highlight the need for further significant improvement in their speedup. In our work, we concentrate on the Kripke structure and the LTL automaton. We propose a fully parallelized algorithm, while the approach does not incorporate any parallelization techniques.

VII. Conclusion

This paper introduces a novel approach for parallel LTL symbolic model checking. Departing from product computation between the model and the automaton, our approach computes images on the automaton states in parallel. Experimental results showcase the scalability of our method, outperforming existing methods that rely solely on parallel BDD library or are based on explicit parallel method.

Acknowledgment

This work was supported by the Foundation of Laboratory for Advanced Computing and Intelligence Engineering (Grant No. 2023-LYJJ-01-013), Beijing Municipal Natural Science Foundation (Grant No. 4252024), the Basic Research Projects from the Institute of Software, Chinese Academy of Sciences (Grant No. ISCAS-JCZD-202307) and the National Natural Science Foundation of China (Grant No. 62372438).

References

[1] E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith, *Model checking, 2nd Edition*. MIT Press, 2018.

[2] E. M. Clarke, W. Klieber, M. Novácek, and P. Zuliani, "Model checking and the state explosion problem," in *Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, ser. Lecture Notes in Computer Science, B. Meyer and M. Nordio, Eds., vol. 7682. Springer, 2011, pp. 1–30. [Online]. Available: https://doi.org/10.1007/978-3-642-35746-6_1

[3] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *J. ACM*, vol. 50, pp. 752–794, 9 2003. [Online]. Available: https://doi.org/10.1145/876638.876643

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10ˆ20 states and beyond," *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, 1992.

[5] A. Laarman, J. van de Pol, and M. Weber, "Multi-core ltsmin: Marrying modularity and scalability," in *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, ser. Lecture Notes in Computer Science, M. G. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi, Eds., vol. 6617. Springer, 2011, pp. 506–511. [Online]. Available: https://doi.org/10.1007/978-3-642-20398-5_40

[6] S. Ben-David, O. Grumberg, T. Heyman, and A. Schuster, "Scalable distributed on-the-fly symbolic model checking," *International Journal on Software Tools for Technology Transfer*, vol. 4, pp. 496–504, 2003.

[7] E. Bartocci, R. DeFrancisco, and S. A. Smolka, "Towards a gpgpu-parallel spin model checker," in *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*, ser. SPIN 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 87–96.

[8] J. Barnat, V. Bloemen, A. Duret-Lutz, A. Laarman, L. Petrucci, J. van de Pol, and E. Renault, "Parallel model checking algorithms for linear-time temporal logic," in *Handbook of Parallel Constraint Reasoning*, Y. Hamadi and L. Sais, Eds. Cham: Springer International Publishing, 2018, ch. 12, pp. 457–507. [Online]. Available: https://doi.org/10.1007/978-3-319-63516-3_12

[9] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, Inc., 1995.

[10] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis, "Memory-efficient algorithms for the verification of temporal properties," *Formal Methods Syst. Des.*, vol. 1, no. 2/3, pp. 275–288, 1992.

[11] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, pp. 677–691, 8 1986. [Online]. Available: https://doi.org/10.1109/TC.1986.1676819

[12] E. A. Emerson and C. Lei, "Efficient model checking in fragments of the propositional mu-calculus (extended abstract)," in *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*. IEEE Computer Society, 1986, pp. 267–278.

[13] S. Ben-David, T. Heyman, O. Grumberg, and A. Schuster, "Scalable distributed on-the-fly symbolic model checking," in *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, ser. Lecture Notes in Computer Science, W. A. H. Jr. and S. D. Johnson, Eds., vol. 1954. Springer, 2000, pp. 390–404. [Online]. Available: https://doi.org/10.1007/3-540-40922-X_24

[14] A. James, A. Tiu, and N. Yatapanage, "PFMC: A parallel symbolic model checker for security protocol verification," in *Formal Methods and Software Engineering - 23rd International Conference on Formal Engineering Methods, ICFEM 2022, Madrid, Spain, October 24-27, 2022, Proceedings*, ser. Lecture Notes in Computer Science, A. Riesco and M. Zhang, Eds., vol. 13478. Springer, 2022, pp. 173–189. [Online]. Available: https://doi.org/10.1007/978-3-031-17244-1_11

[15] A. Cohen, K. S. Namjoshi, Y. Sa'ar, L. D. Zuck, and K. I. Kisyova, "Parallelizing a symbolic compositional model-checking algorithm," in *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Barner, I. G. Harris, D. Kroening, and O. Raz, Eds., vol. 6504. Springer, 2010, pp. 46–59. [Online]. Available: https://doi.org/10.1007/978-3-642-19583-9_9

[16] O. Grumberg, T. Heyman, and A. Schuster, "Distributed symbolic model checking for μ-calculus," *Formal Methods in System Design*, vol. 26, pp. 197–219, 2005.

[17] M. Xie, Q. Yang, J. Zhai, and Q. Wang, "A vertex centric parallel algorithm for linear temporal logic model checking in pregel," *Journal of Parallel and Distributed Computing*, vol. 74, no. 11, pp. 3161–3174, 2014.

[18] R. Gerth, D. A. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995*, ser. IFIP Conference Proceedings, P. Dembinski and M. Sredniawa, Eds., vol. 38. Chapman & Hall, 1995, pp. 3–18.

[19] M. Böhm and E. Speckenmeyer, "A fast parallel sat-solver — efficient workload balancing," *Annals of Mathematics and Artificial Intelligence*, vol. 17, pp. 381–400, 1996.

[20] T. Van Dijk and J. Van de Pol, "Sylvan: multi-core framework for decision diagrams," *International Journal on Software Tools for Technology Transfer*, vol. 19, pp. 675–696, 2017.

[21] "Ppsmc artifact," https://github.com/gipsyh/PPSMC.

[22] G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, and T. van Dijk, "Ltsmin: High-performance language-independent model checking," in *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, ser. Lecture Notes in Computer Science, C. Baier and C. Tinelli, Eds., vol. 9035. Springer, 2015, pp. 692–707. [Online]. Available: https://doi.org/10.1007/978-3-662-46681-0_61

[23] K. Claessen and N. Sörensson, "A liveness checking algorithm that counts," in *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, G. Cabodi and S. Singh, Eds. IEEE, 2012, pp. 52–59. [Online]. Available: https://ieeexplore.ieee.org/document/6462555/

[24] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, 2014, pp. 334–342. [Online]. Available: https://doi.org/10.1007/978-3-319-08867-9_22

[25] A. Laarman and D. Faragó, "Improved on-the-fly livelock detection," in *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, ser. Lecture Notes in Computer Science, G. Brat, N. Rungta, and A. Venet, Eds., vol. 7871. Springer, 2013, pp. 32–47. [Online]. Available: https://doi.org/10.1007/978-3-642-38088-4_3

[26] T. Heyman, D. Geist, O. Grumberg, and A. Schuster, "Achieving scalability in parallel reachability analysis of very large circuits," in *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, ser. Lecture Notes in Computer Science, E. A. Emerson and A. P. Sistla, Eds., vol. 1855. Springer, 2000, pp. 20–35. [Online]. Available: https://doi.org/10.1007/10722167_6

[27] S. K. Iyer, D. Sahoo, C. Stangier, A. Narayan, and J. Jain, "Improved symbolic verification using partitioning techniques," in *Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Geist and E. Tronci, Eds., vol. 2860. Springer, 2003, pp. 410–424. [Online]. Available: https://doi.org/10.1007/978-3-540-39724-3_35

DCDiff: Enhancing JPEG Compression via Diffusion-based DC Coefficients Estimation

Ziyuan Zhang†, Han Qiu†*, Tianwei Zhang‡, Bin Chen§, Chao Zhang†
†Tsinghua University, China, ziyuan-z23@mails.tsinghua.edu.cn, {qiuhan,chaoz}@tsinghua.edu.cn
‡Nanyang Technological University, Singapore, tianwei.zhang@ntu.edu.sg
§Harbin Institute of Technology, Shenzhen, China, chenbin2021@hit.edu.cn

***Abstract*—JPEG is the most widely-used image compression method on low-cost cameras which cannot support learning-based compressors. One promising approach to enhance JPEG aims to drop DC coefficients at the cameras' ends (without extra computation) and reconstruct those DC coefficients after receiving them. They all face the challenge that their DC reconstruction relies on a statistical property, which will cause deviation-introduced errors and propagate. In this paper, we propose `DCDiff`, a novel end-to-end DC estimation method to tackle the above challenge. Instead of using statistical methods to recover DC coefficients and then fix errors, we directly leverage a generative model to estimate DC coefficients in an end-to-end manner. In the meantime, we generate masks to correct certain image locations that do not satisfy the statistical distribution to suppress error propagation. Extensive experiments show that `DCDiff` not only outperforms all baselines on compression performance but also introduces a tiny impact on downstream tasks and is fully compatible with 2 typical low-cost processors with JPEG support.**

***Index Terms*—JPEG, Image compression, Diffusion model**

I. Introduction

The deployment of numerous low-cost front-end cameras enables the development of smart Internet of Things (IoT) systems such as security surveillance [1], smart cities [2], and intelligent transportation [3]. Due to the limited computing and power capacity, front-end cameras capture large volumes of images but transmit them to cloud servers for further processing [4], which generates an increasing need for efficient image compression on front-end sensors. Many image compression approaches are proposed such as transformation-based ones (e.g. JPEG2000 [5], WebP [6], etc.) and learning-based ones (e.g. compressive sensing [7], neural image codec [8], etc.). However, many actual cameras tend to use JPEG [9] still due to JPEG's real-time capability, low computational requirements, and high compatibility. For example, ESP32-CAM, a widely-used low-power camera [10]–[12], has a max power of 1.55 Watt and can only support JPEG [13] instead of learning-based encoders.

JPEG is a tunable method to compress images block by block (e.g. 8×8 block) that allows users to choose a lower quality factor (i.e. Q table [14]) to enhance compression ratio on high-frequency (AC) coefficients [15]. However, using a lower-quality factor decreases image quality at decompression. As shown in Fig. 1, another idea [16] to enhance JPEG compression is to preserve AC coefficients but drop the direct current (DC) coefficients at the sender's end and recover them at the receiver's end. This is based on 3 reasons (details in Section II-B). First, DC coefficients are the average pixel values of each image block, which means after discarding DC, the difference between each pixel value within a block remains unchanged. Second, DC coefficients usually have large values, which need more bits to present when coding. Third, dropping DC coefficients does not require any modifications to the JPEG implementation, making it compatible with any low-cost cameras.

*Corresponding author.

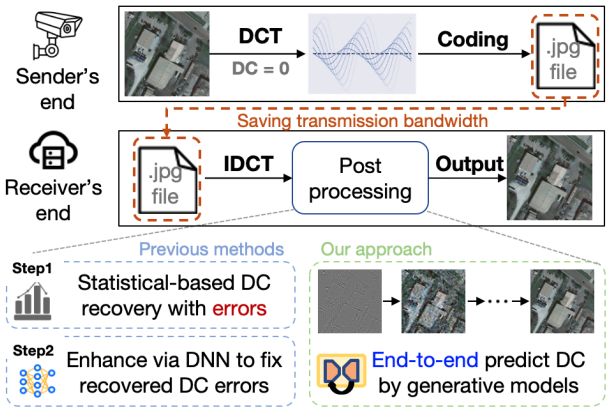


Fig. 1: Pipeline for DC recovery enhanced JPEG compression: existing methods (statistical-based DC recovery with errors need fixing) and our novel approach (i.e. a learned end-to-end method).

Various works [16]–[21] propose different approaches to tackle the challenge: *how to recover the DC coefficients more accurately from rest AC coefficients*. The fundamental idea is based on a statistical property: the differences between neighboring pixel values in natural images generally follow a Laplacian distribution [22]. Thus, based on this property, they recover one unknown DC coefficient from the known neighbor block to iteratively recover all DC coefficients. However, certain locations in natural images (e.g. less than 1% as pointed in [18]) deviate from this distribution, which causes an error when recovering the DC coefficient and will further propagate this error to limit the reconstructed image quality. Even though [19], [20] use deep neural networks (DNNs) to revise the errors after this iterative-based DC recovery, these two-step methods are still suboptimal since deviation-introduced errors are hard to fix.

In this paper, to avoid deviation-introduced errors, we propose `DCDiff` *to enhance JPEG via a learned end-to-end DC estimation*. Our basic idea is to avoid statistical recovery but directly leverage a pre-trained generative model (i.e., Stable Diffusion [23]) to utilize its knowledgeable priors for images to generate DC coefficients based on remaining AC coefficients. This is an end-to-end prediction process that can be optimized using learning methods with more data. Different from existing methods, `DCDiff` is not an iterative computation method block by block, so a deviation of image pixel values does not affect the whole image. However, as the generative model is not specifically designed for DC estimation, it faces challenges in predicting the content consistent average pixel value for each image block. To fill this gap, we propose a novel loss function to constrain the predicted image and the four corner blocks where the DC coefficients are retained to satisfy the Laplacian distribution while generating a mask to filter the image regions that deviate from the statistical property. What's more, to guide the diffusion model in

generating the DC coefficients, we propose a frequency modulation sampling strategy to enhance low-frequency generation.

We compare DCDiff with 3 state-of-the-art (SOTA) baselines across 6 datasets considering 4 metrics. Extensive experiments prove that DCDiff significantly outperforms baselines across different metrics on all datasets such as improving the reconstructed image quality with 3 $\sim$ 6.7 dB more PSNR. DCDiff also has minimal impact on typical downstream tasks such as image classification and can be fully compatible with 2 typical low-cost devices.

II. Preliminaries

A. JPEG compression in IoT system

Due to the broad compatibility and low computational requirements, JPEG is irreplaceable in IoT scenarios, such as safety and security monitoring [24]–[26]. For example, [25] use JPEG for image compression in resilient smart cities, monitoring in real time, and transmitting images to the cloud for processing. Figure 1 shows the workflow of JPEG in IoT systems. The sender's end is often equipped with IoT devices with limited memory and computing capacity, such as surveillance, which can only perform simple mathematical transformations on captured images. The process of JPEG compression includes Discrete Cosine Transform (DCT), quantization according to a quantization table, which determines the quality of compressed image, and entropy coding, typically Huffman coding. The compressed JPEG files are transmitted to the receiver's end, which are often cloud servers with powerful computing resources, for different applications.

Enhanced JPEG compression methods are proposed to save transmission bandwidth. [27] build a JPEG encoder for high visual quality of compressed image with high compression ratio, but introduces significant computation and memory overhead, which is infeasible for resource-constrained devices. [16]–[21] follows the pipeline that drops the DC coefficients at the sender's end to save the transmission bandwidth (see Fig. 2, DC coefficients are generally larger) without any overhead and recover the image at the receiver's end. In this paper, we follow this setting of saving transmission bandwidth by dropping DC coefficients and focus on the post-processing stage at the receiver's end to recover higher-quality images.

B. DC coefficient estimation

DC estimation was first proposed in [22], which demonstrated that it is possible to recover DC coefficient from AC coefficient with reasonable image quality in JPEG encryption systems to provide higher protection for DC coefficient. This method is based on the Laplacian property that the difference between two neighboring pixels follows a Laplacian distribution with zero mean and small variance, as shown in Figure 4. The blue line "w/o mask" means the difference between adjacent pixels across the entire image. Denote $B(i,j)$ as the 8×8 blocks at the $i-th$ row and $j-th$ column of the image. DC value is the average of $B(i,j)$, which is the same for all pixels in $B(i,j)$. Assuming the AC value of $B(i,j)$ and the neighbor block $B(i,j-1)$ is known and DC value of $B(i,j-1)$, $dc_{i,j-1}$, is also known. Based on the Laplacian property, it is highly likely that the pixel values between two adjacent blocks are very similar. [22] minimizes the differences of pixels between two adjacent blocks from three directions to identified DC values.

Recently, some other works have applied DC estimation methods to enhance JPEG compression and further improve DC estimation accuracy. [18] improved the DC estimation from minimizing the value between adjacent pixels to minimizing the distribution trends of the last two columns and the first column of $B(i,j)$ and $B(i,j+1)$.

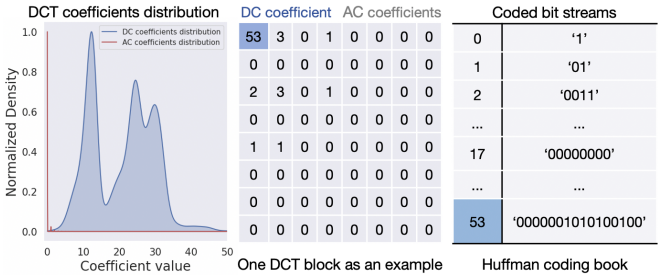


Fig. 2: An example of AC and DC coefficients distribution and corresponding Huffman coding.

With the development of the neural network, [19] proposed a residual network to improve the image quality after calculating the DC coefficient using the methods proposed in [18]. Observing that the optimal prediction direction of each boundary pixel pair is inconsistent, [20] proposed a novel prediction pattern, which chooses pixel pairs from different directions, to improve the DC estimation.

Although the above methods use different pixel pair patterns to suppress the error propagation caused by the region deviated from Laplacian distribution, there are still some pixels that exhibit a significant difference compared to adjacent pixels in any three directions. Furthermore, the iterative process between blocks will amplify this error. To solve the above problems, we first analyze the characteristics of the region deviated from the Laplacian distribution. In addition, instead of using a block-by-block iterative approach, we directly predict the value of each pixel using a neural network, which can avoid the error propagation between adjacent pixels.

III. Method

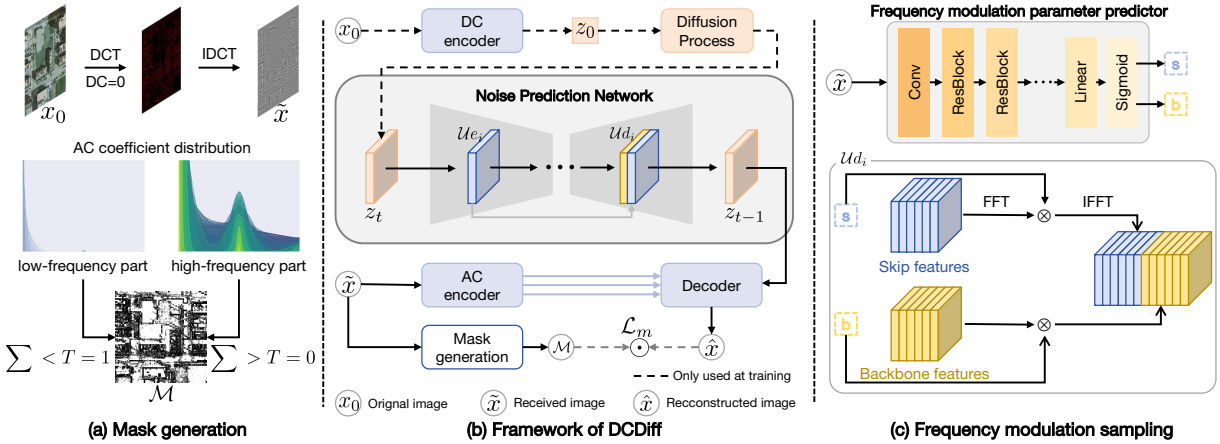
A. Overview

In this section, we describe in detail our end-to-end post-processing method based on generative models. Figure 3 shows our major component. Compared with previous methods [19], DCDiff no longer relies on mathematical methods to iterate between DCT blocks to calculate the DC coefficients, but directly predicts the entire image using the diffusion model (section III-B). To constrain the reconstructed image to have content consistency with the origin image, DCDiff is trained using a masked Laplacian distribution loss to ensure the reconstructed images satisfy the Laplacian property while filtering the regions that may cause error (section III-C). Finally, to enhance the DC coefficients generated by the diffusion model, DCDiff uses a frequency modulation sampling strategy during the inference process, and a frequency modulation parameter predictor is applied to predict different scale factors adapting to different input images (section III-D).

B. DC coefficient feature extraction and generation

We denote the origin image as x_0, the reconstructed image as $\hat{x}$, and $\tilde{x}$ presents the image without DC coefficients after the receiver performs IDCT.

DC coefficients feature extraction. As illustrated in section I, DC coefficients represent the average of each image block, which means the DC coefficients have a much smaller feature space than the origin image. First, we introduce an image encoder (DC encoder) and an image decoder, denoted as $\mathcal{E}^{DC}$ and $\mathcal{D}$, to compress the image into a smaller latent space, which can present the DC component feature space, and the extracted feature denoted as z_0. To ensure that $\mathcal{E}^{DC}$ pays more attention to extracting the DC component of the image, we introduce another image encoder (AC encoder, $\mathcal{E}^{AC}$), taking $\tilde{x}$



(a) Mask generation (b) Framework of DCDiff (c) Frequency modulation sampling

Fig. 3: Main components of DCDiff. (a) The insight to generate a spatial mask to apply Laplacian property (section III-C). (b) The pipeline of DCDiff to recover the JPEG images at the receiver's end (section III-B). (c) During inference, frequency modulation sampling is used to enhance the low-frequency generation (section III-D).

as input. $\mathcal{E}^{AC}$ can only extract the feature of AC coefficients, as the DC coefficients in $\tilde{x}$ have been set to zero at the sender's end. The image decoder $\mathcal{D}$ receives two parts of information, one of which is the AC features from the AC encoder, and the other is z_0 from the DC encoder. To better reconstruct the image, $\mathcal{D}$ needs both AC features and DC features, which forces the DC encoder to extract the features of DC coefficients from the origin image.

Diffusion generation. The diffusion model [28] is a generative model that can generate images from noise through multi-step denoising. It includes two processes of diffusion and denoising. During the diffusion process, the diffusion model gradually adds Gaussian noise to the image through a T-step Markov process until the image becomes complete Gaussian noise, following a variance schedule denoted by hyperparameter β_t (Eq. 1).

$$q\left(z_t|z_{t-1}\right) = \mathcal{N}\left(z_t; \sqrt{1-\beta_t}z_{t-1}, \beta_t\mathcal{I}\right) \tag{1}$$

The denoising process is the reverse of the diffusion process. Given z_t, a noise prediction network ϵ_θ is used to predict noise added to the image at t-th step to recover z_{t-1} (Eq. 2).

$$z_t = \sqrt{1-\beta_t}z_{t-1} + \beta_t\epsilon_t, \tag{2}$$

where ϵ_t is the noise predicted by the noise prediction network.

In this article, we aim to leverage the powerful generation ability and knowledgeable priors for images of the diffusion model to generate the DC coefficients of the images. Figure 3 (b) shows the framework of DCDiff. We use a noise prediction network to predict the DC features z_0 from Gaussian noise. Although the diffusion model demonstrates superior performance in generation tasks, it lacks the ability to preserve the structure information of the input images. Inspired by [29], we use a control module to extract the structure information from $\tilde{x}$ and inject it into the convolution block of diffusion model to improve structure consistency.

C. Masked Laplacian distribution loss

While the diffusion model with the control module demonstrates superior performance in image generation with given structure information, it lacks the ability to preserve the content consistency. For example, with no constraints, the generated image $\hat{x}$ may show a car that is yellow or blue—both appearing harmonious to the human but differing from x_0, mainly due to no explicit color information in $\tilde{x}$. To solve this problem, we propose the Laplacian distribution loss to constrain the reconstructed image to have content consistency with four corner blocks, whose DC coefficients are retained.

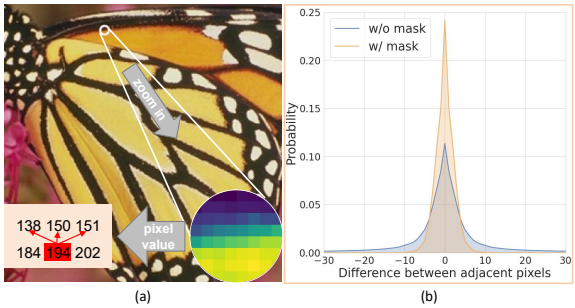


Fig. 4: (a) An example of pixel pairs have an abrupt change and significant differences across all three prediction directions. (b) comparison of the distribution of the difference between adjacent pixels with the high-frequency mask.

We first analyze the characteristics of pixel pairs that deviate from the Laplacian distribution. For example, as shown in Figure 4 (a), there are significant differences in pixel pairs across all three prediction directions. As a result, selecting any direction as the prediction direction will lead to substantial errors. We find that these pixel pairs are usually located in image regions with complex textures and sharp edges, where the pixel values exhibit significant fluctuations. The abrupt changes in pixel values in the spatial domain correspond to the increase in high-frequency components in the frequency domain. As illustrated in Figure 4 (b), we generate a mask to set all high-frequency components in the image to zero and recalculate the pixel value distribution of the low-frequency components in the image. Without the high-frequency components, the difference between two neighboring pixels follows the distribution with a much smaller variance. And the probability of pixel pairs with similar values is much higher than without the mask. Therefore, it is more appropriate to apply the Laplacian property to the low-frequency part of the image to calculate accurate pixel values.

Directly applying a low-pass filter in the frequency domain to filter out high frequencies may lead to changes in pixel values in the spatial domain, thereby affecting the accuracy of the prediction results. So we try to generate a spatial mask. When DC coefficients are zero, after IDCT, the pixel values of $\tilde{x}$ are the weighted sum of AC coefficients. As illustrated in Figure 3 (a), we show the AC coefficients of different parts of the images, the low-frequency part

corresponding to the small pixel values part and the high-frequency part corresponding to the large pixel values part. Thus, the larger the value of $\tilde{x}$, the more high-frequency components correspond at this position. So we follow Equation 3 to generate the mask.

$$\mathcal{M}(i,j) = \begin{cases} 0, \tilde{x}(i,j) > \mathcal{T} \\ 1, \tilde{x}(i,j) \leqslant \mathcal{T} \end{cases} \tag{3}$$

where $\mathcal{T}$ is the threshold used to control the number of high-frequency components to be masked, and we discuss in detail the effect of different $\mathcal{T}$ on the results in section IV-C.

After obtaining the mask, we calculate the Laplacian constraint between pixels in the low-frequency region and define the masked Laplacian distribution (MLD) loss as Equation 4.

$$\mathcal{L}_m = \sum_{i,j} \mathcal{M}_{i,j} \odot ((\Delta_h \hat{x}_{i,j} - \Delta_h \hat{x}_{i,j-1})^2 + (\Delta_w \hat{x}_{i,j} - \Delta_w \hat{x}_{i-1,j})^2) \tag{4}$$

where $\Delta_h \hat{x}_{i,j}$ denotes $\hat{x}_{i,j-1} - \hat{x}_{i,j}$.

D. Frequency modulation sampling strategy

As a generative model, the diffusion model tends to produce high-frequency detail information in images. However, in this paper, we aim to use the diffusion model to generate the DC coefficients, which represent the low-frequency features of the images. Inspired by [30], which investigates that the noise prediction network's skip connections mainly introduce high-frequency features, we try to modulate the frequency to enhance the low-frequency generation. [30] proposes to re-weight the backbone features and skip features during the denoising process using two manual scale factors. However, for image restoration, since the frequency information varies for each image, manually adjusting these two scale factors as hyperparameters is not suitable for every image.

To accommodate the frequency variations of different images, as shown in Figure 3 (c), we use a frequency modulation parameter predictor (FMPP), which relies on the ResNet architecture, to predict the scale parameters s and b. The predictor taking $\tilde{x}$ as input, estimates the scale factors to control the intensity of the enhancement of low-frequency information based on the high-frequency information contained in $\tilde{x}$. To control the magnitude of the enhancement, a sigmoid function is used in the final layer to regulate the range of the scale factor. Based on the experimental results in [30], we constrain the scale factor between 0 and 2. During the denoising process, FMPP initially takes $\tilde{x}$ as input to predict the scale factors and then DDIM is applied to denoise from noise, generating the DC coefficients of the images.

E. Training details

The training is divided into two stages. In the first stage, we train $\mathcal{E}^{DC}$, $\mathcal{E}^{AC}$ and $\mathcal{D}$ together using the loss function as Equation 5.

$$\mathcal{L}_{fir} = \mathcal{L}_{rec} + \mathcal{L}_{per} + \mathcal{L}_{dis}, \tag{5}$$

where $\mathcal{L}_{rec}$ is the $\mathcal{L}_1$ loss between the reconstructed image $\hat{x}$ and the origin image x_0, $\mathcal{L}_{per}$ is the perceptual loss and $\mathcal{L}_{dis}$ is the discriminator loss. In the second stage, we freeze the parameters of $\mathcal{E}^{DC}$, $\mathcal{E}^{AC}$ and $\mathcal{D}$, and train the noise prediction network with the goal of making the DC features $z_0\prime$ generated by the diffusion model as close as possible as z_0. Further, to ensure the generated image satisfies the Laplacian constraint, during each training step, we use the predicted noise ϵ_t to project from z_t to $z_0\prime$ following the Markov process: $z_0\prime = \frac{z_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t}{\sqrt{\bar{\alpha}_t}}$, where $\bar{\alpha}_t = \prod_{i=1}^{t} \sqrt{1-\beta_i}$. Then use the $\mathcal{D}$ to project $z_0\prime$ to the pixel space and calculate the masked Laplacian distribution loss. The loss function used for training the noise prediction network is as Equation 6.

$$\mathcal{L}_{sec} = \mathcal{L}_{ldm} + \sigma \mathcal{L}_m, \tag{6}$$

where $\mathcal{L}_{ldm}$ is the MSE loss to minimize the noise distance between the target and learning noise in latent space, and σ is the weighting parameter. For FMPP, we freeze all other parameters and optimize FMPP using MSE loss, minimizing the distance between x_0 and $\hat{x}$ sampled under the scale factors predicted by FMPP.

IV. Experiment

A. Experiment setup

Dataset. For training data, we randomly select 300K images from OpenImages [31] and randomly crop the image to a resolution of 256 × 256 pixels. For testing data, we use 5 general CV datasets and one dedicated remote sensing dataset [32].

Metrics. We follow [33] to use 4 metrics for quantitative measures. We use the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) to validate the pixel fidelity and the structure similarity. What's more, we use another two perceptual metrics, Multi-Scale Structural Similarity Index Measure (MS-SSIM) and Learned Perceptual Image Patch Similarity (LPIPS) to evaluate the quality of reconstructed images for human perception.

Implementation details. For the first training stage, we train the DC encoder $\mathcal{E}^{DC}$, the AC encoder $\mathcal{E}^{AC}$ and the decoder $\mathcal{D}$ together using the loss function in Equation 5 for 5 epochs with a batchsize of 16 and a learning rate of $1e-4$. For the second training stage, we use the pre-trained stable diffusion [23] model as noise prediction network and finetune it using $\mathcal{L}_{ldm}$ for 5 epochs with a learning rate of $1e-4$ to learn how to retain structural information in $\tilde{x}$ first. And then we add $\mathcal{L}_m$ to loss function, and finetune the model for 10 epochs with a learning rate of $1e-5$ and $\sigma = 2e-4$ to ensure generated images satisfy Laplacian constraint. All training experiments are performed on a server with 8× Nvidia H800 GPUs using the Adam optimizer. During inference, we utilize the DDIM sampling [34] with 50 steps. For baseline comparison, we consider three baselines for DC estimation, including methods based on mathematical calculations [18], [20] and based on neural networks [19].

B. Evaluation results

Qualitative comparisons. Table I shows the comparison results with baselines on six datasets. We compress the RGB images in each dataset with the standard Q_{50} quantization table following the previous works. Compared to compressing grayscale images, during JPEG compression, the discarded DC coefficients of the U and V channels retrain the chrominance information of the image. The error propagation caused by incorrect DC coefficients for U and V channels may greatly affect the color of the final reconstructed image, resulting in lower PSNR and SSIM for SmartCom 2019 [18] and ICIP 2022 [20]. Note that due to AC coefficients retaining high-frequency details from the origin image, the detailed information of the image is not lost during the discarding of DC coefficients, resulting in a relatively low LPIPS. Therefore, compared to directly using a low-quality Q quantization table to reduce the amount of transmission data, discarding the DC coefficients can better preserve image detail without affecting human perception. IEEE TII 2021 [19] tries to use a residual neural network to correct error propagation. However, optimizing solely for mean square error (MSE) can lead to excessive image smoothing, corrupting the detail of high-frequency information in the image, resulting in visual artifacts. As shown in Table I, the LPIPS of IEEE TII 2021 is much higher than others. In contrast,

TABLE I: Quantitative results of comparison DCDiff with 3 baselines on 6 datasets. Throughout this paper, ↑ (↓) means higher (lower) is better. Best performances are highlighted in **bold**.

Methods	Set5				Set14				Kodak			
	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓
SmartCom 2019 [18]	21.70	0.8557	0.9501	0.0396	21.34	0.8804	0.9224	0.0508	22.09	0.9132	0.8861	0.0533
IEEE TII 2021 [19]	22.99	0.8819	0.9511	0.0408	21.40	0.8967	0.9203	0.0582	21.87	0.9101	0.8809	0.0624
ICIP 2022 [20]	23.36	0.8764	0.9585	0.0354	22.04	0.8954	0.939	0.0436	23.47	0.9296	0.9134	0.0437
DCDiff	**28.46**	**0.9494**	**0.9652**	**0.0254**	**25.84**	**0.9498**	**0.9490**	**0.0280**	**26.96**	**0.9544**	**0.9358**	**0.0309**
Method	BSDS200				Urban100				Inria			
	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓
SmartCom 2019 [18]	21.68	0.8981	0.9049	0.0502	19.34	0.8681	0.8759	0.0517	21.15	0.9207	0.9066	0.0497
IEEE TII 2021 [19]	21.85	0.9030	0.9019	0.0579	19.62	0.8782	0.8737	0.0585	20.96	0.9116	0.9001	0.0623
ICIP 2022 [20]	22.76	0.9128	0.9264	0.0429	20.67	0.8902	0.9050	0.0416	22.53	0.9381	0.9290	0.0421
DCDiff	**25.76**	**0.9481**	**0.9436**	**0.0314**	**26.01**	**0.9553**	**0.9532**	**0.0187**	**27.94**	**0.9556**	**0.9457**	**0.0308**

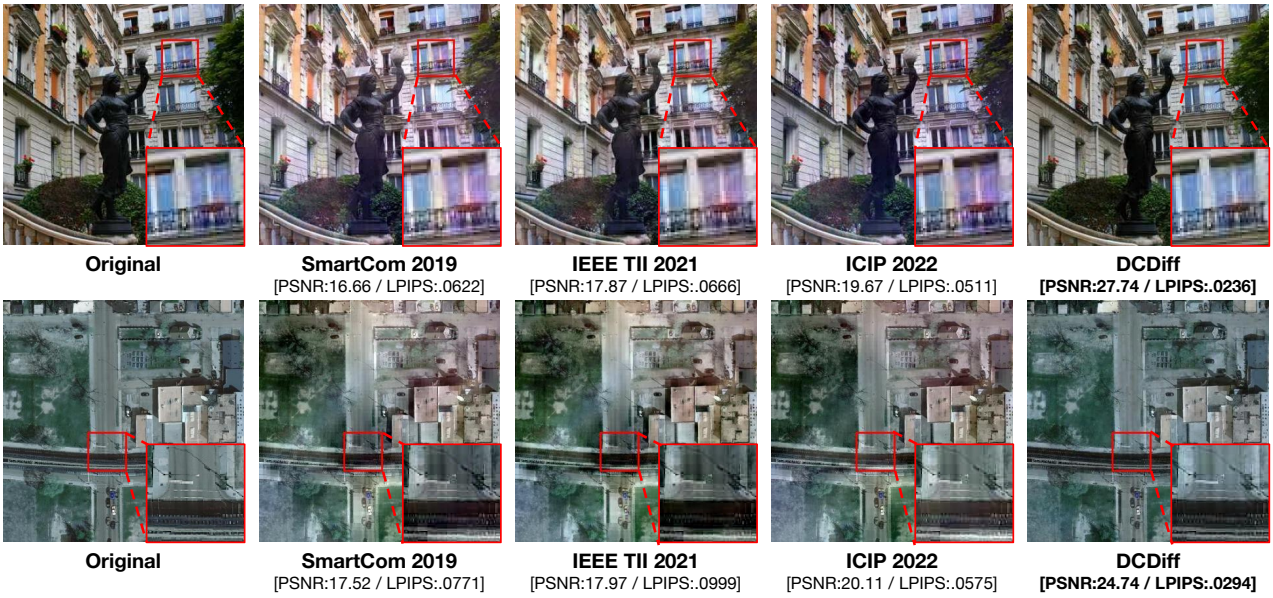


Fig. 5: The reconstructed image after post-processing.

DCDiff is not a block-based iterative method, which avoids error propagation. Additionally, it constrains the generated images using a mask Laplacian distribution loss, preventing excessive smoothing. The results demonstrate that DCDiff can achieve SOTA performance across all metrics on all datasets, increasing PSNR by $3 \sim 6.7dB$.

Visual results. Figure 5 shows the reconstructed images after post processing of DCDiff. We selected a street-view image and an aerial image for visualization comparison. As analyzed above, error propagation in U and V channels may lead to unnatural colors, such as the purple windows in the first row of Figure 5. Furthermore, the baseline's optimization objective is to enforce Laplacian constraints as closely as possible between any two adjacent blocks, which leads to color bleeding in areas with abrupt pixel value changes. For example, at the intersection in the image of the second row, the vertical road is dyed black. Since we use a mask to filter out high-frequency parts of the image and do not rely on block-based iteration, there is no color bleeding or unnatural colors due to error propagation, achieving SOTA reconstruction results.

Compression Ratio. At the sender's end, we follow the setting in [19] to set all DC coefficients to zero except 4 corner blocks, and calculate the compression ratio compared with standard JPEG using standard JPEG coding. As shown in Table II, we first use the same Q-table (i.e. Q_{50} quantization table) for standard JPEG and DCDiff. On all datasets, just by dropping DC coefficients, DCDiff can compress 25% more than JPEG on average. Besides, we also compare DCDiff with more compression of standard JPEG when using a more aggressive Q-table. We tune the Q-table of JPEG until the reconstructed images achieve a similar image quality (i.e. LPIPS) after DCDiff post-processing. While maintaining the same image quality reconstructed at the receiver's end, DCDiff can still compress 10-14% more than tuning Q-table in default JPEG.

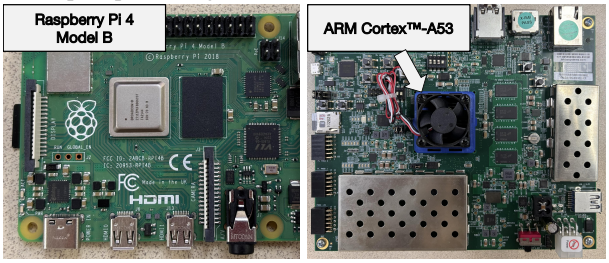


Fig. 6: Two low-cost testbeds for evaluating DCDiff.

C. Ablation study

W/o MLD loss. The masked Laplacian distribution loss can better guide the image reconstruction based on DC coefficients of four corner blocks and AC coefficients. We remove the MLD loss and train the noise prediction network until converges. As shown in Table III, the results show that without MLD loss, the model cannot capture the relation between four corner blocks and the other part of the images, leading to noticeable color distortion in reconstructed images.

W/o FMPP. To show that the frequency modulation can enhance the low-frequency information, we remove the frequency modulation

TABLE II: Compression ratios of JPEG with zero DC coefficients except for 4 corner blocks.

Dataset	Set5	Set14	Kodak	BSDS200	Urban100	Inria
	Compression ratio of JPEG (the same Q-table, Q_{50})					
min	68.66%	68.37%	63.53%	59.96%	63.89%	43.71%
max	81.11%	90.19%	83.39%	86.26%	93.96%	86.34%
avg	74.73%	78.64%	73.41%	75.99%	81.38%	78.89%
	Compression ratio of JPEG under similar LPIPS					
min	77.84%	77.65%	75.16%	66.38%	68.62%	46.62%
max	96.00%	97.70%	98.76%	98.30%	98.14%	95.24%
avg	86.98%	89.16%	89.10%	86.45%	88.50%	86.46%

TABLE III: Ablation study of DCDiff's variants. "w/o MLD" indicates removing masked Laplacian distribution loss at the 2nd stage training. "w/o FMPP" denotes that during sampling the predictor is removed. Different thresholds are evaluated when generating masks.

Dataset	Methods	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓
Kodak	w/o MLD	24.80	0.9425	0.9238	0.0380
	w/o FMPP	26.57	0.9544	0.9341	0.0321
	$\mathcal{T}=0$	26.04	0.9530	0.9325	0.0324
	$\mathcal{T}=5$	26.11	0.9539	0.9341	0.0322
	$\mathcal{T}=10$	26.96	0.9544	0.9358	0.0309
	$\mathcal{T}=15$	26.17	0.9487	0.9307	0.0347
Inria	w/o MLD	26.35	0.9511	0.9409	0.0356
	w/o FMPP	27.46	0.9553	0.9445	0.0319
	$\mathcal{T}=0$	27.29	0.9553	0.9449	0.0322
	$\mathcal{T}=5$	27.59	0.9555	0.9456	0.0316
	$\mathcal{T}=10$	27.94	0.9556	0.9457	0.0308
	$\mathcal{T}=15$	27.17	0.9497	0.9412	0.0349

parameter predictor during sampling. The scale factors s and b are equal to 1, which is the same as regular DDIM sampling. The results in Table III show that frequency modulation can help to enhance the backbone features and improve the quality of reconstructed images.

Influence of mask threshold. We further investigate the impact of the threshold used to generate the mask $\mathcal{M}$ on the reconstructed image quality as shown in Table III. A smaller threshold results in most of the image content being masked, leaving only a small portion of the image associated with the information from the four corner blocks, which causes the model to rely on diffusion to generate most of the unconstrained content. A larger threshold causes some high-frequency information in the image, such as object edges, to be included to calculate the loss, which results in blurred edges around objects in the image, creating visual artifacts. Based on the results in Table III, we select $\mathcal{T} = 10$ as the threshold to generate the mask, which can capture most of the image information while filtering out high-frequency content such as object edges.

D. Use case analysis in IoT systems

We deploy DCDiff on 2 widely-used low-power processors: ARM Cortex™-A53[1] and Raspberry Pi 4 Model B[2] (as shown in Fig. 6) for evaluation. On these 2 processors, we compress and transmit captured images using standard JPEG and DCDiff and calculate the throughput. The results in Table IV show that compared to standard JPEG, DCDiff does not add extra computational overhead, which proves that DCDiff can be easily adapted to low-cost front-end cameras with JPEG support.

E. Influence on downstream tasks

Images reconstructed at the receivers' ends are usually used for further processing. To confirm that DCDiff has a tiny influence on the downstream tasks, we choose a remote-sensing image classification task [35] as an example. For all baselines, we use the same setting to drop DC coefficients at the sender's end and input the reconstructed images after post-processing to the image classification model at the receiver's end. As shown in Table V, DCDiff outperforms all baselines with a tiny impact on model accuracy (drop 0.49%).

[1]https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a53

[2]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

TABLE IV: Deploying DCDiff on 2 low-power front devices.

Methods	Compression throughput (Gbps)	
	Raspberry Pi 4	ARM Cortex™-A53
JPEG Encoder	1.85	0.92
DCDiff Encoder	1.95	0.92

TABLE V: Post-processing influence on a remote-sensing image classification tasks. ↓ denotes the accuracy reduction.

Methods	Original	SmartCom 2019 [18]	IEEE TII 2021 [19]	ICIP 2022 [20]	DCDiff
ACC (%)	97.54	↓ 1.96%	↓ 4.42	↓ 1.47	↓ **0.49**

V. Discussion and Future Work

JPEG v.s. Learned image compression. Transformation-based image compression method mainly applies a spatial-to-frequency transformation (e.g. DCT, DWT) to get the frequency coefficients and then uses a tunable quantization step to compress them. Today, they are outperformed by novel learned compression methods [8], [36]–[38] in various scenarios. Those learned compression methods generally replace the transformation-based compressor with a DNN-based encoder to extract image features and use a decoder to reconstruct images. Since DNNs can be trained to fit data in a similar distribution, learned image compression methods can outperform static ones (e.g. JPEG/JPEG2000) with sufficient data to train powerful encoders [39], [40]. However, there are still a large number of cameras that do not have enough resources to support a DNN-based encoder. On the contrary, JPEG, as the most widely-used image compression method, is well implemented on all existing cameras such that enhanced JPEG methods like DCDiff can also be easily adopted in real-world scenarios to improve image compression.

More improvements in future. First, in this paper, we use the default coding technique of JPEG for evaluating DCDiff. We are aware that, as a step after transformation and quantization, better coding techniques [41], [42] may get better results. Thus, it is important to emphasize that DCDiff studies how to reduce the number of DCT coefficients which is orthogonal to coding techniques. Second, DCDiff uses the diffusion model to reconstruct the images but it can be replaced as any other generative models as long as they can be trained to get rid of deviation-induced errors.

VI. Conclusion

We present DCDiff, an end-to-end DC estimation method for enhanced JPEG compression based on generative models. We propose the MLD loss to suppress error propagation while improving the content consistency of reconstructed images and a frequency modulation sampling strategy to enhance DC coefficient generation. Extensive results indicate that DCDiff can achieve SOTA performance to recover high-quality images and is compatible with various IoT devices to save transmission bandwidth while maintaining high throughput.

Acknowledgment

This work is supported by NSFC under grant No. U24A20337 and 62301189, and also the Center of High performance computing, Tsinghua University, China.

REFERENCES

[1] M. Imani, A. Zakeri, H. Chen, T. Kim, P. Poduval, H. Lee, Y. Kim, E. Sadredini, and F. Imani, "Neural computation for robust and holographic face detection," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 31–36.

[2] S. Xia, T. Xing, C. Q. Wu, G. Liu, J. Yang, and K. Li, "AQMon: A Fine-grained Air Quality Monitoring System Based on UAV Images for Smart Cities," *ACM Transactions on Sensor Networks*, vol. 20, no. 2, pp. 1–20, 2024.

[3] S. Liu, B. Yu, J. Tang, and Q. Zhu, "Towards fully intelligent transportation through infrastructure-vehicle cooperative autonomous driving: Challenges and opportunities," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1323–1326.

[4] C. Jia, M. Hu, Z. Chen, Y. Yang, X. Xie, Y. Liu, and M. Chen, "AdaptiveFL: Adaptive heterogeneous federated learning for resource-constrained AIoT systems," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[5] R. M. Asiyabi, A. Anghel, A. Focsa, M. Datcu, M. Martone, P. Rizzoli, and E. Imbembo, "On the use of JPEG2000 for SAR raw data compression," in *EUSAR 2024; 15th European Conference on Synthetic Aperture Radar*. VDE, 2024, pp. 249–253.

[6] J. Bankoski, P. Wilkins, and Y. Xu, "Technical overview of vp8, an open source video codec for the web," in *2011 IEEE International Conference on Multimedia and Expo*. IEEE, 2011, pp. 1–6.

[7] Y. Huang, B. Chen, J. Zhang, Q. Han, and S.-T. Xia, "Compressive sensing based asymmetric semantic image compression for resource-constrained iot system," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 877–882.

[8] R. Yang and S. Mandt, "Lossy image compression with conditional diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] G. K. Wallace, "The jpeg still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, 1991.

[10] R. Salikhov, V. K. Abdrakhmanov, and I. Safargalin, "Internet of things (iot) security alarms on esp32-cam," in *Journal of Physics: Conference Series*, vol. 2096, no. 1. IOP Publishing, 2021, p. 012109.

[11] K. Elhattab, K. Abouelmehdi, and S. Elatar, "New model to monitor plant growth remotely using esp32-cam and mobile application," in *2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2023, pp. 1–6.

[12] T. Jaya, S. Nath *et al.*, "Surveillance monitoring using esp32-cam module for fog detection in agriculture." *Grenze International Journal of Engineering & Technology (GIJET)*, vol. 10, no. 1, 2024.

[13] https://github.com/espressif/esp32-camera.

[14] X. Luo, H. Talebi, F. Yang, M. Elad, and P. Milanfar, "The rate-distortion-accuracy tradeoff: JPEG case study," *arXiv preprint arXiv:2008.00605*, 2020.

[15] Y. Peng, C. Fu, G. Cao, W. Song, J. Chen, and C.-W. Sham, "JPEG-compatible joint image compression and encryption algorithm with file size preservation," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 20, no. 4, pp. 1–20, 2024.

[16] S. Li, J. J. Ahmad, D. Saupe, and C.-C. J. Kuo, "An improved DC recovery method from AC coefficients of DCT-transformed images," in *2010 IEEE International Conference on Image Processing*. IEEE, 2010, pp. 2085–2088.

[17] S. Ong, S. Li, K. Wong, and K. Tan, "Fast recovery of unknown coefficients in DCT-transformed images," *Signal Processing: Image Communication*, vol. 58, pp. 1–13, 2017.

[18] H. Qiu, Q. Zheng, M. Qiu, and G. Memmi, "Dc coefficients recovery from ac coefficients in the jpeg compression scenario," in *Smart Computing and Communication: 4th International Conference, SmartCom 2019, Birmingham, UK, October 11–13, 2019, Proceedings 4*. Springer, 2019, pp. 266–276.

[19] H. Qiu, Q. Zheng, G. Memmi, J. Lu, M. Qiu, and B. Thuraisingham, "Deep residual learning-based enhanced jpeg compression in the internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 2124–2133, 2020.

[20] J. Zhang, B. Chen, Y. Huang, H. Qiu, Z. Wang, and S. Xia, "Improved dc estimation for jpeg compression via convex relaxation," in *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2022, pp. 2461–2465.

[21] M. Ouyang and Z. Chen, "JPEG Quantized Coefficient Recovery via DCT Domain Spatial-Frequential Transformer," *IEEE Transactions on Image Processing*, 2024.

[22] T. Uehara, R. Safavi-Naini, and P. Ogunbona, "Recovering DC coefficients in block-based DCT," *IEEE Transactions on Image Processing*, vol. 15, no. 11, pp. 3592–3596, 2006.

[23] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[24] A. Sharma, P. K. Singh, and Y. Kumar, "An integrated fire detection system using iot and image processing technique for smart cities," *Sustainable Cities and Society*, vol. 61, p. 102332, 2020.

[25] S. A. Shah, D. Z. Seker, M. M. Rathore, S. Hameed, S. B. Yahia, and D. Draheim, "Towards disaster resilient smart cities: Can internet of things and big data analytics be the game changers?" *IEEE Access*, vol. 7, pp. 91 885–91 903, 2019.

[26] F. Saeed, A. Paul, A. Rehman, W. H. Hong, and H. Seo, "Iot-based intelligent modeling of smart home environment for fire prevention and safety," *Journal of Sensor and Actuator Networks*, vol. 7, no. 1, p. 11, 2018.

[27] https://github.com/google/guetzli.

[28] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[29] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3836–3847.

[30] C. Si, Z. Huang, Y. Jiang, and Z. Liu, "Freeu: Free lunch in diffusion u-net," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4733–4743.

[31] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit *et al.*, "Openimages: A public dataset for large-scale multi-label and multi-class image classification," *Dataset available from https://github. com/openimages*, vol. 2, no. 3, p. 18, 2017.

[32] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *2017 IEEE International geoscience and remote sensing symposium (IGARSS)*. IEEE, 2017, pp. 3226–3229.

[33] Z. Wang, D. Li, G. Li, Z. Zhang, and R. Jiang, "Multimodal low-light image enhancement with depth information," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 4976–4985.

[34] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[35] A. Bahri, S. G. Majelan, S. Mohammadi, M. Noori, and K. Mohammadi, "Remote sensing image classification via improved cross-entropy loss and transfer learning strategy based on deep convolutional neural networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 17, no. 6, pp. 1087–1091, 2019.

[36] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan, "DeepN-JPEG: A deep neural network favorable JPEG-based image compression framework," in *Proceedings of the 55th annual design automation conference*, 2018, pp. 1–6.

[37] F. Mentzer, G. D. Toderici, M. Tschannen, and E. Agustsson, "High-fidelity generative image compression," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 913–11 924, 2020.

[38] M. Li, S. Gao, Y. Feng, Y. Shi, and J. Wang, "Content-oriented learned image compression," in *European Conference on Computer Vision*. Springer, 2022, pp. 632–647.

[39] Y. Yang and S. Mandt, "Computationally-efficient neural image compression with shallow decoders," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 530–540.

[40] J. Liu, H. Sun, and J. Katto, "Learned image compression with mixed transformer-cnn architectures," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14 388–14 397.

[41] Z. Hu, Z. Chen, D. Xu, G. Lu, W. Ouyang, and S. Gu, "Improving deep video compression by resolution-adaptive flow coding," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 193–209.

[42] J. Ballé, P. A. Chou, D. Minnen, S. Singh, N. Johnston, E. Agustsson, S. J. Hwang, and G. Toderici, "Nonlinear transform coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 2, pp. 339–353, 2020.

MiniWear: Minimizing Flash Wear via Hybrid Persistent Cache for Extended EF-SMR Lifetime

Chenlin Ma[1], Kaoyi Sun[1], Yuxuan Qi[1], Jiaxian Chen[1], Xiaochuan Zheng[1], Tianyu Wang[1] and Yi Wang[1,2]
1 *College of Computer Science and Software Engineering, Shenzhen University*
2 *State Key Laboratory of Radio Frequency Heterogeneous Integration*

***Abstract*—As the huge discrepancy between traffic and capacity persists, the lifetime of flash in EF-SMR systems faces a grave issue. EF-SMR systems combine NAND flash with Shingled Magnetic Recording (SMR) disks to achieve both low cost and high performance. However, previous research has primarily focused on issues such as *write amplification* and *tail-latency* in EF-SMR disks, overlooking the critical issue of flash lifetime. Studying the durability of EF-SMR systems is essential for developing future high-performance, low-cost storage solutions.**

This paper presents *MiniWear*, a hybrid persistent cache (PC) design aimed at extending the lifetime of EF-SMR systems. MiniWear adopts a hybrid medium persistent cache and proposes a customized scheduling strategy to reduce flash wear without impacting the EF-SMR system performance. At the hardware level, the hybrid PC of EF-SMR, composed of flash and SMR disk, is organized into Flash-PC and SMR-PC. At the software level, a fine-grained scheduling strategy is proposed to better manage PC resources. Additionally, we introduce a proactive balancing strategy to address PC resource idleness. Experimental results show that, compared to existing methods, MiniWear can reduce flash wear by up to 66.67%.

***Index Terms*—Lifetime, Embedded Flash, Shingled Magnetic Recording, Cache Management.**

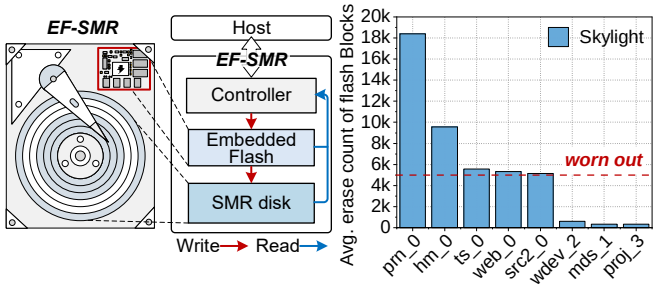
I. Introduction

Shingled Magnetic Recording (SMR) is a promising technique to promote disk density to more than 10 Tbits/in^2 by overlapping tracks one on another like shingling on a roof [1], [2]. As a side effect of the high density, SMR disks have inherent *write amplification* and *tail-latency* issues caused by Read-Modify-Write (RMW) operations. RMW operations can accommodate random writes but also drive the read/write head to move back and forth frequently, which significantly degrade the system's performance [3], [4]. Embedded Flash with SMR disk (EF-SMR) [5] integrates high-performance NAND flash memory, where random write data is first buffered in the flash before being written to the disk through RMW operations. However, the flash memory in EF-SMR systems faces significant write pressure, as shown in Fig. 1(a), and once the flash is occupied, it needs to be erased by a blocking background garbage collection (GC) process. The frequent GC processes lead to repeated wear of flash blocks, thereby introducing the system's ***lifetime*** issue.

We have conducted a preliminary experiment to demonstrate flash memory wear in the EF-SMR system. As Fig. 1(b) shows, the average erase count commonly exceeded 5k, reaching the upper limits for Multi-Level Cell (MLC) flash memory [6]. Considering the limited durability of NAND flash in erase cycles and the terabyte-scale write traffic in EF-SMR systems, our findings indicate that the *lifetime* of EF-SMR disks faces significant challenges.

Most previous works aim to resolve *write amplification* and *tail-latency* issues [7]–[15]. For example, PORE [12] implements a Partial Open-zone Eviction Strategy to mitigate SMR's write amplification by better balancing cache access frequencies. ROCO [13] uses a solid state drive (SSD) cache that absorbs random writes to reduce the write amplification of the Host-Aware SMR disk. HS-BAS [16] also utilizes SSD caching to diminish the collection costs associated with SMR's internal persistent cache. MAID-Q [14] targets to minimize tail latency by adopting a lightweight reinforcement learning model. MCM [15] reduces flash erasures by partitioning data into cold and hot segments. Unlike previous works, this paper proposes a *hybrid persistent cache* architecture designed to extend the *lifetime* of the EF-SMR system.

This paper introduces ***MiniWear***, a design for the EF-SMR system. Our main objective is to minimize flash wear via hybrid persistent cache, thereby extending the overall lifetime of the EF-SMR system. MiniWear incorporates hardware design and software optimization. The hardware design adopts a new persistent cache architecture with a hybrid medium, defining specific track zones of the SMR disk as SMR-PC. It also establishes a hybrid persistent cache mapping to distribute write stress to SMR-PC, effectively reducing the writing pressure faced by the flash. Based on the hardware design, the software optimization performs a fine-grained scheduling strategy of the persistent cache (PC), achieving efficient allocation of persistent cache resources. This ensures MiniWear can extend the lifetime of the EF-SMR system without degrading performance. In terms of system lifetime, MiniWear is compared with representative benchmark methods, showing a maximum lifetime extension of 66.67%.



(a) EF-SMR system. (b) Average erase count.

Fig. 1: Illustration of EF-SMR system.

The major contributions include:

- *Hybrid Persistent Cache Architecture:* A hybrid persistent cache designed for the EF-SMR system, along with a customized mapping that reduces the write traffic flowing through the flash memory.
- *Scheduling Strategy:* A scheduling strategy is proposed, which enables a fine-grained allocation of PC resources to extend the lifetime of the EF-SMR system further.
- *Experiments:* Comprehensive method validation through a comparative analysis with representative solutions.

The rest of this paper is organized as follows: Section II provides background and discusses the motivation of this paper. Section III presents our proposed MiniWear in detail. Section IV presents experimental results. Finally, Section V concludes this paper and discusses future work.

II. Background and Motivation

A. Shingled Magnetic Recording Disk

In the era of big data, there is an exponential growth in data generation [17], [18]. The emergence of the Internet of Things (IoT), Artificial Intelligence (AI), and other data-intensive technologies has accelerated this trend, leading to a continuous increase in the demand for storage space. Traditional storage solutions, such as Conventional Magnetic Recording (CMR) disks, as shown in Fig. 2(a), struggle to meet the increasing demands due to their limited storage density. This challenge has led to a need for more advanced storage solutions. Shingled Magnetic Recording (SMR) drives are one such solution, offering higher storage density by overlaying magnetic tracks on top of each other in a "shingled" manner, as illustrated in Fig. 2(b). In essence, while CMR maintains gaps between tracks to avoid interference, SMR maximizes space utilization by overlapping magnetic tracks to address the extensive storage demands [19].

B. Embedded Flash with SMR Disk

SMR overlaps track on previous tracks, like shingling the roofs to break through the areal density limitation of conventional hard disk drives (HDDs) [20]–[24]. However, this shingled-like structure incurs an accidental data loss on the tracks due to the write head is wider than the track width. Writing to a specific track can destroy the valid data stored on the adjacent tracks. To avoid accidental data loss, SMR requires an approach named Read-Modify-Write (RMW) operation to address random writes. The RMW operation will first read all valid data on the Band, and after modifying with the randomly written data, the modified data will be written back to the Band sequentially.

Since every RMW operation needs to overwrite the large-size SMR Band, it can lead to the write amplification issue and degrade the system performance. To achieve satisfactory run-time performance, several prior works [5], [12], [13], [25] integrate a NAND flash as a first-level persistent cache (PC) to form a faster EF-SMR disk. All random written data will be provisionally cached into the flash for high performance. When the flash is full, the valid data cached in the flash will be migrated to corresponding SMR Bands through a decent number of RMW operations. However, the capacity of the embedded flash is only about 1% of the total capacity of the EF-SMR disk [5]. Therefore, this indiscriminate data storing into flash puts severe wear pressure on the embedded flash, making it quickly worn out and unusable [26].

C. Motivation

To maintain low costs, the capacity of embedded flash memory in EF-SMR system is limited to 1% of the total capacity [5]. The existing EF-SMR system caches all random write requests to the flash, leading to durability issues, as shown in Fig. 3(a). Specifically, the limited flash capacity handles a large volume of write traffic, causing frequent programming and erasure of flash blocks to continuously satisfy write requests, thereby resulting in significant wear pressure. On the other hand, as each cell can store more bits by the multi-level cell technology, the durability of flash memory cells deteriorates [27]–[30]. For instance, the maximum P/E cycles of single-level cell (SLC) are about 10k, while those of triple-level cell (TLC) are decreased to 3k [6].

The limited flash capacity is the fundamental reason affecting the lifetime of the EF-SMR system, but increasing the flash capacity would raise costs. Therefore, there is an urgent need for a hybrid medium persistent cache in the EF-SMR system, as illustrated in Fig. 3(b). EF-SMR blurs the boundaries between flash and SMR disk by offering low latency and large capacity. Thus, the challenge of adding a hybrid medium persistent cache lies in how to extend the lifetime of the EF-SMR system as much as possible without sacrificing read/write performance. To address the challenge, this paper proposes MiniWear, which achieves this goal by designing a hybrid persistent cache and resource scheduling strategy.

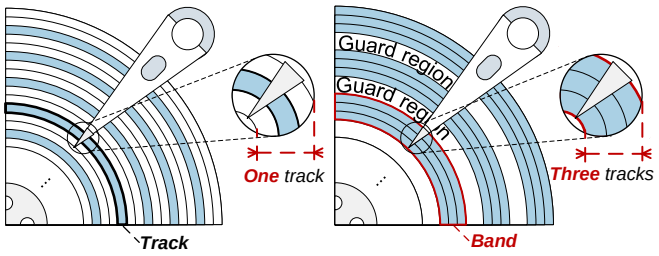


Fig. 2: SMR overlaps magnetic tracks within a Band to achieve higher storage density compared to CMR.

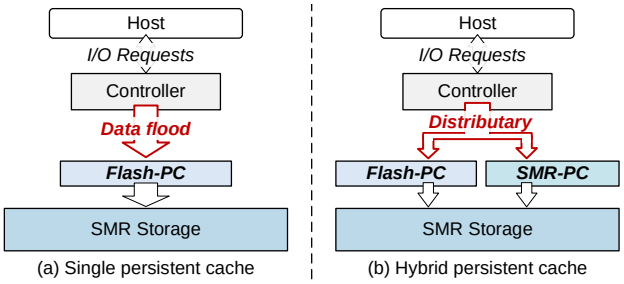


Fig. 3: Single PC utilizes only flash medium, while a hybrid PC incorporates both flash and SMR disk medium.

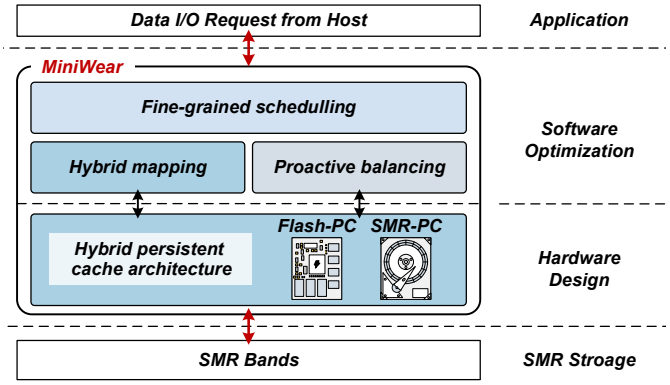


Fig. 4: The system overview of MiniWear.

III. MiniWear: A Hybrid Persistent Cache Design for Embedded Flash with SMR Disk

A. Overview of MiniWear

MiniWear adopts a hardware/software co-design, operating between applications and SMR storage devices to provide data scheduling, as shown in Fig. 4. The system comprises three main components: hybrid persistent cache architecture, fine-grained scheduling, and proactive balancing. The hybrid persistent cache provides hardware support. This foundation allows for a fine-grained scheduling strategy to ensure rational resource allocation. If PC resources are improperly allocated, a proactive balancing strategy adjusts and enhances the system. The hybrid address mapping and scheduling strategies are implemented through the PC manager. These three components are interconnected, forming the core of MiniWear. The hybrid persistent cache consists of Flash-PC and SMR-PC. Introducing SMR-PC helps reduce writing pressure on flash. The fine-grained scheduling strategy manages persistent cache allocation and reclamation, optimizing resource use and minimizing flash wear. The proactive balancing strategy recycles underutilized Flash-PC resources, improving utilization and preventing the cycle of frequent garbage collection due to shortages. The detailed designs of these components are as follows.

B. Hybrid Persistent Cache Architecture

In MiniWear, the hybrid persistent cache architecture consists of two parts: high-performance NAND flash (Flash-PC) and a few designated outer tracks on SMR disk platters (SMR-PC). In the following, we describe Flash-PC, SMR-PC, and the mapping strategy.

Flash-PC: Flash-PC consists of NAND flash memory, with the capacity of 1% of the EF-SMR system, which uses "Block" as the basic unit for scheduling, as shown in Fig. 5(a). MiniWear also adopts this approach, defining a flash block as the unit for Flash-PC scheduling, where one flash block represents one Flash-PC resource. However, the scheduling strategy for flash blocks in MiniWear differs from that of other works.

SMR-PC: We have innovatively integrated SMR-PC into EF-SMR, using a hardware-software co-design approach to extend the system's lifetime without compromising performance. SMR-PC consists of 1% of the outer tracks of the SMR disk, which supports random writes and offers better performance compared to the SMR Bands. As shown in Fig. 5(b). In MiniWear, SMR-PC is divided into equal-sized sectors, with each sector serving as a unit for SMR-PC scheduling. We designed a principle for SMR-PC that caches data using an append-write approach. The mapping relationships are recorded in a mapping table. For repeatedly written data, the append-write approach is used, and the mapping relationships are updated, with the original data automatically becoming invalid. This provides a high-performance caching mechanism to deal with random writes on the SMR disk. This design eliminates the write amplification, ensuring the superior performance of the EF-SMR system.

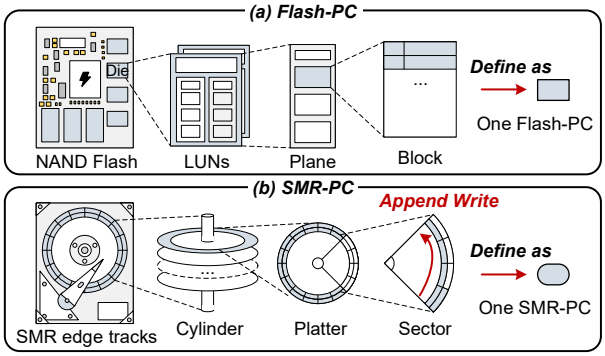


Fig. 5: Hardware design of hybrid persistent cache.

Hybrid Mapping: To effectively utilize the hybrid persistent cache, MiniWear introduces a hybrid mapping from the logic address to the address of PC or SMR. A page-level mapping table is used to record the mapping relationship. A logical page address may be mapped to SMR, Flash-PC or SMR-PC. MiniWear records whether the logical page address is mapped to the PC through the entry in the mapping table. When mapped to SMR, the physical address is the same as the logical address. Otherwise, the corresponding PC type and physical page address are recorded in the entry. Besides a page-level mapping table, there are two priority queues and bifurcated linked lists to handle mapping relationships efficiently. Priority queues record the starting addresses of available resources in Flash-PC and SMR-PC for quick lookup. A bifurcated linked

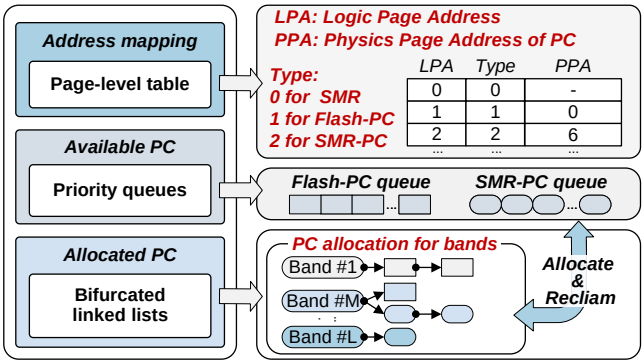


Fig. 6: Hybrid mapping.

list for each Band records the ownership status of both Flash-PC and SMR-PC resources. With the support of these data structures, MiniWear can efficiently traverse available PC and PC used by a specific Band when scheduling. The memory overhead caused by the mapping tables, priority queues, and linked lists will be analyzed in subsection III-E (Memory Overhead).

C. Fine-Grained Scheduling

The fine-grained scheduling strategy is designed for the hybrid persistent cache we defined. This strategy is executed by the PC controller, operating within the translation layer of the EF-SMR system. It controls the *allocation* and *reclamation* of PC to minimize wear on Flash-PC as much as possible, as shown in Algorithm 1.

Allocation: PC is allocated exclusively at the Band level, meaning that once PC is allocated to a Band, it is dedicated to that Band until reclaimed. To ensure the optimal performance of the EF-SMR system, the PC manager prioritizes the allocation of Flash-PC resources, as shown in Fig. 7(a). When Flash-PC resources are insufficient, SMR-PC resources are prioritized, as illustrated in Fig. 7(b). Resource allocation failure can occur in two scenarios: (1) when both Flash-PC and SMR-PC resources are exhausted, and (2) when the SMR Band reaches the maximum limit of SMR-PC resources. Since SMR-PC is less efficient than Flash-PC, its usage should be appropriately restricted. When allocation fails, the system performs garbage collection and reallocates resources after the collection process is complete.

Algorithm 1: Fine-Grained Scheduling

```
Input: LSN: Logical Sector Number and data of write request
Output: write result
// Allocation Strategy
LBN ← LSN / sector_of_Band
fpc_w ← Flash-PC_Writable[LBN]
spc_w ← SMR-PC_Writable[LBN]
if not fpc_w then
    fpc_w ← Distribute_Fpc()
end if
if not fpc_w and not spc_w then
    spc_w ← Distribute_Spc()
end if
// Reclamation Strategy
if not fpc_w and not spc_w then
    fpc_w ← Erase_and_Allocate(Find_Obsolete())
    if not fpc_w then
        fpc_w, spc_w ← Partial_Reclaim(LBN, longest_Band)
    end if
    if not fpc_w and not spc_w then
        Full_Reclaim(longest_Band)
    end if
end if
// Write
if fpc_w then
    Flash-PC_Write(DATA)
else
    SMR-PC_Write(DATA)
end if
```

Reclamation: Garbage collection tasks are also controlled by the PC manager, and we categorize garbage collection into three levels: obsolete flash block reclamation, Band-level partial reclamation, and Band-level full reclamation.

Obsolete flash block reclamation: Obsolete blocks refer to allocated flash blocks whose data has become entirely invalidated. Reclaiming obsolete flash blocks can rapidly complete the garbage collection process because it does not require data migration from persistent cache to SMR storage. This maximizes the performance of the EF-SMR system. During this operation, the manager only needs to locate an obsolete block and erase it to complete the current GC process, as shown in Fig. 7(c). *Band-level partial reclamation:* This operation reclaims all SMR-PCs owned by the Band and a portion of flash blocks (whose invalidation rate exceeds a specific threshold, which can be adjusted as needed; for example, set at 25%). The manager sequentially checks whether the Band currently being written to and the Band with the most flash resources meet the criteria for partial reclamation, and reclaims resources from the first Band to meet these criteria, as shown in Fig. 7(e) and (f). *Band-level full reclamation:* If none of the above conditions are met, the SMR Band with the most Flash-PC resources is chosen as the victim, and all its associated PC resources are completely reclaimed, as shown in Fig. 7(d).

Fine granularity is reflected in the individual and partial reclamation of flash blocks. Compared to complete reclamation, these two methods allow for more detailed consideration of the system's state, facilitating more nuanced GC processes. Obsolete blocks contain no valid data, and their reclamation does not involve data write-back operations, maintaining optimal system performance. Reclaiming a single invalid block significantly reduces the number of flash erasures compared to complete reclamation; partial reclamation aims to minimize the number of flash blocks erased during each GC process as much as possible.

D. Proactive Balancing

This strategy alleviates the issue of resource scarcity by proactively reclaiming underutilized flash resources, executed by the PC manager. In the EF-SMR system, there are underutilized flash blocks, also known as "zombie blocks", which are allocated on-demand to SMR Bands but are subsequently seldom used. This directly reduces the total amount of available Flash-PC, forcing more frequent garbage collection processes and indirectly impairing the system's lifetime.

$$T_i = \frac{Access_i}{NumFpc_i} \tag{1}$$

The strategy is implemented as follows: it is executed when the number of passive GC processes reaches a preset threshold (an adjustable empirical value). To identify the victim Band for proactive reclamation, we propose a metric T to estimate the hotness of Bands. This metric is determined by the relationship between the number of accesses to the Band during the interval between two proactive reclamations and the number of Flash-PC resources owned by the Band. Defined in Equation 1, T_i

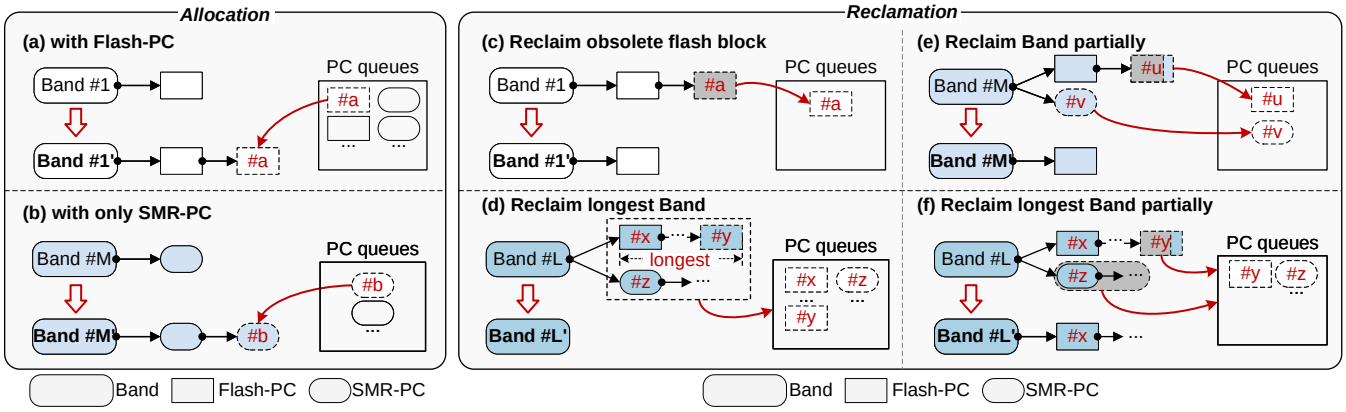


Fig. 7: Fine-grained scheduling.

represents the hotness of $Band_i$, $Access_i$ denotes the number of data accesses for $Band_i$, during the interval between two proactive reclamations, $NumFpc_i$ is the total number of Flash-PC resources owned by $Band_i$ (excluding Bands that do not have Flash-PC). Ultimately, the manager is responsible for reclaiming all PC from the Band with the lowest hotness.

E. Overhead Analysis

Memory Overhead: Flash-PC and SMR-PC each account for 1% of the total SMR Band capacity, resulting in a total persistent cache capacity of 40.96 GiB for a 2 TB SMR device. Specifically, the page-level mapping table requires 40.96 MiB (assuming a 4 KiB page size and 4 B page address), while the priority queue and linked list for Flash-PC consume 640 KiB and 1.28 MiB, respectively (with each linked list node occupying 8 B). The overhead for SMR-PC is similar to that of Flash-PC, and the total memory consumption remains acceptable for EF-SMR systems.

Computational Overhead: For fine-grained scheduling and proactive balancing strategies, we propose several algorithms. The time complexity of these algorithms is $\mathcal{O}(n)$, which does not impose an excessive burden on computational resources, making it acceptable for the system.

IV. Evaluation

A. Experimental Setup

Baseline. We compare MiniWear with Skylight [5], PORE [12], MCM [15] and Duchy [25]. Skylight uses static address mapping to integrate flash memory into SMR disks. PORE reduces I/O latency by limiting the reclamation scope and aggressively using flash resources. MCM differentiates between sequential hot/cold writes and random writes to improve latency and lifetime. Duchy proposes a write filtering scheme to enhance SSD endurance.

Performance Metrics. we compare MiniWear to Skylight [5], PORE [12], MCM [15] and Duchy [25] with the following performance metrics: (1) flash-based PC wear; (2) response time; (3) write amplification;

Parameter Setting. The specific configurations of the EF-SMR are set according to the observation from the realistic SMR disk [5] and Samsung NAND flash chip [31], which are depicted as follows: (1) The capacity of the persistent cache is 2% of the SMR disk capacity, with Flash-PC and SMR-PC each accounting for 1%. (2) The rotation speed of the disk is set as 7200 rpm, and the data transfer time is 0.03 ms; (3) The width of the write head is three times that of the read head (i.e., K=3); (4) The Band/Zone size is set as 40 MiB, totaling 256 in number; (5) The flash operation latencies for one page read, one page write and one block erase are 0.2 ms, 1.3 ms, and 1.5 ms, respectively. (6) The outermost circular tracks used as SMR-PC resources are logically divided into 128 sectors of equal capacity;

Datasets. We employed eight publicly accessible datasets characterized by varying sizes and access patterns, as elucidated in Table I. These datasets have been widely utilized for benchmarking the EF-SMR, as referenced in prior works [32].

B. Result and Discussion

Flash-based PC wear: As shown in Fig. 8(a), we evaluated the average wear count of the flash memory across different datasets and normalized the results. It can be observed that, on average, MiniWear reduces the flash-based PC wear count by 20.08%, 34.82%, 33.76%, and 32.93%, with maximum reductions of 30.67% (i.e., proj_4), 66.67% (i.e., web_2), 58.79% (i.e., web_2), and 62.52% (i.e., web_2), compared to Duchy, MCM, PORE, and Skylight, respectively. The reason for the optimization of flash memory lifetime is the introduction of the new medium (SMR-PC), which alleviates the write pressure on the flash and reduces its erasures. To further investigate the performance under more limited flash, we evaluated the average wear count of the flash memory with different flash capacities and normalized the results. As shown in Fig. 8(b), it can be observed that as the flash capacity decreases, the wear count increases for all methods. However, MiniWear shows a slower rate of increase, indicating that it has the capability to cope with extreme flash wear.

Response time: The average response time is a key indicator of system performance. We collected average response time data for quantitative analysis and normalized the data. As shown in Fig. 9(a), it can be observed that MiniWear reduces

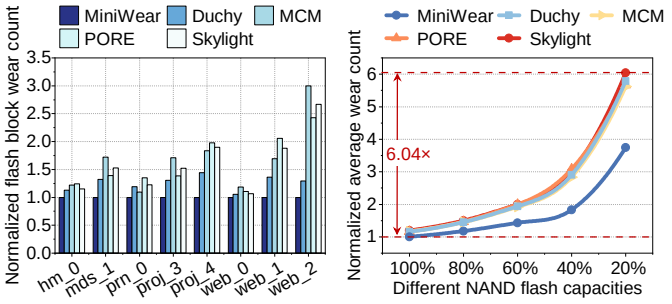
TABLE I: The characteristics of the traces.

Traces	Data Written (GB)	Data Read (GB)	Write Ratio	Update Ratio
hm_0	9.83	5.41	64.50%	97.72%
mds_1	0.45	5.80	7.12%	79.99%
prn_0	19.01	2.30	89.21%	94.52%
proj_3	0.44	8.12	5.18%	65.61%
proj_4	0.37	24.30	1.48%	88.17%
web_0	5.43	2.31	70.12%	99.21%
web_1	0.28	0.33	45.89%	91.69%
web_2	0.15	19.59	0.75%	52.49%

the average response time by 4.14%, 30.21%, 32.33%, and 53.21%, respectively, compared to Duchy, MCM, PORE, and Skylight. The maximum reduction rates reach 27.59% (hm_0), 64.08% (web_0), 67.55% (proj_3), and 83.48% (proj_3). This improvement is attributed to the fine-grained scheduling strategy, which leverages the architectural advantages of the hybrid persistent cache. Additionally, we analyzed the frequency distribution of response times for each request, as shown in Fig. 9(b), where the x-axis represents increasing response times. MiniWear exhibits the lowest frequency of high response times, indicating greater stability. This stability results from the ability of MiniWear to minimize unnecessary RMW operations during garbage collection.

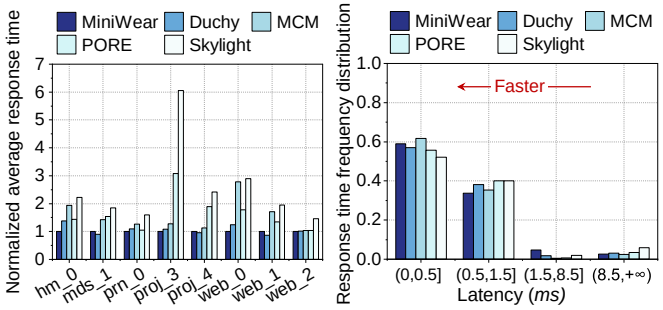
Observant readers may notice that in Fig. 9(a), for read-intensive workloads (e.g., mds_1), the average response time of MiniWear is slightly higher than that of Duchy. This phenomenon is attributed to the proactive balancing strategy, which involves reclaiming additional flash blocks. This approach can cause fluctuations in I/O latency for read-intensive workloads.

Write amplification: Write amplification is defined as the ratio of the actual amount of data written to the original input data volume. The optimization goal is to minimize this ratio to ensure system performance. We collected write amplification data from Duchy, MCM, PORE, and Skylight and normalized it to our proposed MiniWear. As shown in Fig. 10(a), MiniWear significantly reduces write amplification across all write-intensive and medium write volume traces compared to the baseline schemes. When handling read-intensive traces (e.g., proj_3), MiniWear exhibits slightly higher write amplification than Duchy. This is due to the proactive reclamation process involving RMW operations, which tend to increase write amplification. Additionally, we analyzed the actual data written for the same set of requests, with results summarized in



(a) Normalized flash block wear count. (b) Trends under different capacities.

Fig. 8: Flash-based PC wear.



(a) Normalized average response time. (b) I/O latency during runtime.

Fig. 9: Response time.

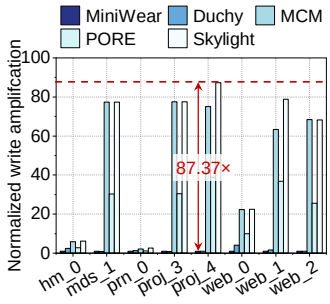
Table II. The data indicate that MiniWear effectively reduces the actual amount of data written, highlighting its capability to minimize unnecessary writes.

V. Conclusion

In this paper, we explore the issue of minimizing wear on NAND flash within EF-SMR disk storage systems. We introduce MiniWear, a novel hybrid persistent cache intermediate layer for the EF-SMR system. By introducing SMR-PC and optimizing address mapping and resource scheduling strategies, MiniWear can effectively reduce the number of flash erasures and extend the system's lifetime. Experimental results demonstrate its superior performance. In the future, we plan to optimize the Wear-leveling module to address "zombie blocks" with finer granularity. We also plan to investigate alternative approaches to partitioning SMR-PC so that MiniWear can perform better under various workloads.

Acknowledgements

The work described in this paper is supported in part by NSFC (U23B2040), in part by National Key Research and Development Program of China (2024YFB4505702), in part by Shenzhen Science and Technology Program (RCJC20221008092725019), in part by Guangdong Province Key Laboratory of Popular High Performance Computers (2017B030314073), in part by Guangdong Province Engineering Center of China-made High Performance Data Computing System and Shenzhen Key Laboratory of Service Computing and Applications. Tianyu Wang is the corresponding author.



(a) Normalized write amplification.

Fig. 10: Write amplifcation.

TABLE II: Actual written.

Method	Volume (GB)
MiniWear	914
Duchy	1,370
MCM	3,128
PORE	1,687
Skylight	3,595

REFERENCES

[1] D. Sun and Y. Chai, "SAC: A Co-Design Cache Algorithm for Emerging SMR-Based High-Density Disks," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 1047–1061.

[2] C.-F. Wu, M.-C. Yang, and Y.-H. Chang, "Improving Runtime Performance of Deduplication System with Host-Managed SMR Storage Drives," in *Design Automation Conference (DAC)*, 2018, pp. 1–6.

[3] C. Jin, W.-Y. Xi, Z.-Y. Ching, F. Huo, and C.-T. Lim, "HiSMRfs: A High Performance File System for Shingled Storage Array," in *Mass Storage Systems and Technologies (MSST)*, 2014, pp. 1–6.

[4] C. Ma, Z. Shen, L. Han, R. Chen, and Z. Shao, "FC: Built-in Flash Cache with Fast Cleaning for SMR Storage Systems," *Elsevier Journal of Systems Architecture (JSA)*, vol. 98, pp. 214–220, 2019.

[5] A. Aghayev, M. Shafaei, and P. Desnoyers, "Skylight—A Window on Shingled Disk Operation," in *Conference on File and Storage Technologies (FAST)*, 2015, pp. 1–16.

[6] J. Wu, J. Li, Z. Sha, Z. Cai, and J. Liao, "Adaptive Switch on Wear Leveling for Enhancing I/O Latency and Lifetime of High-Density SSDs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 11, pp. 4040–4051, 2022.

[7] C. Ma, Z. Zhou, Y. Wang, Y. Wang, and R. Mao, "MU-RMW: Minimizing Unnecessary RMW Operations in the Embedded Flash with SMR Disk," in *Design, Automation, and Test in Europe (DATE)*, 2022, pp. 490–495.

[8] X. Xie, L. Xiao, X. Ge, and Q. Li, "SMRC: An Endurable SSD Cache for Host-Aware Shingled Magnetic Recording Drives," *IEEE Access*, vol. 6, pp. 20 916–20 928, 2018.

[9] B. Zhang, M.-H. Yang, X. Xie, and D. H. Du, "Idler: I/O Workload Controlling for Better Responsiveness on Host-Aware Shingled Magnetic Recording Drives," *IEEE Transactions on Computers (TC)*, vol. 69, no. 6, pp. 777–788, 2020.

[10] C. Ma, Y. Wang, Z. Shen, and Z. Shao, "KFR: Optimal Cache Management with K-Framed Reclamation for Drive-Managed SMR Disks," in *Design Automation Conference (DAC)*, 2020, pp. 1–6.

[11] S. Huang, Q. Wei, D. Feng, J. Chen, and C. Chen, "Improving Flash-Based Disk Cache with Lazy Adaptive Replacement," *ACM Transactions on Storage (TOS)*, vol. 12, no. 2, pp. 1–24, 2016.

[12] C. Wang, D. Wang, Y. Chai, C. Wang, and D. Sun, "Larger Cheaper But Faster: SSD-SMR Hybrid Storage Boosted by a New SMR-Oriented Cache Framework," in *Conference on Mass Storage Systems and Technologies (MSST)*, 2017, pp. 1–16.

[13] W.-G. Liu, L.-F. Zeng, D. Feng, and K. B. Kent, "ROCO: Using a Solid State Drive Cache to Improve the Performance of a Host-Aware Shingled Magnetic Recording Drive," *Springer Journal of Computer Science and Technology (JCST)*, vol. 34, no. 1, pp. 61–76, 2019.

[14] C. Ma, Z. Zhou, Y. Wang, Y. Wang, and R. Mao, "MAID-Q: Minimizing Tail Latency in Embedded Flash with SMR Disk via Q-Learning Model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 11, pp. 3709–3720, 2022.

[15] Z. Chen, G. Wang, Z. Shi, Y. Guan, and T. Wang, "Region-Based Flash Caching with Joint Latency and Lifetime Optimization in Hybrid SMR Storage Systems," in *Design, Automation, and Test in Europe (DATE)*, 2023, pp. 1–6.

[16] W. Xiao, H. Dong, L. Ma, Z. Liu, and Q. Zhang, "HS-BAS: A Hybrid Storage System Based on Band Awareness of Shingled Write Disk," in *International Conference on Computer Design (ICCD)*, 2016, pp. 64–71.

[17] C.-F. Wu, M. Kuo, M.-C. Yang, and Y.-H. Chang, "Performance Enhancement of SMR-Based Deduplication Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 9, pp. 2835–2848, 2022.

[18] Y.-P. Liang, T.-Y. Chen, C.-H. Chi, H.-W. Wei, and W.-K. Shih, "Enabling a B+-Tree-Based Data Management Scheme for Key-Value Store over SMR-Based SSHD," in *Design Automation Conference (DAC)*, 2020, pp. 1–6.

[19] S. Zhou, E. Xu, H. Wu, Y. Du, J. Cui, W. Fu, C. Liu, Y. Wang, W. Wang, S. Sun, X. Wang, B. Feng, B. Zhu, X. Tong, W. Kong, L. Liu, Z. Wu, J. Wu, Q. Luo, and J. Wu, "SMRSTORE: A Storage Engine for Cloud Object Storage on HM-SMR Drives," in *Conference on File and Storage Technologies (FAST)*, 2023, pp. 395–408.

[20] S.-H. Chen, M.-C. Yang, Y.-H. Chang, and C.-F. Wu, "Enabling File-Oriented Fast Secure Deletion on Shingled Magnetic Recording Drives," in *Design Automation Conference (DAC)*, 2019, pp. 1–6.

[21] Y. Pan, Z. Jia, Z. Shen, B. Li, W. Chang, and Z. Shao, "Reinforcement Learning-Assisted Cache Cleaning to Mitigate Long-Tail Latency in DM-SMR," in *Design Automation Conference (DAC)*, 2021, pp. 103–108.

[22] L. Shi, J. Li, C. Jason Xue, and X. Zhou, "Hybrid Nonvolatile Disk Cache for Energy-Efficient and High-Performance Systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1, pp. 1–23, 2013.

[23] J. Wan, N. Zhao, Y. Zhu, J. Wang, Y. Mao, P. Chen, and C. Xie, "High Performance and High Capacity Hybrid Shingled-Recording Disk System," in *International Conference on Cluster Computing (CLUSTER)*, 2012, pp. 173–181.

[24] D. Luo, J. Wan, Y. Zhu, N. Zhao, F. Li, and C. Xie, "Design and Implementation of a Hybrid Shingled Write Disk System," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 4, pp. 1017–1029, 2016.

[25] X. Xie, T. Yang, Q. Li, D. Wei, and L. Xiao, "Duchy: Achieving Both SSD Durability and Controllable SMR Cleaning Overhead in Hybrid Storage Systems," in *International Conference on Parallel Processing (ICPP)*, 2018, pp. 1–9.

[26] J. Bhimani, Z. Yang, J. Yang, A. Maruf, N. Mi, R. Pandurangan, C. Choi, and V. Balakrishnan, "Automatic Stream Identification to Improve Flash Endurance in Data Centers," *ACM Transactions on Storage (TOS)*, vol. 18, no. 2, pp. 1–29, 2022.

[27] B. Li, C. Deng, J. Yang, D. Lilja, B. Yuan, and D. Du, "HAML-SSD: A Hardware Accelerated Hotness-Aware Machine Learning based SSD Management," in *International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[28] C. Gao, M. Ye, Q. Li, C. J. Xue, Y. Zhang, L. Shi, and J. Yang, "Constructing Large, Durable and Fast SSD System via Reprogramming 3D TLC Flash Memory," in *International Symposium on Microarchitecture (MICRO)*, 2019, pp. 493–505.

[29] Dharamjeet, Y.-S. Chen, T.-Y. Chen, Y.-H. Kuan, and Y.-H. Chang, "LLSM: A Lifetime-Aware Wear-Leveling for LSM-Tree on NAND Flash Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 11, pp. 3946–3956, 2022.

[30] S. Lee and J. Kim, "Improving Performance and Capacity of Flash Storage Devices by Exploiting Heterogeneity of MLC Flash Memory," *IEEE Transactions on Computers (TC)*, vol. 63, no. 10, pp. 2445–2458, 2014.

[31] Samsung-Corporation, "K9G4G08U0F 2GB MLC NAND Flash Datasheet." [Online]. Available: https://datasheetspdf.com/pdf/776555/Samsung/K9GAG08U0F/6

[32] "MSR Cambridge Block I/O Traces," 2024. [Online]. Available: http://iotta.cs.hmc.edu/traces/388

MARIO: A Superadditive Multi-Algorithm Interworking Optimization Framework for Analog Circuit Sizing

Wangzhen Li[1], Yuan Meng[1], Ruiyu Lyu[1], Changhao Yan[1], Keren Zhu[1], Zhaori Bi[1,*], Dian Zhou[1,2,†], Xuan Zeng[1,*]
[1] State Key Lab of ASIC & System, Microelectronics Department, Fudan University, Shanghai, China
[2] Department of Electrical Engineering, the University of Texas at Dallas, Richardson, Texas, USA

***Abstract*—Numeric optimization methods are widely utilized to tackle complex analog circuit sizing problems, where the challenges include expensive simulations, non-linearity, and high parameter dimensionality. However, the diverse characteristics exhibited by different circuits result in varied optimization landscapes, making it difficult to identify a single algorithm that consistently outperforms others across all problems. In this paper, we introduce a multi-algorithm interworking optimization framework, which achieves optimization superadditivity based on a pool of member algorithms and a powerful algorithm-interworking protocol. We propose a computing resource reallocation method, which employs multi-task Gaussian process regression and portfolio optimization techniques, leading to flexible and prudent online adaption of member algorithms. To efficiently utilize the computing resources for local exploitation, an evaluation data broadcast strategy enables cooperativeness across member algorithms. Besides, algorithms with different modeling overheads are integrated time-adaptively via an asynchronous parallelization mechanism. Comparative experiments against state-of-the-art algorithm-combining tools and optimization algorithms demonstrate the superiority of the proposed optimization framework.**

***Index Terms*—Analog circuit sizing, optimization framework, superadditivity, black-box optimization algorithm, algorithm-combining tool.**

I. Introduction

Analog integrated circuit automation remains in high demand to expedite the time-to-market delivery of industrial products, and the sizing stage is a bottleneck for human practitioners since both subtle expert experience and labor-intensive repetitive work are required. Therefore, various optimization algorithms [1]–[10] have been proposed to automate the analog circuit sizing, dealing with the challenges of **expensive simulations**, **non-linearity**, and **high parameter dimensionality**. However, while the diverse characteristics of analog circuits, including **topologies**, **specifications**, and **technologies**, result in varied optimization landscapes, existing works merely propose algorithms based on a limited number of case-study analog circuit instances, incurring poor optimization generalizability.

In this work, we recognize that it is difficult to identify or design a single algorithm that consistently outperforms others across diverse analog circuits. This difficulty is fatally incurred by the famous No Free Lunch (NFL) theorem [11], which indicates that the superiority of an optimization algorithm on several problems is always accompanied by inferiority on several other problems. Thus, given a pool of candidate algorithms, it is imperative to develop a universal optimization framework for sizing diverse analog circuits.

We introduce a critical concept of **superaddivity** as the destination of optimization frameworks, inspired by the active research in academic fields and landmark success in modern technologies, e.g. Gestalt psychology [12], coalition games [13], compute unified device architecture (CUDA) [14] and large language models (LLMs) [15]. Superadditivity means that the performance of a whole system should be qualitatively better than the sum of the performance of all parts, e.g. CUDA and LLMs respectively organize graphics processing kernels and self-attention modules as their parts, and then emerge with fundamentally superior computing capacity as systems. In the proposed optimization framework, we take member algorithms as parts and pursue the optimization superadditivity, which depends on two prerequisites as follows.

① Construct a heterogeneous pool of optimization algorithms.
② Develop an efficient and effective algorithm-interworking protocol.

For prerequisite ①, we extensively analyze and collect different algorithms, focusing on the optimization characteristics in terms of **sampling efficiency**, **convergency**, and **scalarbility**. Additionally, incorporating expensive simulations in the optimization loop, the ability of **batch sampling** is important for a candidate algorithm to speed up the optimization progression. In our framework, we integrate evolutionary algorithms (**EAs**), Bayesian optimization (**BO**), and mathematical derivative-free optimization (**DFO**), of which the complementary strengths and shortages are summarized in Table I.

TABLE I
The Strengths and Shortages of Different Algorithms

Algorithm	Sampling Efficiency	Convergency	Scalarbility
EAs [1]–[3], [16]	Low	Low	High
BO [7]–[10], [17]	High	Medium	Low
DFO [18], [19]	Medium	High	Medium

EAs, such as differential evolution (DE) [1], genetic algorithm (GA) [2], and particle swarm optimization (PSO) [3], are compatible with batch sampling and scalable for high-dimensional problems, since these algorithms evolve an elite population of visited design points for candidate suggestions, and the related heuristical operators are computing-inexpensive. However, EAs lack strategies to capture fine-grained features of the optimization landscapes and hence have weaknesses in local exploitation. **BO** balance the tradeoff between exploration and exploitation by optimizing acquisition functions in advance of candidate suggestions and thus achieving high sampling efficiency. The formulation of acquisition functions are based on online regression models. However, given high-dimensional design variables, online regression modeling is prone to underfit, incurring over-exploration in BO. Therefore, state-of-the-art BO algorithms, e.g. TuRBO [8], cVTS [9] utilize localization techniques to improve the scalability. As for **DFO**, Nelder-Simplex [18] and BOBYQA [19] are influential, and the latter is popular nowadays due to good sampling efficiency. BOBYQA also employs online models but differs from BO due to the focus on local convergence rather than global search, thereby suggesting candidate points by solving mathematical subproblems rather than optimizing acquisition functions. Since DFO is generally local and serial-limited, we can run several optimization procedures in parallel with different starting points to enable global exploration and batch sampling. This multi-starting point (MSP) strategy is widely adopted [20], although it is still limited by the random selection of starting points, which is inefficient for global exploration in heavily multi-modal optimization landscapes.

*Corresponding authors: {zhaori_bi, xzeng}@fudan.edu.cn
†Emeritus professor, the University of Texas at Dallas.
This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711901, partly by National Natural Science Foundation of China (NSFC) research projects 92473206, 62304052, 62474050, 92473207, 62474051.

For prerequisite ②, algorithm-combining protocols, including passive algorithm selection, active algorithm selection, chaining, and splitting, have been actively studied by recent works. **Passive algorithm selection** [21], [22] picks one algorithm from a pre-built decision tree at the beginning of the optimization run, according to the given dimensionality of parameters, budget of evaluations, amount of computing resources, etc. **Active algorithm selection** [23]–[25] means starting multiple optimization algorithms in parallel and selecting one of them to continue with a decision-maker after a certain number of evaluations. The decision-maker can be realized either with heuristical strategies or pre-trained neural networks. **Chaining** [26] involves running several algorithms sequentially, with the following ones initialized using the evaluation information from previous ones. **Splitting** [27] partitions the design variables, and then each group of them is optimized by its own optimizer. Further, active algorithm selection, chaining, and splitting can be integrated into **hybrid passive algorithm selection**, It has been implemented by Nevergrad [27], making the related ABBO [28] and Shiwa [29] the state-of-the-art algorithm-combining tools.

The main drawback of existing algorithm-combining protocols is the reliance on massive prior experimental experience, which is a burden for the developers and the combined optimization performance is prone to deteriorate in tackling previously unseen problems. Therefore, considering extensibility and generalizability, active algorithm selection with a heuristical decision-maker is more **flexible**. Nevertheless, current heuristical active decision-makers e.g. choosing one algorithm once based on the best observation [23], take speculative risks to produce satisfactory optimization results. Targeting a more **prudent** algorithm selection framework, it is necessary to refine the switching process across member algorithms and employ online modeling techniques to make the adaption of algorithms more comprehensive. In addition, it is also crucial to make the member algorithms **cooperative**, which means enabling heterogeneous algorithms to utilize the evaluation data from each other, hence efficient strategies for evaluation information extraction and transfer are dispensable. Besides, for analog circuits with simulation time limited to several seconds, BO and DFO algorithms suffer from non-neglectable modeling overheads and get distinctly slower than evolutionary algorithms. Therefore, a powerful algorithm-interworking protocol should be **time-adaptive** across different algorithms.

To address the aforementioned challenges of flexibility, prudence, cooperativeness, and time-adaptivity for achieving optimization superadditivity, we propose an innovative multi-algorithm interworking optimization framework MARIO. Overall, we extensively collect a pool of heterogeneous optimization algorithms, run them in parallel, and develop a powerful algorithm-interworking protocol for joint optimization. Our main contributions are summarized as follows.

1) We propose an active computing resource reallocation method, achieving **flexible** and **prudent** online adaption of member algorithms. We employ a customized multi-task Gaussian process to model the optimization potential of member algorithms and utilize portfolio optimization to reallocate the computing resources, improving the optimization desirability periodically.
2) We propose an efficient evaluation data broadcast strategy, making the member algorithms **cooperative** for joint optimization. With the accumulation of evaluated design points, we incrementally partition the design space into Voronoi nests and filter out informative data to broadcast. This strategy is especially beneficial to local algorithms since they can be effectively reinitialized to exploit promising design regions.
3) We propose a practical asynchronous parallelization mechanism, leading to **time-adaptive** algorithm integration in MARIO. If some algorithms exhibit non-negligible modeling overheads, this mechanism adapts faster algorithms to evaluate more design points and get more competitive to gain computing resources, thus assuring reasonable optimization performance.

With fair utilization of computing resources, MARIO achieves more than $3.83\times$ optimization speedup on a test-function benchmark suite and more than $2.59\times$ optimization speedup on an analog-circuit benchmark suite against state-of-the-art black-box optimization algorithms and algorithm-combining tools, based on the overall optimization performance measured by data profiles (see Sect. II-B). Python implementation and experimental results of MARIO can be found at https://github.com/RogerLee0038/MARIO.

II. BACKGROUND

A. Problem Formulation

The automatic sizing for a given analog circuit can be formulated as an optimization problem p as 1.

$$\begin{aligned} p: \quad & \arg\min_{\boldsymbol{x}\in\mathbb{R}^{d_p}} f_p(\boldsymbol{x}) \\ s.t. \quad & \boldsymbol{l}_p \leq \boldsymbol{x} \leq \boldsymbol{u}_p, \end{aligned} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_p}$ represents the d_p-dimensional vector of design variables, f_p represents the objective function, and $\boldsymbol{l}_p \sim \boldsymbol{u}_p$ represent the box constraints of the design space, which are defined by the experience of circuit designers and the rules of process design kits.

Considering constraints on circuit specifications, the objective function is formulated as a weighted sum as $f_p(\boldsymbol{x}) = s_{p,0}(\boldsymbol{x}) + \lambda_{p,1}c_{p,1}(\boldsymbol{x}) + \cdots + \lambda_{p,n_s}c_{p,n_s}(\boldsymbol{x})$, where $s_{p,0}$ is the target specification to be minimized, $c_{p,1:n_s} = max(0, s_{p,1:n_s})$ are constraint violation of other specifications, and $\lambda_{p,1:n_s}$ are weight multipliers.

B. Performance Measurement

Given a set $\mathcal{P}$ of diverse analog circuit sizing problems and a set $\mathcal{M}$ of optimizers, we measure the overall performance of each optimizer $m \in \mathcal{M}$ by the data profile [30] as 2.

$$D_m(a) = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} T_{p,m}(a), \tag{2}$$

where the input variable a represents the number of objective evaluations or the optimization time. $T_{p,m}(a)$ measures the optimization performance of m on a single problem p as 3.

$$T_{p,m}(a) = \frac{f_p(\boldsymbol{x}_0^m) - f_p(\boldsymbol{x}_a^m)}{f_p(\boldsymbol{x}_0^m) - f_p^L}, \tag{3}$$

where $\boldsymbol{x}_0^m$ is the optimization starting point of m, $\boldsymbol{x}_a^m$ is the a-th point suggested by m, and f_p^L is the smallest objective value obtained by any $m \in \mathcal{M}$ within a given optimization budget $\hat{a_p}$. This formulation of $T_{p,m}(a)$ measures whether the problem p can be well optimized by m, based on the objective value reduction $f_p(\boldsymbol{x}_0) - f_p(\boldsymbol{x}_a)$ relative to the best possible reduction $f_p(\boldsymbol{x}_0) - f_p^L$.

In addition, considering different problem dimensionality d_p, the optimization budget $\hat{\alpha_p}$ is usually normalized as $\hat{\kappa}\cdot(d_p+1)$, where $\hat{\kappa}$ represents a budget factor that is common for all $p \in \mathcal{P}$. Then, $D_m(a)$ can be normalized as $D_m(\kappa)$, with $\kappa = \alpha/(d_p+1) \in [0, \hat{\kappa}]$.

C. Portfolio Optimization

For prudent reallocation of computing resources across member algorithms in MARIO, we treat the parallel workers for circuit simulations as funds and different optimization algorithms as investments, then utilize portfolio optimization [31] to periodically make

reallocation decisions. Specifically, given a collection of possible investments with a vector $\boldsymbol{r}$ of expected returns and a matrix $\boldsymbol{G}$ of covariances, an investor can construct a portfolio by putting a fraction $\boldsymbol{\omega}_i$ of the available funds into investment i. Then, the desirability of the portfolio is measured and optimized as 4.

$$\begin{aligned} &\arg\max_{\boldsymbol{\omega}} \ \boldsymbol{\omega}^T \boldsymbol{r} - \eta \boldsymbol{\omega}^T \boldsymbol{G} \boldsymbol{\omega} \\ &s.t. \ \Sigma_{i=1} \boldsymbol{\omega}_i = 1, \ \boldsymbol{\omega} \geq \boldsymbol{0}, \end{aligned} \tag{4}$$

where η is a risk aversion parameter that can be small for conservative investors and large for more daring investors.

D. Multi-Task Gaussian Process

The return vector $\boldsymbol{r}$ and covariance matrix $\boldsymbol{G}$ in portfolio optimization are constructed as introduced in Sect. III-A, with the aid of a multi-task Gaussian process (MTGP) [32]. Given a d-dimensional input space $\mathbb{R}^d$ and M tasks, an MTGP builds a regression model based on an observation dataset $\mathcal{D} = \{(\boldsymbol{\kappa}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{\kappa}_{1:N} \in \mathbb{R}^d$ are training features and $\boldsymbol{y}_{1:N} \in \mathbb{R}^M$ are response labels. We adopt $\boldsymbol{\kappa}$ instead of $\boldsymbol{x}$ here because the MTGP in MARIO will take the normalized number of evaluations rather than the design variables of analog circuits as the training features. Specifically, an MTGP models the covariances over input features and the similarities over tasks with a kernel function $k^{\kappa}(\boldsymbol{\kappa}_i, \boldsymbol{\kappa}_j)$ and a positive semi-definite matrix $\boldsymbol{K}^{tas}$, respectively. After training, the MTGP predicts an arbitrary unobserved response at the input location $\boldsymbol{\kappa}^*$ of the task l with the posterior mean and variance as 5.

$$\mu_l(\boldsymbol{\kappa}^*) = (\boldsymbol{k}^{\kappa}(\boldsymbol{\kappa}^*) \otimes \boldsymbol{K}_l^{tas})^T (\boldsymbol{K}^{\kappa} \otimes \boldsymbol{K}^{tas} + \boldsymbol{I}_N \otimes \boldsymbol{\Lambda})^{-1} \boldsymbol{y} \tag{5a}$$

$$\begin{aligned} \sigma_l^2(\boldsymbol{\kappa}^*) =& k^{\kappa}(\boldsymbol{\kappa}^*, \boldsymbol{\kappa}^*) \cdot \boldsymbol{K}_{l,l}^{tas} - (\boldsymbol{k}^{\kappa}(\boldsymbol{\kappa}^*) \otimes \boldsymbol{K}_l^{tas})^T \cdot \\ & (\boldsymbol{K}^{\kappa} \otimes \boldsymbol{K}^{tas} + \boldsymbol{I}_N \otimes \boldsymbol{\Lambda})^{-1} (\boldsymbol{k}^{\kappa}(\boldsymbol{\kappa}^*) \otimes \boldsymbol{K}_l^{tas}), \end{aligned} \tag{5b}$$

where $\boldsymbol{y}$ is a vector containing the MN response values in the dataset ordered as $(\boldsymbol{y}_{11}, \cdots, \boldsymbol{y}_{1M}, \cdots, \boldsymbol{y}_{N1}, \cdots, \boldsymbol{y}_{NM})$, $\otimes$ denotes the Kronecker product, $\boldsymbol{K}_l^{tas}$ and $\boldsymbol{K}_{l,l}^{tas}$ select the l-th row and the (l,l) element from $\boldsymbol{K}^{tas}$ respectively, $\boldsymbol{K}^{\kappa}$ represents the matrix of covariances between all pairs of $\boldsymbol{\kappa}_{1:N}$, $\boldsymbol{k}^{\kappa}(\boldsymbol{\kappa}^*)$ is a vector of covariances between $\boldsymbol{\kappa}^*$ and $\boldsymbol{\kappa}_{1:N}$, and $\boldsymbol{\Lambda}$ is an $M \times M$ matrix with the (l,l) element as σ_l^2, i.e. the noise variance for the l-th task.

The hyperparameters in a MTGP include the parameters $\boldsymbol{\theta}_{\kappa}$ in the kernel function k^{κ}, the entries of matrix $\boldsymbol{K}^{tas}$ and the noise variances $\sigma_{1:M}^2$ in $\boldsymbol{\Lambda}$. They can be trained by maximizing the marginal likelihood according to the distribution $\boldsymbol{y}|\boldsymbol{\kappa}_{1:N} \sim \mathcal{N}(\boldsymbol{0}, (\boldsymbol{K}^{\kappa} \otimes \boldsymbol{K}^{tas} + \boldsymbol{I}_N \otimes \boldsymbol{\Lambda}))$, which is updated along with the dataset $\mathcal{D}$.

III. PROPOSED METHODS

In this section, we introduce the key techniques of MARIO in detail, including active computing resource reallocation, evaluation data broadcast, and asynchronous parallelization mechanism. The overall workflow of MARIO is depicted in Fig. 1

A. Active Computing Resource Reallocation

Since the circuit simulations are expensive, we measure the computing resources involved in MARIO by the number of parallel simulation workers. These workers parallelize the simulations of the candidate design points, which are suggested batch by batch from member algorithms. Then, the active computing resource reallocation is implemented via a customized MTGP and portfolio optimization.

We employ an MTGP enhanced by exponentially decaying kernels to periodically extrapolate the optimization curves and quantify the optimization potential of member algorithms. Specifically, the current-best-curve progression of member algorithms is regarded as

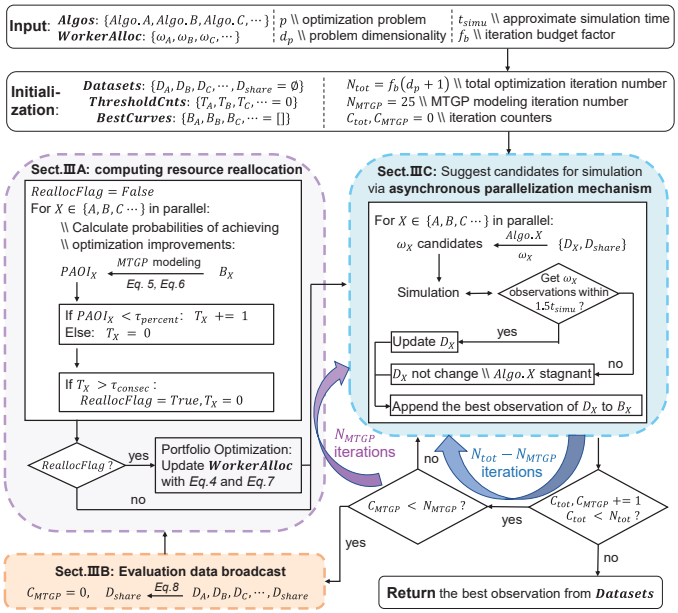


Fig. 1. The overall workflow of MARIO. Given a pool of $\boldsymbol{Algos}$, MARIO runs them in parallel for N_{tot} iterations, while the evaluation data broadcast and the computing resource reallocation are conducted in N_{MTGP} iterations. For broadcast, MARIO maintains a dataset D_{share}, which contributes to the candidate suggestions of all member algorithms. For reallocation, the probability of achieving optimization improvements (PAOI) is calculated for member algorithms by modeling the $\boldsymbol{BestCurves}$. If an algorithm exhibits a poor (less than $\tau_{percent}$) PAOI for a consecutive number τ_{consec} of modeling rounds, the $\boldsymbol{WorkerAlloc}$ is updated by portfolio optimization.

different modeling tasks, with the normalized number of evaluations as the common input variable and the objective values as the responses. After modeling, the MTGP will predict the extrapolated objective values of different member algorithms through a posterior normal distribution. Based on the posterior mean μ and variance σ, the probability of achieving optimization improvement (PAOI) of each member algorithm is calculated as $\mathrm{PAOI}(\kappa^*; \mathcal{D}) = \Phi\left(\frac{v - \mu(\kappa^*; \mathcal{D})}{\sigma(\kappa^*; \mathcal{D})}\right)$, where v represents the current-best observed objective value, κ^* is a prediction point and Φ is the standard normal cumulative distribution. Note that in addition to μ and σ, we also obtain a matrix $\boldsymbol{C}$ of extrapolated objective covariances across member algorithms, which will be used later in 7 for portfolio optimization.

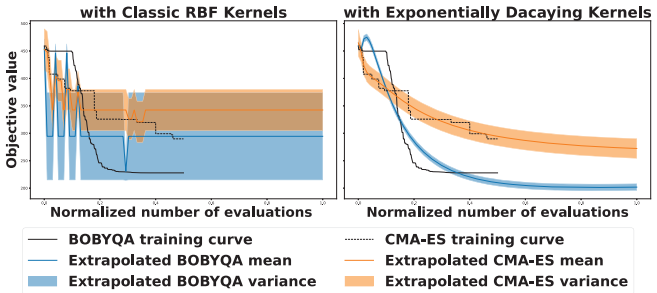


Fig. 2. Extrapolation performance with two different kernels for two different algorithms while optimizing a 10D Ackley test function.

The ability to extrapolate optimization curves is achieved with exponentially decaying kernels [33] as 6.

$$k(\kappa, \kappa') = \int_0^{\infty} e^{-\xi(\kappa+\kappa')} \frac{\beta^{\alpha}}{\Gamma(\alpha)} \xi^{\alpha-1} e^{-\xi\beta} d\xi = \frac{\beta^{\alpha}}{(\kappa + \kappa' + \beta)^{\alpha}}, \tag{6}$$

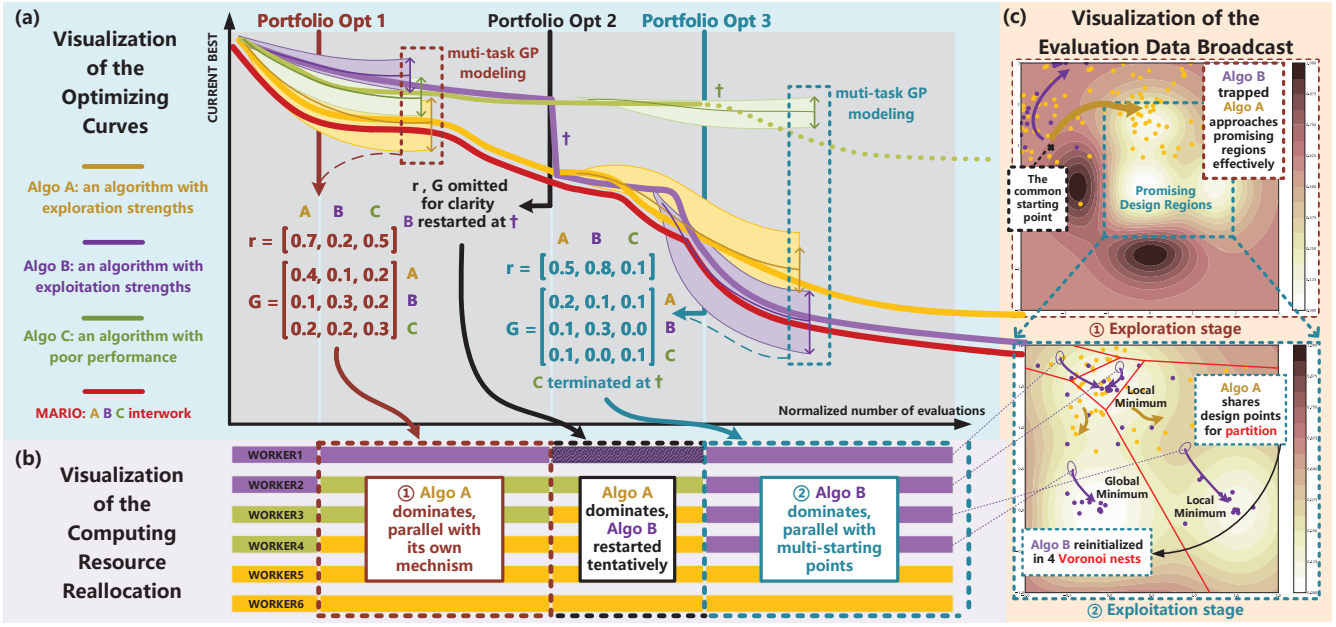


Fig. 3. An illustration of the optimization progression of MARIO. **(a)** A visualization of three optimizing curves and three periods of computing resource reallocation. The red curve results from combining the three heterogeneous member algorithms A, B, and C within MARIO. The $\boldsymbol{r}$ and $\boldsymbol{G}$ are modeled by the multi-task Gaussian process and the parallel workers are reallocated by portfolio optimization. **(b)** A visualization of the evolution of parallel computing resources, where algorithm A dominates optimization after the first two periods of portfolio optimization, and algorithm B dominates in turn after the third period of portfolio optimization. **(c)** A visualization of the evaluation data broadcast strategy. In the early exploration stage, algorithm B is trapped in a local minimum, while algorithm A can approach the promising design regions. In the later exploitation stage, the design space is partitioned into Voronoi nests based on the shared evaluated design points. Then algorithm B is enabled to reinitialize its optimization procedures, exploiting 4 promising regions in parallel.

which is constructed by integrating an infinite set of exponentially decaying functions of the form $f(\kappa,\xi)=e^{-\xi\kappa}$, with κ representing the 1-dimensional input variable. $\frac{\beta^{\alpha}}{\Gamma(\alpha)}\xi^{\alpha-1}e^{-\xi\beta}$ is a non-negative mixing measure weighting the integration range with parameters $\alpha,\beta>0$, and $\Gamma(\cdot)$ is the gamma function. As shown in Fig. 2, exponentially decaying kernels produce more informative and smoother extrapolation performance than classic RBF kernels. Additionally, since the MTGP is based on the 1-dimensional input variable and is efficiently accelerated in MARIO through black-box matrix-matrix multiplication techniques [34], its computational cost is neglectable in comparison to the overhead of circuit simulations.

If a member algorithm exhibits a poor POAI for a certain consecutive number of MTGP-modeling periods, the computing resource reallocation is activated, based on the portfolio optimization as formulated in 4, where the $\boldsymbol{r}$ and $\boldsymbol{G}$ are constructed as 7.

$$\boldsymbol{r} = max(\boldsymbol{0}, \boldsymbol{\mu} - v) \circ \boldsymbol{\omega}_{cur}^{\circ(-1)} \tag{7a}$$

$$\boldsymbol{G} = \boldsymbol{C} \circ (\boldsymbol{\omega}_{cur}^{T}\boldsymbol{\omega}_{cur})^{\circ(-\frac{1}{2})}, \tag{7b}$$

where $\boldsymbol{\mu}$ represents the vector of extrapolated objective means, $\boldsymbol{C}$ represents the matrix of extrapolated objective covariances, $\boldsymbol{\omega}_{cur}$ represents the current allocation of computing resources, $\circ$ represents the Hadamard product and $^{\circ()}$ represents the element-wise power.

We additionally introduce two constraints for portfolio optimization. The first constraint is $\sum_i |\boldsymbol{\omega}_i - \boldsymbol{\omega}_{cur}| \leq \Omega$, which restricts the total turnover Ω of computing resources, considering that the optimization potential of member algorithms is approximately in proportion to the computing resources within a relative-small linearized range. The second constraint is $\boldsymbol{\omega}_i \leq \omega_{max}$ for local algorithms, which avoids the waste of computing resources due to meaningless local exploitation. ω_{max} represents the number of promising design regions, which is derived systematically later in Sect.III-B,

Besides, we adopt a dynamic risk aversion parameter η for portfolio optimization in 4 to make the computing resource reallocation more comprehensive. In the early optimization stages, η is relatively large to encourage member algorithms with high optimization variance. It is reasonable since these algorithms generally serve for global exploration and need to gobble up computing resources to reach promising design regions quickly. Then η gets smaller during the optimization progression, gradually preferring local algorithms that can steadily exploit the promising design regions.

A visualization of three periods of computing resource reallocation among three member algorithms is presented in Fig. 3(a) and 3(b).

B. Evaluation Data Broadcast

To empower optimization superadditivity in MARIO, it is critical to make the member algorithms cooperate with each other. Therefore, we maintain a parallel broadcast process to periodically collect the evaluated design points from different member algorithms and extract informative evaluation data to transfer.

To assure the efficiency of evaluation data broadcast, it is necessary to prune uncompetitive observations because these redundant data heavily retard and even mislead the progression of optimization. Hence, with the accumulation of evaluated design points, We incrementally partition the design space into Voronoi nests and filter out information of promising design regions to broadcast.

The Voronoi nests are constructed by clustering the design points, and the clustering is based on the objective-weighted distance as 8.

$$dist(\boldsymbol{x_i}, \boldsymbol{x_j}) = w\sqrt{\frac{d_s^2}{d_p}} + (1-w)\sqrt{d_o^2}, \tag{8}$$

where $d_s = ||\boldsymbol{x}_i - \boldsymbol{x}_j||$ is the spatial distance, $d_o = \sqrt{(f_i - f_j)^2}$ is the objective distance, d_p is the dimensionality of the design

variables, and w is a weight parameter. Consequently, the design points that are spatially close and qualitatively similar will be clustered into the same Voronoi nest. Note that the total number of Voronoi nests can be controlled by limiting the maximum number of design points inside a nest [9]. After partition, we score each Voronoi nest according to the best-achieved objective value, and the involved design point that achieves this value is regarded as the representative design point of the nest. Based on the scores, we filter out competitive Voronoi nests, which exactly represent the promising design regions.

While global algorithms mainly utilize representative design points to promote exploration, local algorithms can further make use of the information of promising design regions to reinitialize themselves, which leads to adaptive and effective local exploitation, significantly contributing to the superadditivity of MARIO. The reinitialization requires no extra sampling overheads since the broadcasted design observations can be directly reused. Moreover, considering that the serial-limited local algorithms are parallelized in MARIO through the MSP strategy, the computing resource allocation for these algorithms can be restricted according to the number of promising design regions, which accounts for ω_{max} in Sect.III-A.

Fig. 3(c) presents a visualization of the evaluation data broadcast.

C. Asynchronous Parallelization Mechanism

Heterogeneous algorithms naturally exhibit different computational overhead for the candidate-point suggestions, and thus there will be optimization blocks in MARIO if the evaluated design points are synchronously collected. To make MARIO time-adaptive, we design an asynchronous parallelization mechanism, which is important in practice, especially for sizing the module-level analog circuits with simulation time limited to several seconds.

The proposed mechanism employs a timeout parameter t_{out} for the poll of parallelly evaluated design points across member algorithms. If no points are provided by a slow algorithm within t_{out}, this algorithm will be considered stagnant and other faster algorithms will become relatively more competitive in gaining computing resources. In MARIO, we set $t_{out} = 1.5t_{simu}$, where t_{simu} is the allowed simulation time for a given target circuit. This setting indicates that the collection of evaluated design points from a member algorithm will be postponed once this algorithm exhibits a modeling overhead 1.5 times more than the simulation overhead.

IV. Experimental Results

In this section, we show the superiority of MARIO against existing optimizers. The baseline optimizers include state-of-the-art BO algorithms for analog circuit sizing (**TuRBO** [8] and **cVTS** [9]), widely-used evolutionary and DFO algorithms (**CMA-ES** [16], **DE** [27], **TBPSA** [35] and **BOBYQA** [36]), and state-of-the-art algorithm-combining tools (**ABBO** [28] and **Shiwa** [29]).

For MARIO, we realize an instance as **MARIO-1**, combining TuRBO, cVTS, CMA-ES, DE, and BOBYQA w.l.o.g. In MARIO-1, TuRBO, CMA-ES, and DE are expected to act as global algorithms, while cVTS and BOBYQA are expected to act as local algorithms. Note that we remove the original global exploration strategy in cVTS because the evaluation data broadcast strategy in MARIO is a more informative alternative and the main advantage of VTS lies in the boundary-restricted local exploitation. In addition, we realize a counterpart of MARIO-1 as **MARIO-1se**, where the computing resource reallocation is disabled for ablation studies.

For fair utilization of computing resources, the number of parallel workers for all optimizers is 20. Specifically, all optimizers suggest 20 design points for parallel simulations in each iteration, either based on the MSP strategy (for serial-limited optimizers such as cVTS and BOBYQA) or batch sampling (for parallel-enabled optimizers). Besides, we unify the constraint handling for all optimizers via the weighted-sum objective function as introduced in Sect.II-A.

We conduct optimization experiments on two benchmark suites and run all optimizations 10 times to measure the variances of random fluctuations. The random seeds are provided in our code repository for reproducibility. All experiments are conducted on a Linux server with Intel Xeon 6230 CPUs and 128GB RAM.

A. Test-Function Experiments

The test-function benchmark suite includes 17 widely-used functions i.e. Ackley, CosineMixture, Exponential, Griewank, Levy-Montalvo1, LevyMontalvo2, Neumaier3, Paviani, Rastrigin, Rosenbrock, Salomon, Schwefel, Sinusoidal, Michalewicz, SineEnvelope, EggHolder, and Rana. All these functions have scalable dimensions, and we choose the dimensionality of $\{20, 40\}$ for demonstration.

Fig. 4 presents the data-profile accumulating curves of different optimizers on the test-function benchmark suite, with the x-axis representing the normalized number of evaluations. This normalization indicates that given a test function with dimensionality d_p, the maximum number of evaluations is limited to $20 \cdot 50 \cdot d_p$, where 20 is the number of parallel workers and 50 is a fixed budget factor. MARIO-1 and MARIO-1se significantly outperform other baseline optimizers, while MARIO-1 is steadily better than MARIO-1se. In comparison with the best baseline optimizer TuRBO, MARIO-1 achieves 3.83× speedup for the normalized number of evaluations.

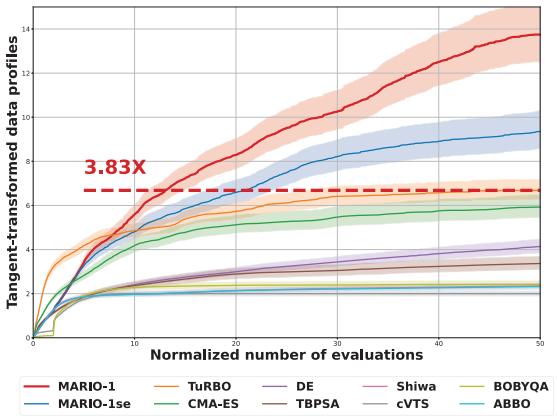


Fig. 4. The data-profile accumulating curves of different optimizers on the test-function benchmark suite. The translucent area shows a variance of 0.1σ.

A breakdown of the best-achieved data profiles of different optimizers on the test-function benchmark suite is shown in Fig. 5(a), MARIO-1 achieves competitive optimization performance across all test functions, leading to the highest data profile of 0.954, which is 5.41%∼35.32% better than the data profiles of baseline optimizers.

B. Analog-Circuit Experiments

Table II summarizes the experiment setup for 15 analog circuits, with a part of them open-sourced [37] and available in our repository. The evaluation-number budget and optimization-time budget for analog circuits are $20{\cdot}20{\cdot}(d_p{+}1)$ and $t_{simu}{\cdot}20{\cdot}(d_p{+}1)$ respectively, with the use of 20 parallel workers and a budget factor of 20.

Fig. 6 presents the data-profile accumulating curves of different optimizers on the analog-circuit benchmark suite, MARIO-1 and TuRBO significantly outperform other optimizers, while MARIO-1 achieves 2.59× and 1.91× speedup compared to TuRBO for the normalized number of evaluations and optimization time, respectively.

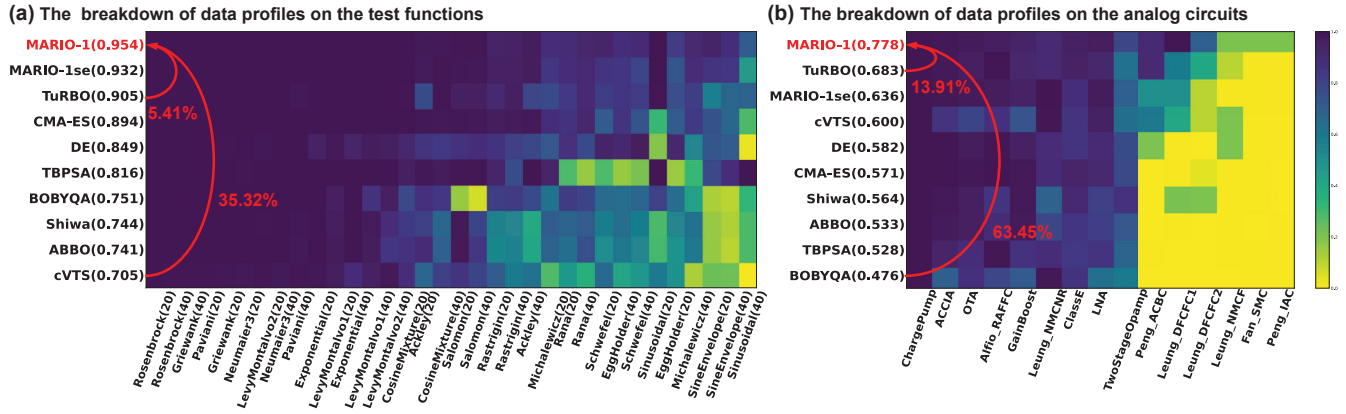


Fig. 5. The breakdown of the best-achieved data profiles for different optimizers on the test-function and analog-circuit benchmark suites. The x-axes represent different optimization problems p, the y-axis represents different optimizers m, and the grids visualize the values of $T_{p.m}$, with the darker color indicating better optimization performance. The row-wise average of $T_{p,m}$ is the data profiles, which are presented following each optimizer on two benchmark suites.

TABLE II
THE EXPERIMENT SETUP FOR ANALOG CIRCUITS

Name	d_p, t_{simu}	Specification Description	Evaluation Budget	Time Budget
Charge Pump	36, 10s	current deviation in 27 PVT corners	14800	2.06h
ACCIA	38, 10s	input referred noise, power, gbw, etc.	15600	2.17h
OTA	59, 5s	gain, ugf, etc.	24000	1.67h
ClassE	13, 10s	out power, dc power	5600	0.78h
LNA	19, 60s	NF, IIP3, etc.	8000	6.67h
GainBoost	20, 10s	gain, slew rate, etc.	8400	1.17h
TwoStageOpamp	10, 10s	loop gain, pm, etc.	4400	0.61h
8 circuits from AnalogGym [37]	23∼33, 20s	10 specs, leading to FOM_{AMP}	9600∼ 13600	2.67h∼ 3.78h

In Fig. 6, no optimizer distinctly outperforms others in early exploration stages, thus the superiority of MARIO-1 mainly results from efficient local exploitation later. Evolutionary algorithms CMA-ES, DE, and TBPSA can not exploit promising design regions effectively, and neither can ABBO and Shiwa, since these algorithm-combining tools also select evolution algorithms in most cases. On the other hand, BOBYQA is talented in local exploitation but fails to reach promising design regions from the beginning. Besides, cVTS and BOBYQA are uncompetitive in our experiments because these two algorithms are serial-limited and merely utilize parallel computing resources with the MSP strategy, which is inefficient for exploring the heavily multi-modal optimization landscapes of analog circuits. The challenge of multi-modality also accounts for the poor performance of MARIO-1se, since it can't reallocate more computing resources for parallel local exploitation. Additionally, the modeling overheads of TuRBO and cVTS are non-neglectable, which is inferred by comparing the optimization performance along with time and the performance along with evaluations. The optimization efficiency of TuRBO is also limited by its restart strategy, which constructs new trust regions from scratch once the old ones are converged.

A breakdown of the best-achieved data profiles of different optimizers on the analog-circuit benchmark suite is shown in Fig. 5(b). The optimization superadditivity of MARIO-1 is remarkable, especially for difficult cases in the right part. The data profile of MARIO-1 is 13.91%∼63.45% better than those of baseline optimizers.

V. CONCLUSIONS

In this work, we propose a superadditive multi-algorithm interworking optimization framework MARIO for analog circuit sizing.

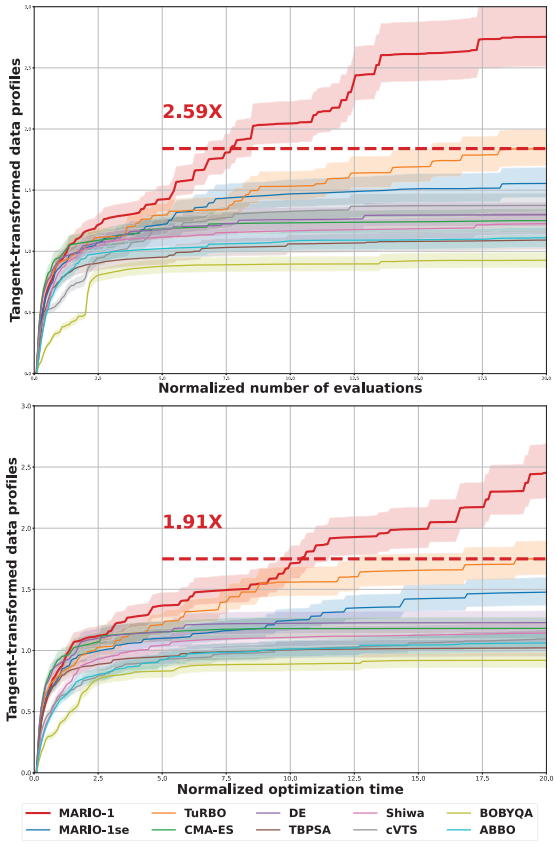


Fig. 6. The data-profile accumulating curves of different optimizers on the analog-circuit benchmark suite. The translucent area shows a variance of 0.1σ.

To demonstrate the superiority of MORIO, we implement MARIO-1, which combines 5 heterogeneous optimization algorithms. In comparison with the baseline optimizers for sizing 15 diverse analog circuits, MARIO-1 achieves more than $2.59\times$ evaluation-number speedup, and more than $1.91\times$ optimization-time speedup.

For future research, the application scenarios of MARIO are not necessarily limited to analog circuit sizing or single objective optimization, and we can further unleash the capacity of MARIO with multi-machine parallelism. Moreover, MARIO is extending-friendly for any new algorithms, with promising optimization superadditivity.

REFERENCES

[1] B. Liu, Y. Wang, Z. Yu, L. Liu, M. Li, Z. Wang, J. Lu, and F. V. Fernández, "Analog circuit optimization system based on hybrid evolutionary algorithms," *Integration*, vol. 42, no. 2, pp. 137–148, 2009.

[2] M. Taherzadeh-Sani, R. Lotfi, H. Zare-Hoseini, and O. Shoaei, "Design optimization of analog integrated circuits using simulation-based genetic algorithm," in *Signals, Circuits and Systems, 2003. SCS 2003. International Symposium on*, vol. 1, 2003, pp. 73–76 vol.1.

[3] R. A. Vural and T. Yildirim, "Analog circuit sizing via swarm intelligence," *AEU-International journal of electronics and communications*, vol. 66, no. 9, pp. 732–740, 2012.

[4] B. Liu, D. Zhao, P. Reynaert, and G. G. Gielen, "Gaspad: A general and efficient mm-wave integrated circuit synthesis method based on surrogate model assisted evolutionary algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 2, pp. 169–182, 2014.

[5] A. F. Budak, M. Gandara, W. Shi, D. Z. Pan, N. Sun, and B. Liu, "An efficient analog circuit sizing method based on machine learning assisted global optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1209–1221, 2021.

[6] Y. Yang, H. Zhu, Z. Bi, C. Yan, D. Zhou, Y. Su, and X. Zeng, "Smart-msp: A self-adaptive multiple starting point optimization approach for analog circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 3, pp. 531–544, 2017.

[7] W. Lyu, P. Xue, F. Yang, C. Yan, Z. Hong, X. Zeng, and D. Zhou, "An efficient bayesian optimization approach for automated optimization of analog circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 6, pp. 1954–1967, 2017.

[8] K. Touloupas, N. Chouridis, and P. P. Sotiriadis, "Local bayesian optimization for analog circuit sizing," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1237–1242.

[9] A. Zhao, X. Wang, Z. Lin, Z. Bi, X. Li, C. Yan, F. Yang, L. Shang, D. Zhou, and X. Zeng, "cvts: A constrained voronoi tree search method for high dimensional analog circuit synthesis," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[10] T. Gu, W. Li, A. Zhao, Z. Bi, X. Li, F. Yang, C. Yan, W. Hu, D. Zhou, T. Cui *et al.*, "Bbgp-sdfo: Batch bayesian and gaussian process enhanced subspace derivative free optimization for high-dimensional analog circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[11] S. P. Adam, S.-A. N. Alexandropoulos, P. M. Pardalos, and M. N. Vrahatis, "No free lunch theorem: A review," *Approximation and optimization: Algorithms, complexity and applications*, pp. 57–82, 2019.

[12] W. Köhler, "Gestalt psychology," *Psychologische forschung*, vol. 31, no. 1, pp. XVIII–XXX, 1967.

[13] W. Saad, Z. Han, M. Debbah, A. Hjorungnes, and T. Basar, "Coalitional game theory for communication networks," *Ieee signal processing magazine*, vol. 26, no. 5, pp. 77–97, 2009.

[14] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "Gpgpu processing in cuda architecture," *arXiv preprint arXiv:1202.4347*, 2012.

[15] Z. Wang, S. Mao, W. Wu, T. Ge, F. Wei, and H. Ji, "Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration," *arXiv preprint arXiv:2307.05300*, 2023.

[16] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[17] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable global optimization via local bayesian optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[18] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[19] M. J. Powell *et al.*, "The bobyqa algorithm for bound constrained optimization without derivatives," *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, vol. 26, 2009.

[20] L. Qian, Z. Bi, D. Zhou, and X. Zeng, "Automated technology migration methodology for mixed-signal circuit based on multistart optimization framework," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2595–2605, 2014.

[21] N. Baskiotis and M. Sebag, "C4. 5 competence map: a phase transition-inspired approach," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 10.

[22] J. Rapin, P. Dorval, J. Pondard, N. Vasilache, M.-L. Cauwet, C. Couprie, and O. Teytaud, "Openly revisiting derivative-free optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 267–268.

[23] T. Weise, Z. Wu, and M. Wagner, "An improved generic bet-and-run strategy for speeding up stochastic local search," *arXiv preprint arXiv:1806.08984*, 2018.

[24] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[25] P. Kerschke and H. Trautmann, "Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning," *Evolutionary computation*, vol. 27, no. 1, pp. 99–127, 2019.

[26] D. Molina, M. Lozano, and F. Herrera, "Memetic algorithm with local search chaining for continuous optimization problems: A scalability test," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*. IEEE, 2009, pp. 1068–1073.

[27] J. Rapin and O. Teytaud, "Nevergrad-a gradient-free optimization platform," 2018.

[28] L. Meunier, H. Rakotoarison, P. K. Wong, B. Roziere, J. Rapin, O. Teytaud, A. Moreau, and C. Doerr, "Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 3, pp. 490–500, 2021.

[29] J. Liu, A. Moreau, M. Preuss, J. Rapin, B. Roziere, F. Teytaud, and O. Teytaud, "Versatile black-box optimization," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 620–628.

[30] J. J. Moré and S. M. Wild, "Benchmarking derivative-free optimization algorithms," *SIAM Journal on Optimization*, vol. 20, no. 1, pp. 172–191, 2009.

[31] F. J. Fabozzi, H. M. Markowitz, and F. Gupta, "Portfolio selection," *Handbook of finance*, vol. 2, 2008.

[32] E. V. Bonilla, K. Chai, and C. Williams, "Multi-task gaussian process prediction," *Advances in neural information processing systems*, vol. 20, 2007.

[33] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw bayesian optimization," *arXiv preprint arXiv:1406.3896*, 2014.

[34] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in neural information processing systems*, vol. 31, 2018.

[35] M. Oquab, J. Rapin, O. Teytaud, and T. Cazenave, "Parallel noisy optimization in front of simulators: Optimism, pessimism, repetitions, population control," in *Workshop Data-driven Optimization and Applications at CEC*, 2019.

[36] C. Cartis, L. Roberts, and O. Sheridan-Methven, "Escaping local minima with local derivative-free methods: a numerical investigation," *Optimization*, vol. 71, no. 8, pp. 2343–2373, 2022.

[37] J. Li, H. Zhi, R. Lyu, W. Li, Z. Bi, K. Zhu, Y. Zeng, W. Shan, C. Yan, F. Yang *et al.*, "Analoggym: An open and practical testing suite for analog circuit synthesis," *arXiv preprint arXiv:2409.08534*, 2024.

Move Less, Retrieve Fast: A Retrieval-in-Memory Architecture for Language Models

Jiaxian Chen[1], Yuxuan Qi[1], Jianan Yuan[1], Kaoyi Sun[1], Tianyu Wang[1], Chenlin Ma[1] and Yi Wang[1,2]
1. College of Computer Science and Software Engineering, Shenzhen University
2. State Key Laboratory of Radio Frequency Heterogeneous Integration

***Abstract*—Retrieval-augmented language models (RALMs) have attracted widespread attention for addressing the limitations of traditional large language models. However, challenges involved in retrieval, including substantial data movement and irregular access patterns, seriously impact the efficiency and deployment of RALMs. The emerging 3D-stacked processing-in-memory (PIM) architecture, characterized by its high memory bandwidth and near-data computing capabilities, presents a promising solution for efficient retrieval. To support large-scale retrieval in RALMs, the PIM architecture should be carefully designed with joint software and hardware optimization.**

This paper presents *Rimast*, a retrieval-in-memory architecture for fast retrieval in RALMs. The objective is to minimize data movement and improve overall performance through hardware-software co-design. At the hardware level, a hierarchical PIM architecture with a retrieval-in-memory dataflow is designed to reduce unnecessary data transfer. At the software level, skew-free data mapping and adaptive offloading strategies are proposed to address the irregular access patterns associated with retrieval in RALMs. We demonstrate the effectiveness of the proposed Rimast using extensive experiments. The experimental results demonstrate that Rimast effectively reduces data movement, achieving average speedups of 273×, 55×, and 2.41× over CPUs, GPUs, and prior art accelerators, respectively.

***Index Terms*—Retrieval, RALM, Processing-in-Memory**

I. Introduction

Retrieval-augmented language models (RALMs) [5], [22] have been successfully deployed in various application domains. RALMs combine non-parametric external storage with internal parameters, and this hybrid approach provides better performance than traditional large language models [9], [10]. RALMs typically employ a large-scale external database with billions of embeddings to enhance language models. RALMs suffer from low efficiency caused by large-scale retrieval from external database, making them 12× slower than traditional language models (LMs) [27].

Fig. 1(a) illustrates the workflow of a typical RALM, which consists of three major steps: (i) *BERT*; (ii) *Retrieval*; and (iii) *LM*. Upon receiving an input context, it is transformed into chunk embeddings using BERT [8]. These embeddings query a key-value database to retrieve the most similar text chunks, which are then integrated into LMs to generate high-quality responses. Although RALMs greatly enhance response accuracy, *retrieval* (the second step) accounts for 93.8% of the processing time in the RALMs pipeline, as illustrated in Fig. 1(b), highlighting the significant potential for optimization.

Efficient retrieval approaches mainly comprise graph-based, hashing-based, and compression-based algorithms. Although

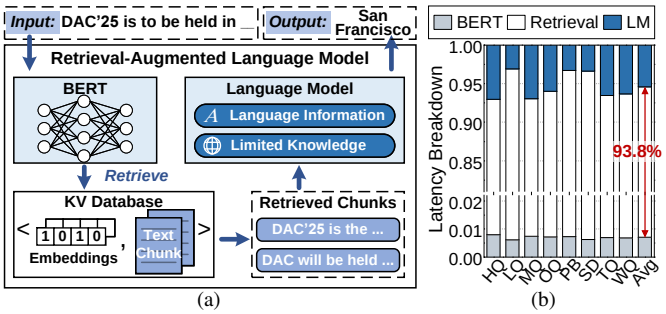


Fig. 1: (a) RALM workflow. (b) Latency breakdown.

graph-based [17] and hashing-based [26] algorithms perform well with high accuracy during retrieval, they often require a high memory footprint of up to several terabytes of memory in RALMs [11]. In contrast, compression-based retrieval algorithms can achieve high performance and low memory footprint [13]. Many RALMs adopt compression-based algorithms for large-scale retrieval [5], [20], [23], [25]. However, retrieval in RALMs still involves substantial data movement that algorithm optimization can not fully address. Numerous embeddings in external databases will be transferred from memory to processors, which causes significant performance degradation and energy consumption.

The processing-in-memory (PIM) architecture [6], [7], [21], [28] provides a promising solution for RALMs. The advanced 3D memory [14] stacks multiple DRAM dies on a base die, offering significant capacity and bandwidth for RALMs. Furthermore, computing units can be placed in DRAM dies to enjoy *in-memory retrieval*, effectively reducing data movement. However, processing retrieval efficiently via the 3D-stacked PIM architecture is non-trivial. First, the existing centralized processing dataflow is not suitable for in-memory retrieval. Second, irregular access issues in RALMs lead to imbalanced workloads, limiting the benefits of PIM architectures.

This paper proposes *Rimast*, a retrieval-in-memory architecture for fast billion-scale retrieval in RALMs. The proposed Rimast minimizes data movement and improves the overall performance for retrieval using a hardware-software co-design approach. At the hardware level, a hierarchical PIM architecture with a retrieval-in-memory dataflow is designed to minimize unnecessary data transfer. At the software level, a skew-free data mapping strategy is presented to achieve fine-grained data partitioning and even data distribution. An adaptive offloading strategy is further proposed to reallocate tasks at runtime, eliminating imbalanced workloads. We demonstrate the effec-

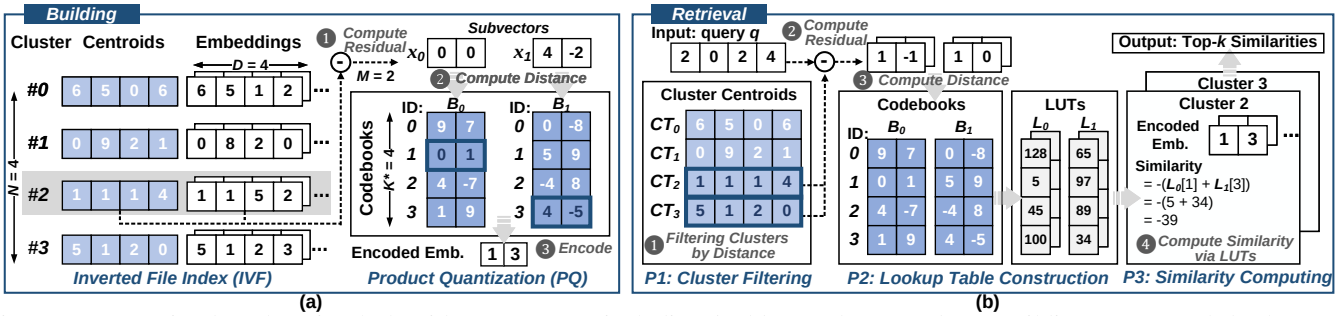


Fig. 2: Compression-based retrieval algorithm [13], [24], including *building* and *retrieval*. (a) Building an external database. (b) Three phases of retrieval.

tiveness of Rimast using extensive experiments. Experimental results show that Rimast outperforms existing representative schemes, substantially reducing processing latency and energy consumption.

The main contributions of this work include:

- *PIM for RALM Retrieval*: To the best of our knowledge, this is the first PIM-based retrieval accelerator to address the data movement and the irregular access issues in RALMs.
- *Architecture Design*: A retrieval-in-memory architecture with customized dataflow is designed to minimize off-chip data movement.
- *Software Optimization*: A skew-free data mapping strategy and an adaptive offloading strategy are proposed to address the imbalanced workloads caused by irregular accesses.

The rest of this paper is organized as follows: Section II illustrates the background and motivation, Section III details the proposed techniques, and Section IV presents and discusses the experimental results. Finally, Section V concludes the paper.

II. Background

A. Large-Scale Embedding-Based Retrieval

Embedding-based retrieval aims to identify the top-k embeddings in a database that are most similar to a given query embedding. With the L2 (Euclidean) distance, the similarity between the query q and an embedding y is defined as:

$$S(q,y) = -\|q-y\|_2^2 = -\|q\|^2 - \|y\|^2 + 2\langle q,y\rangle \quad (1)$$

An exact search over a large-scale database is computationally expensive, often requiring significant time and memory.

To alleviate the overhead, compression-based retrieval algorithms are widely adopted for their high performance and low memory footprint. Compression-based algorithms primarily adopt inverted file (IVF) index and product quantization (PQ) [13]. The IVF index divides embeddings into N clusters, each represented by a centroid. It significantly reduces computation by restricting similarity computations to the selected clusters. PQ reduces storage and computation costs by mapping high-dimensional embeddings to lower-dimensional spaces. It achieves this by dividing a D-dimensional embedding into M subvectors, each of which is quantized using a learned codebook with $K*$ codewords. An example of building a database using IVF and PQ is illustrated in Fig. 2(a). The retrieval process

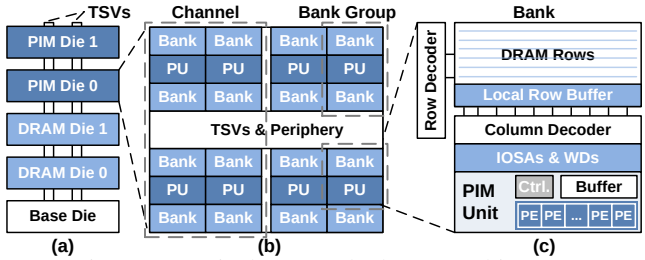


Fig. 3: A typical 3D stacked PIM architecture.

consists of three main phases: (1) *cluster filtering*, (2) *lookup table (LUT) construction*, and (3) *similarity computing*. An example of retrieval is depicted in Fig. 2(b). Given a query embedding $q = \|2\ 0\ 2\ 4\|$, the process starts by selecting candidate clusters with the most similar centroids (i.e., CT_2 and CT_3). Next, LUTs are built to reduce redundant computations, as each encoded embedding may comprise repeated codewords. For instance, the residual embedding of q relative to CT_2 is split into two segments and compared with codewords in B_0 and B_1 to form L_0 and L_1. Finally, similarities between q and encoded embeddings are efficiently computed via LUTs.

B. 3D-Stacked Processing-in-Memory Architecture

Fig. 3 illustrates a typical processing-in-memory (PIM) architecture utilizing 3D stacked memory [14], where multiple DRAM dies are stacked on top of a base die using through-silicon vias (TSVs). DRAM dies equipped with PIM capabilities are termed PIM dies. As shown in Fig. 3(b), each PIM die contains multiple memory channels, with each channel comprising bank groups and each bank group comprising banks. Fig.3(c) shows the structure of a DRAM bank with a PIM unit, which includes SRAM buffers and processing elements for near-bank computing. Each bank contains DRAM cells, a global row buffer of I/O sense amplifiers (IOSAs), address decoders, and write drivers (WDs). DRAM cells are arranged in rows and columns. Before accessing memory, the relevant row should be activated. As only one row per bank can be active at a time, accessing two different rows consecutively causes a row buffer conflict, resulting in low bandwidth utilization.

C. Motivation

*1) **Data Movement Issue**:* Compression-based retrieval involves substantial movement of embeddings from memory to processors, making it memory-bound. We analyze the memory bandwidth requirement of retrieval with the well-established

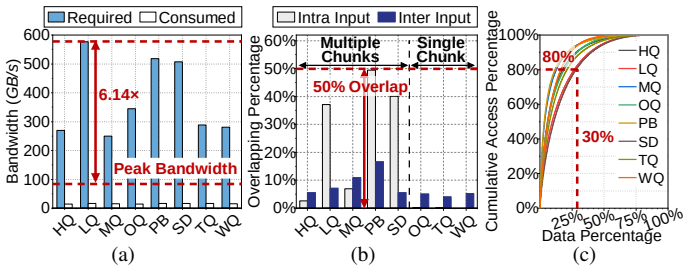


Fig. 4: (a) Up to 6.14× bandwidth demand. (b) Access locality pattern with up to 50% data overlap. (c) Skewed access pattern with 30% data contributing 80% access.

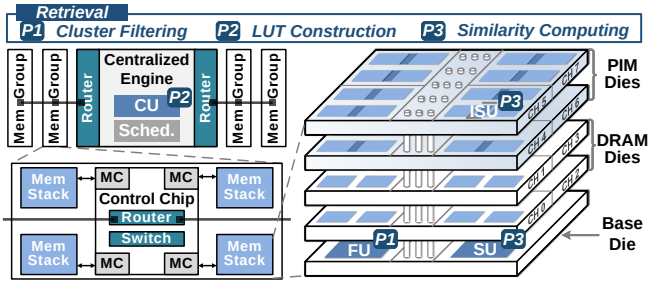


Fig. 5: Architecture design overview. Dedicated compute units (CU, FU, SU, and ISU) are added for retrieval.

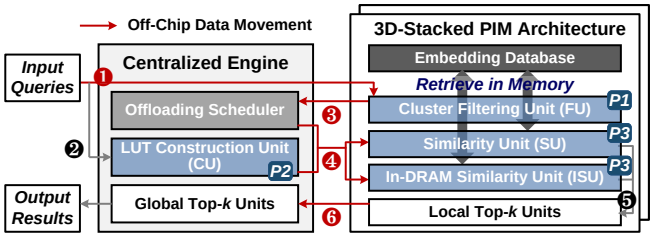


Fig. 6: Retrieval-in-memory dataflow.

library Faiss [12]. Fig. 4(a) indicates that retrieval performance is constrained by limited memory bandwidth. To match the throughput of inference, the system requires a higher bandwidth, ranging from 2.65× to 6.14× the peak memory bandwidth of a four-channel system. Since the memory-bounded similarity computing phase mainly involves addition operations, it can be efficiently processed in memory with low hardware overhead.

*2) **Irregular Access Issue**:* The workloads of RALMs exhibit irregular access patterns. First, RALMs exhibit strong temporal locality. A long input is divided into chunks, generating several query chunk embeddings. Adjacent chunk embeddings within an input often access the same clusters, with up to 50% overlap (Fig. 4(b)). Second, retrieval has a skewed access pattern. 30% of the clusters account for over 80% of the accesses, while others receive minimal attention (Fig. 4(c)). These access patterns cause imbalanced workloads, especially in PIM architectures with in-place data processing.

Motivated by the above-mentioned challenges, we focus on exploiting PIM architectures for retrieval via a hardware and software co-design approach.

III. Rimast: An Retrieval-in-Memory Architecture for RALMs

Rimast is a HW/SW co-design architecture tailored for retrieval in RALMs. At the hardware level, a hierarchical PIM architecture with a customized dataflow is designed to exploit in-memory retrieval. At the software level, Rimast applies two strategies to address irregular access patterns. First, *skew-free data mapping* is proposed to distribute embeddings evenly, avoiding imbalanced workloads caused by the skewed pattern. Second, *adaptive offloading* is proposed to reduce latency by rescheduling tasks with temporal locality patterns.

A. Hardware Architecture Design

1) Overview: The overall hardware architecture of Rimast is demonstrated in Fig. 5. Rimast comprises a centralized engine (CE) and multiple memory groups, where a chain topology [18] is adopted to connect them. Each memory group integrates multiple memory stacks and a control chip. The control chip contains memory controllers (MCs), an internal switch, and a router for off-chip communication. The CE constructs LUTs via a *LUT construction unit (CU)*, schedules retrieval tasks, and aggregates similarity results, while PIM-enabled memories process in-memory retrieval. Rimast adds dedicated logic to multiple levels to accelerate retrieval jointly. At the base die level, the *cluster filtering unit (FU)* is customized to select candidate clusters, and the *similarity unit (SU)* is designed to compute similarities for overlapping accessed clusters. At the PIM die level, the *in-DRAM similarity unit (ISU)* is integrated into each memory bank group to enjoy near-bank processing.

2) Retrieval-in-Memory Dataflow: A retrieval-in-memory dataflow is proposed to perform similarity computations directly in memory, transferring only reduced intermediate results rather than the entire database. Fig. 6 presents the proposed dataflow. Once queries come, query embeddings are broadcast to PIM-enabled memories (step ❶) and simultaneously sent to CU (step ❷). Then, FUs within the PIM memories filter candidate clusters, sending the filtered results to the offloading scheduler in the CE (step ❸). Next, the shared partial LUTs are broadcast to all PIM memories, and the scheduler distributes tasks to the PIM-enabled memories (step ❹), enabling similarity computing. Finally, local top-k units within each memory stack reduce these similarities (step ❺), and the global top-k unit in the CE aggregates the reduced similarities (step ❻), completing the retrieval process. Throughout this process, only steps ❶ ❸ ❹ ❻ involve minimal off-chip data movement, with lightweight intermediate data and scheduling information being transferred.

The retrieval dataflow is carefully designed with a two-step LUT construction method to reduce data movement between the CE and PIM architectures while avoiding adding extra multipliers to PIM units. The CE handles shared portions of LUTs, and PIM units perform partial accumulations. Given a query q, a cluster centroid y_C, and a valid codebook keyword combination y_B, the L2 distance is decomposed in Equation (2). Here, $term\ 1$ is computed during cluster filtering, and $term\ 2$ is precomputed and stored as it is independent of q. The CE computes $term\ 3$ to form partial LUTs. The complete LUTs are built on demand by accumulating these terms during the similarity computing phase.

$$\|q - y_C - y_B\|_2^2 = \underbrace{\|q - y_C\|_2^2}_{term\ 1} + \underbrace{\|y_B\|^2 + 2\langle y_C, y_B\rangle}_{term\ 2} - \underbrace{2\langle q, y_B\rangle}_{term\ 3} \quad (2)$$

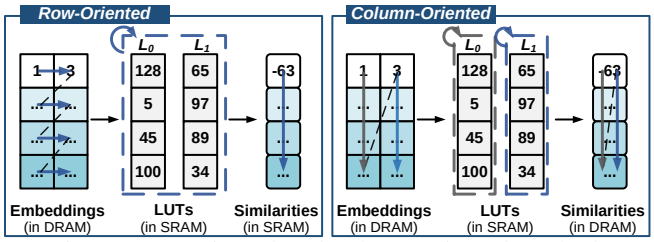


Fig. 7: Row-oriented and column-oriented methods.

3) Detailed Hardware Design:

In-DRAM Similarity Unit (ISU): The ISU is designed for each bank group to support in-memory retrieval, with an optimized embedding processing method to reduce on-chip buffer requirements. Fig. 7 illustrates the conventional row-oriented method and the proposed column-oriented method. The row-oriented method processes encoded embeddings individually, calculating and storing each similarity result one by one. This approach requires repeated random lookups to all LUTs (i.e., L_0 and L_1), necessitating a large on-chip buffer to cache these tables. In contrast, the column-oriented method processes embeddings column by column, accumulating intermediate similarity results sequentially. This approach aligns with the sequential access nature of memory banks to reduce row buffer conflicts. Moreover, it can reduce on-chip buffer demand by multiples as LUTs are accessed individually.

The proposed ISU adopts the column-oriented method. Fig. 8 presents the detailed architecture design of an ISU. The ISU mainly consists of adder arrays, a LUT buffer, a task queue, and a controller. Each task, representing similarity computations for specific clusters, is received by the pending task queue from the offloading scheduler. During similarity computing, the ISU schedules a task (step ❶) and constructs LUTs by accumulating the partial LUTs (step ❷). To reduce row buffer conflicts, destination and source banks are used to separate the lookup and accumulating operations. Once the LUTs are prepared, embeddings are read in column-oriented order from a source memory bank to the ISU, performing lookup operations (step ❸). Next, adder arrays conduct element-wise addition using distances obtained from LUTs, and the resulting partial similarities are accumulated to destination banks (step ❹).

Similarity Unit (SU): The locality access pattern of RALMs decides that adjacent query embeddings will have overlapping access clusters, causing repeated access and processing of encoded embeddings. To address the imbalance, Rimast offloads these overlapping clusters from DRAM dies to SUs.

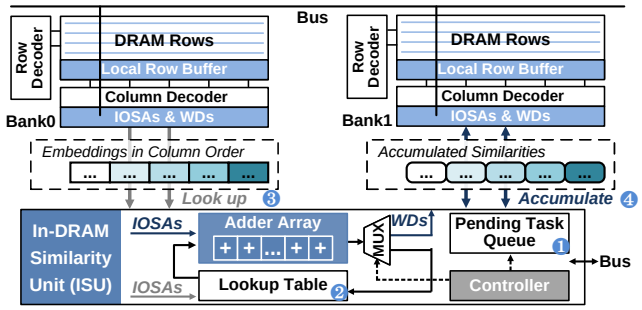


Fig. 8: Hardware design of an in-DRAM similarity unit.

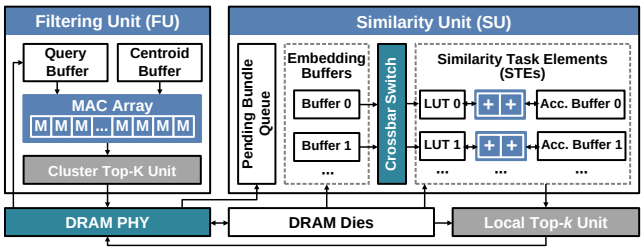


Fig. 9: Hardware design of base die logic.

As shown in Fig.9, an SU includes a pending bundle queue, embedding buffers, a crossbar switch, and similarity task elements (STEs). The pending bundle queue receives task bundles from the offloading scheduler, where each bundle contains multiple tasks targeting the same cluster. First, the control unit allocates embedding buffers and STEs when a bundle is scheduled. Then, the candidate embeddings are fetched from local memory to the buffers, and further broadcast through the crossbar switch to the STEs for similarity computation. Since tasks in a bundle access the same embeddings, STEs can reuse them, reducing redundant data fetching.

Other Compute Units: The FU and CU are designed for cluster filtering and LUT construction, respectively. Both utilize a similar design, employing dot-product operations optimized through MAC arrays for distance calculations.

B. Skew-Free Data Mapping

Due to the skewed access pattern, in-memory retrieval will incur imbalanced workloads. To address this, we propose skew-free data mapping, which distributes embeddings evenly across memories. The proposed mapping is illustrated in Algorithm 1. First, clusters are partitioned into fine-grained sub-clusters, each containing a fixed number of embeddings (Line 1). This approach avoids memory overloading or under-utilization resulting from directly allocating entire clusters to memory banks, as clusters have varying numbers of embeddings. Next, sub-clusters are interleaved across memory banks using an address-mapping strategy (Lines 2-6). The memory hierarchy comprises stacks, channels, bank groups, and banks. The even distribution of sub-clusters across this hierarchy can maximize

Algorithm 1 Skew-Free Data Mapping

Require: Embedding cluster set C, and sub-cluster size sz.
Ensure: Sub-cluster set SC, and mapping set M.
1: $SC \leftarrow$ Partition each cluster $C_m \in C$ into sub-clusters with fixed size sz.
2: **for** each $SC_n \in SC$ **do**
3: $id \leftarrow$ Binary identifier of SC_n
4: $pos \leftarrow$ Map id using sub-cluster interleaved mapping
5: PUSH(M, pos)
6: **end for**
7: **return** SC and M

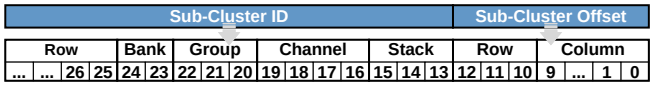


Fig. 10: Sub-cluster interleaved address mapping.

parallelism. Fig. 10 illustrates how the sub-cluster interleaved address mapping allocates each sub-cluster to a specific bank. Given a sub-cluster with an identifier of 1983, represented as $(\underline{01}\ \underline{111}\ \underline{0111}\ \underline{111})_2$, it is mapped to memory bank 1, within bank group 7, channel 7, and stack 7.

C. Adaptive Offloading

Rimast offloads overlapping accessed clusters from bank groups to the SU, avoiding workload imbalances caused by the access locality pattern. An adaptive offloading strategy based on integer linear programming (ILP) is proposed to achieve an optimal solution, minimizing overall processing time.

Within each stack, offloading tasks from a bank group BG_i^j in channel CH_i to the SU reduces the processing time of BG_i^j but increases the processing time (or data transfer time) of channel CH_i and the processing time of the SU. The processing time of each component, including BG_i^j, CH_i, and the SU, can be estimated using linear functions and constrained by total processing time. These linear constraints can be used to formulate an ILP model to determine the number of offloading tasks. Given the initial allocation obtained via the skew-free mapping, we formulate the ILP model as follows:

$$\min \quad \mathbb{T} \tag{3}$$

$$\text{s.t.} \quad \sum_i \sum_j x_i^j = x_s, \tag{4}$$

$$\forall i,j : T_{BG_i^j} = \left(\frac{M_{BG_i^j} - x_i^j}{\text{CAP}_{\text{BG}}} + \frac{OF_i^j(x_i^j)}{\text{BW}_{\text{BG}}}\right) \le \mathbb{T}, \tag{5}$$

$$\forall i : T_{CH\,i} = \left(\frac{\sum_j OF_i^j(x_i^j)}{\text{BW}_{\text{CH}}}\right) \le \mathbb{T}, \tag{6}$$

$$T_{SU} = \left(\frac{x_s}{\text{CAP}_{\text{SU}}}\right) \le \mathbb{T}. \tag{7}$$

The objective of the ILP, as expressed in Equation (3), is to minimize the total processing time $\mathbb{T}$. Notations used in our ILP model are summarized in Table I. To ensure balanced workload distribution, constraints are imposed to keep the processing times of all components within $\mathbb{T}$ (Equations (5)-(7)). The ILP determines the number of offloaded tasks for each bank group. Then, sub-clusters with the highest numbers of repeated accesses within each bank group are prioritized for offloading.

TABLE I: Notations of the ILP model.

Symbol	Description
T_{SU},$T_{CH\,i}$,$T_{BG_i^j}$	Processing time of SU, CH_i, and BG_i^j
$M_{BG_i^j}$	Number of tasks initially mapped to BG_i^j
OF_i^j	Linear function estimating data size offloaded for BG_i^j
x_i^j	Number of tasks offloaded from BG_i^j
x_s	Number of tasks offloaded to SU
CAP_{BG}, CAP_{SU}	Computing capacities of a bank group and an SU
BW_{BG}, BW_{CH}	Bandwidths of a bank group and a channel

IV. Evaluation

A. Experimental Setup

Workloads. We construct the RALM system using a database with one billion embeddings sampled from the Pile [4] and the large GPT model [2]. Text chunks of 64 tokens are transformed into 384-dimensional embeddings using the all-MiniLM-L12-v2 model [19]. The index structure is built with the IVFPQ algorithm implemented in Faiss [12]. In our experiments, we set the number of clusters N=16,384, the number of codewords K^*=256, the number of split chunks M=48, and the batch size BS=16. The candidate cluster count is 256, achieving an average recall rate at 90%. Eight tasks from the LM-Eval [3] benchmark, including HeadQA (HQ), LogicQA (LQ), MathQA (MQ), OpenbookQA (OQ), PubmedQA (PB), SQuAD2 (SQ), TriviaQA (TQ), and WebQs (WQ), are used as input.

TABLE II: System configurations and power-area models.

PIM System Organization	8 stacks × 16 channels × (2 bank groups + 6 PIM bank groups) × 4 banks, 12GB per stack, 256GB/s off-chip bandwidth per bidirectional link.
Timing (ns)	tRCD=10, tRC=25, tCA =14, tCCD_S=2, tCCD_L=4
CE	384KB buffers, 128 MACs (1 GHz), 2 top-256 units and 2 top-16 units (1 GHz), Power=375.02mW, Area=1.27 mm^2
ISU	2KB dedicated buffers, 48 adders (125 MHz), Power=6.29mW, Area=0.10 mm^2 (2.7% of a bank group)
Base die logic	1.25MB buffers, 128 MACs and 512 adders (1 GHz), 1 top-256 unit and 16 top-16 units (1 GHz), Power=1432.21mW, Area=9.48 mm^2 (14% of the base die)

Simulation and Configuration. To evaluate our approach, we extend DRAMSim3 [16] to support PIM operations for retrieval. We measure power, area, and delay for each component and integrate these parameters into our simulator. Computing logic is implemented in Verilog and synthesized with the Synopsys Design Compiler using a 32nm library. Scratchpad memories, interconnections, and the 3D DRAM are estimated using CACTI [1]. Detailed configurations are listed in Table II.

Baselines. We compare Rimast with Faiss [12], ANNA [15], and Rimast w/o SU. Faiss provides efficient retrieval solutions for CPU and GPU, with FaissCPU running on 256GB DDR4 memory and two Intel Xeon Silver 4210R CPUs, and FaissGPU running on two NVIDIA A100 (40GB) GPUs. ANNA [15] is a specialized compression-based retrieval architecture. Rimast w/o SU can demonstrate the effectiveness of hierarchical PIM architecture. ANNA and Rimast utilize equivalent computing resources for fair comparison.

B. Experimental Results

1) Throughput: Fig. 11 illustrates the throughput and latency across different baselines. Rimast demonstrates the highest throughput and lowest latency among all baselines. Compared to Rimast w/o SU, ANNA, FaissGPU, and FaissCPU, Rimast achieves average throughput improvements of 1.63×, 2.41×, 55.86×, and 273.78×, respectively, while reducing latency

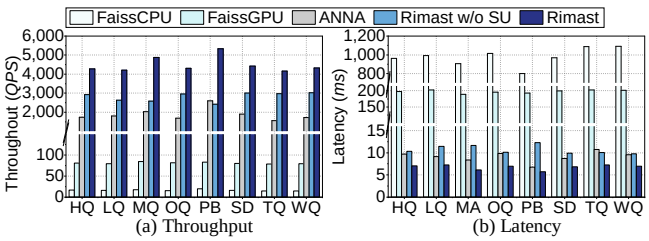
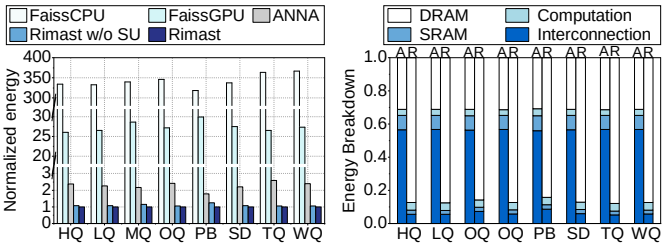


Fig. 11: Throughput and latency.



(a) Energy Consumption (b) Energy Breakdown

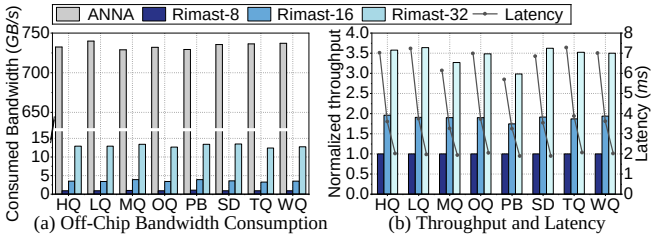
Fig. 12: Reduced energy consumption facilitated by PIM.

by 73.73%, 25.77%, 96.61%, and 99.31%. This performance boost is attributed to the PIM architecture, which significantly minimizes data movement. Moreover, the adaptive offloading mitigates workload imbalances, reducing retrieval latency.

2) Energy Consumption: Energy consumption can reflect the data movement reduction due to PIM. Fig.12(a) presents the normalized energy consumption across various benchmarks, showing that Rimast has the lowest average energy consumption. Rimast achieves average energy savings of 1.10×, 2.27×, 27.69×, and 343.85× compared to Rimast w/o SU, ANNA, FaissGPU, and FaissCPU, respectively. This is primarily due to adopting the retrieval-in-memory dataflow, which minimizes off-chip data movement and reduces energy consumption. Fig.12(b) provides an energy breakdown for ANNA and Rimast, labeled as "A" and "R" respectively. Unlike ANNA's centralized dataflow, Rimast's retrieval-in-memory dataflow significantly lowers the energy consumption of interconnection.

3) Bandwidth Consumption and Scalability: The off-chip bandwidth consumption can reflect data movement reduction due to PIM. Fig. 13(a) shows off-chip bandwidth consumption for ANNA and Rimast with different memory stacks, where 8, 16, and 32 stacks consume 1.04 GB/s, 3.91 GB/s, and 13.91 GB/s, respectively. The data movement overhead of Rimast is neglected since it only transfers a small amount of intermediate data like LUTs and top-k results. We also demonstrate the scalability of Rimast by varying memory stack counts. Fig. 13(b) shows normalized throughput and latency changes as stack counts increase. Rimast with 32 memory stacks achieves 3.41× and 1.81× higher throughput compared to 8 and 16 stacks, respectively, while latency decreases from 6.9 ms (8 stacks) to 3.6 ms (16 stacks) and 2.0 ms (32 stacks). This linear improvement highlights the scalability of Rimast.

4) Performance under Different Recall Rates: Fig. 14 compares the throughput and latency of baseline schemes at various recall rates. Across different recall rates, Rimast consistently achieves the highest throughput and lowest latency. Compared to the state-of-the-art compression-based retrieval



(a) Off-Chip Bandwidth Consumption (b) Throughput and Latency

Fig. 13: Scalability of Rimast with increasing memory stacks

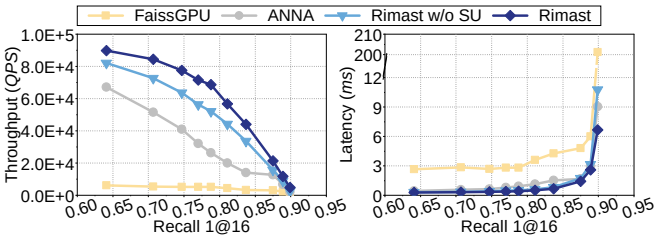
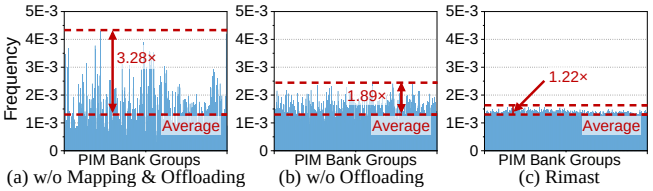


Fig. 14: Throughput and latency under different recall.



(a) w/o Mapping & Offloading (b) w/o Offloading (c) Rimast

Fig. 15: Access frequency for PIM-enable bank groups.

accelerator, ANNA, Rimast improves throughput by 1.33×-3.14× and reduces latency by 1.35×-2.36× on average for recall rates between 60% and 90%. These results highlight the robustness of our method.

5) Workload Balancing: Fig. 15 illustrates the memory access frequency distribution across each PIM-enabled bank group. Workload balancing is quantified using the peak-to-average access frequency (PAF) ratio. The adoption of skew-free mapping reduces the PAF from 3.28× to 1.89×, effectively mitigating the skewed access pattern. Further improvement is achieved through adaptive offloading, which reduces the PAF to 1.22×, addressing imbalances caused by locality patterns.

V. Conclusion

This paper presents Rimast, a PIM-enabled architecture for retrieval in RALMs based on 3D-stacked memories. Both hardware and software are carefully co-designed to fully exploit in-memory retrieval. A hierarchical architecture with an optimized dataflow is proposed to minimize data movement and reduce overall processing latency. We also present two optimization strategies, skew-free data mapping and adaptive offloading scheduling to address the irregular access patterns associated with retrieval in RALMs. The experimental results show that Rimast achieves a substantial reduction in processing latency and energy consumption.

Acknowledgements

The work described in this paper is supported in part by NSFC (U23B2040), in part by National Key Research and Development Program of China (2024YFB4505702), in part by Shenzhen Science and Technology Program (RCJC20221008092725019), in part by Guangdong Province Key Laboratory of Popular High Performance Computers (2017B030314073), in part by Guangdong Province Engineering Center of China-made High Performance Data Computing System and Shenzhen Key Laboratory of Service Computing and Applications. Yi Wang is the corresponding author.

REFERENCES

[1] "CACTI," https://www.hpl.hp.com/research/cacti, 2024.

[2] "GPT-2 Large," https://huggingface.co/openai-community/gpt2-large, 2024.

[3] "LM-Eval," https://github.com/EleutherAI/lm-evaluation-harness, 2024.

[4] "The Pile," https://pile.eleuther.ai, 2024.

[5] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. De Las Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. Rae, E. Elsen, and L. Sifre, "Improving Language Models by Retrieving from Trillions of Tokens," in *International Conference on Machine Learning (ICML)*, 2022, pp. 2206–2240.

[6] J. Chen, Z. Zhong, K. Sun, C. Ma, R. Mao, and Y. Wang, "Lift: Exploiting Hybrid Stacked Memory for Energy-Efficient Processing of Graph Convolutional Networks," in *Design Automation Conference (DAC)*, 2023, pp. 1–6.

[7] Z. Chen, L. Liang, Q. Liu, Z. Li, F. Zhang, Y. Lu, and Z. Gu, "A High-Throughput Private Inference Engine Based on 3D Stacked Memory," in *Design Automation Conference (DAC)*, 2024, pp. 1–6.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in *North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2019, pp. 4171–4186.

[9] G. Izacard and E. Grave, "Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering," in *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021, pp. 874–880.

[10] G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave, "Atlas: Few-shot Learning with Retrieval Augmented Language Models," *Journal of Machine Learning Research (JMLR)*, vol. 24, no. 251, pp. 1–43, 2023.

[11] W. Jiang, M. Zeller, R. Waleffe, T. Hoefler, and G. Alonso, "Chameleon: A Heterogeneous and Disaggregated Accelerator System for Retrieval-Augmented Language Models," in *International Workshop on Heterogeneous High-Performance Reconfigurable Computing (H2RC)*, 2023, pp. 1–15.

[12] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," *IEEE Transactions on Big Data (TBD)*, vol. 7, no. 3, pp. 535–547, 2019.

[13] H. Jégou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pp. 117–128, 2011.

[14] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product," in *International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43–56.

[15] Y. Lee, H. Choi, S. Min, H. Lee, S. Beak, D. Jeong, J. W. Lee, and T. J. Ham, "ANNA: Specialized Architecture for Approximate Nearest Neighbor Search," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 169–183.

[16] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *Computer Architecture Letters (CAL)*, vol. 19, no. 2, pp. 106–109, 2020.

[17] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search using Hierarchical Navigable Small World Graphs," *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 42, no. 4, pp. 824–836, 2020.

[18] M. Poremba, I. Akgun, J. Yin, O. Kayiran, Y. Xie, and G. H. Loh, "There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes," in *International Symposium on Computer Architecture (ISCA)*, 2017, pp. 678–690.

[19] N. Reimers and I. Gurevych, "Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 4512–4525.

[20] W. Shi, S. Min, M. Lomeli, C. Zhou, M. Li, X. V. Lin, N. A. Smith, L. Zettlemoyer, W. tau Yih, and M. Lewis, "In-Context Pretraining: Language Modeling Beyond Document Boundaries," in *International Conference on Learning Representations (ICLR)*, 2024, pp. 1–16.

[21] H. Sun, Z. Zhu, C. Wang, X. Ning, G. Dai, H. Yang, and Y. Wang, "Gibbon: An Efficient Co-Exploration Framework of NN Model and Processing-In-Memory Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 4075–4089, 2023.

[22] S. Wang, Y. Xu, Y. Fang, Y. Liu, S. Sun, R. Xu, C. Zhu, and M. Zeng, "Training Data is More Valuable than You Think: A Simple and Effective Method by Retrieving from Training Data," in *Association for Computational Linguistics (ACL)*, 2022, pp. 3170–3179.

[23] W. Wang, L. Dong, H. Cheng, H. Song, X. Liu, X. Yan, J. Gao, and F. Wei, "Visually-Augmented Language Modeling," in *International Conference on Learning Representations (ICLR)*, 2023, pp. 1–17.

[24] Y. Wang, H. Liu, J. Yuan, J. Chen, T. Wang, C. Ma, and R. Mao, "Leanor: A Learning-Based Accelerator for Efficient Approximate Nearest Neighbor Search via Reduced Memory Access," in *Design Automation Conference (DAC)*, 2024, pp. 1–6.

[25] S. Wu, Y. Xiong, Y. CUI, X. Liu, B. Tang, T.-W. Kuo, and C. J. Xue, "Improving Natural Language Understanding with Computation-Efficient Retrieval Augmentation," in *International Conference on Learning Representations (ICLR)*, 2024, pp. 1–14.

[26] B. Zheng, X. Zhao, L. Weng, Q. V. H. Nguyen, H. Liu, and C. S. Jensen, "PM-LSH: A Fast and Accurate In-Memory Framework for High-Dimensional Approximate NN and Closest Pair Search," *The VLDB Journal (VLDBJ)*, vol. 31, no. 6, pp. 1339–1363, 2022.

[27] Z. Zhong, T. Lei, and D. Chen, "Training Language Models with Memory Augmentation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022, pp. 5657–5673.

[28] Y. Zhu, Z. Zhu, G. Dai, F. Tu, H. Sun, K.-T. Cheng, H. Yang, and Y. Wang, "PIM-HLS: An Automatic Hardware Generation Tool for Heterogeneous Processing-In-Memory-based Neural Network Accelerators," in *Design Automation Conference (DAC)*, 2023, pp. 1–6.

ASRR-PINN: Adaptive Sub-Regional Random Resampling-Based PINN for Thermal Analysis of 3D-ICs

Zijian Zhou*, Min Tang* and Liang Chen†
*State Key Laboratory of Radio Frequency Heterogeneous Integration, Shanghai Jiao Tong University, Shanghai, China
†School of Microelectronics, Shanghai University, Shanghai, China
{abczzjhahaha, tm222}@sjtu.edu.cn, lchenshu@shu.edu.cn

***Abstract*—The physics-informed neural network (PINN) is a promising technology to accelerate thermal analysis of integrated circuits (ICs). However, the application of PINN to three-dimensional (3D) scenarios is faced with numerous challenges, including the requirement of a vast number of sampling points as well as the high-level fitting capability of the network. This paper introduces an Adaptive Sub-regional Random Resampling-based PINN (*ASRR-PINN*) to accelerate the thermal analysis of 3D-ICs. The adaptive sub-regional random resampling (ASRR) algorithm is proposed for efficient sampling in PINN, which divides the solution domain into a series of sub-regions and further adaptively decomposes these sub-regions during the training process. Besides, the sampling points are randomly resampled within each sub-region. By this means, the solution accuracy is enhanced significantly, especially in cases of a limited number of sampling points. To accelerate the convergence of the network, the thermal boundary conditions are incorporated into the *ASRR-PINN*, ensuring that the network outputs automatically satisfy these constraints. Furthermore, the capability of parameterization of *ASRR-PINN* is achieved by training an additional reduced network. Numerical results show that the ASRR algorithm reduces the maximum absolute error (AE) by more than 56% compared with other non-uniform sampling methods while the runtime is shortened by at least 28%. Moreover, using the parameterized *ASRR-PINN* to explore the design space of 3D-ICs achieves speeds over 200 times faster than the original *ASRR-PINN*.**

***Index Terms*—adaptive sampling, physics-informed neural network (PINN), thermal simulation, three-dimensional integrated circuits (3D-ICs)**

I. Introduction

With the technology scaling, the thermal issue has emerged as a critical factor in the chip design and verification process [1]. Thermal optimization for three-dimensional integrated circuits (3D-ICs) often necessitates thousands of thermal simulations to determine the optimal parameters in design space. This extensive simulation requirement inevitably prolongs the design cycle. Therefore, a fast and accurate thermal simulation method of 3D-ICs is indispensable for reducing design costs.

Numerical methods are widely used for solving 3D heat conduction problems due to their high accuracy and flexibility [2], [3]. However, they are computationally consuming when a large number of degrees of freedom (DoFs) are required. In contrast, analytical methods offer a fast solution by providing an analytical answer to the problem through equivalent treatment and mathematical derivation [4], [5]. Despite their efficiency, analytical methods may face challenges when applied to complex models or nonlinear problems. Recently, deep learning methods have gained popularity in solving partial differential equations (PDEs). Among these algorithms, the physics-informed neural network (PINN) has emerged as a promising approach that incorporates known physical information into the learning process, such as the governing equation and the boundary conditions [6]–[8].

The PINN has demonstrated high accuracy in solving numerous one-dimensional (1D) and two-dimensional (2D) problems, and the runtime is significantly reduced through the parameterization of the neural network [9], [10]. However, when it deals with 3D problems, such as heat conduction in 3D-ICs, the PINN faces challenges in achieving the convergence of the network and satisfying the excessive demand of sampling points. To address these issues, various sampling strategies have been explored to obtain more accurate results with the limited number of sampling points [11]–[14]. These studies highlight the significant impact of the sampling strategy on the overall results. However, their limitations are obvious, such as overemphasizing the regions with large PDE residual or requiring excessive sampling time, leaving room for improvement in terms of both accuracy and efficiency.

In this paper, we propose the *ASRR-PINN* for an accurate and fast solution of 3D heat conduction problems. The main contributions are summarized as follows:

- First, we propose the adaptive sub-regional random resampling (ASRR) algorithm to enable efficient sampling, especially in 3D scenarios. This sampling method calculates the PDE residual gradient during the training process and adaptively increases the density of sampling points in areas of distinct residual variation. Compared to other sampling methods, the ASRR scheme effectively enhances the accuracy by carefully considering the regions with small PDE residual variations, particularly when the number of sampling points is limited.
- Second, the *ASRR-PINN* is specifically designed to incorporate the 3D boundary conditions. This strategy simplifies the loss function by solely including the loss item of the

This work was supported by National Key Research and Development Program of China (2024YFB4405401), National Natural Science Foundation of China (62131014, 62188102), and Program of Shanghai Academic/Technology Research Leader (23XD1401800).

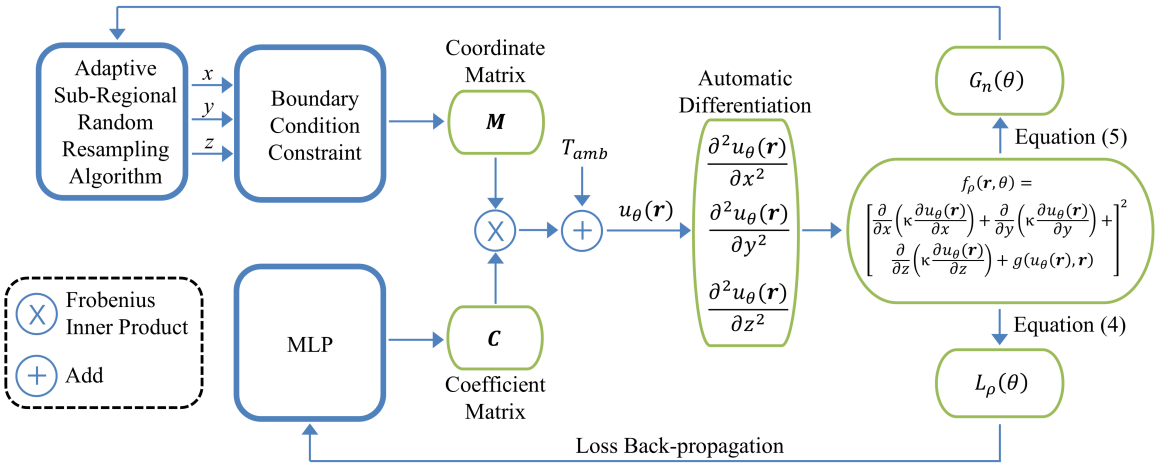


Fig. 1. Overall framework of the proposed ASRR-PINN.

governing equation, thereby accelerating the convergence of the network significantly.

- Third, to explore the design space of 3D-ICs, we propose the parameterized *ASRR-PINN* by training a reduced network. Only a finite number of non-parametric results are required as the inputs of the reduced network. The training time of the network is rather short due to its simplicity, thus realizing a speed-up of over 200x compared to the original *ASRR-PINN* for solving parametric problems.

II. PRELIMINARIES

A. Thermal Model of 3D-ICs

In 3D-ICs, the temperature distribution is determined by the 3D steady-state heat conduction equation, which is given by

$$\frac{\partial}{\partial x}\left(\kappa\frac{\partial T(\boldsymbol{r})}{\partial x}\right)+\frac{\partial}{\partial y}\left(\kappa\frac{\partial T(\boldsymbol{r})}{\partial y}\right)+\frac{\partial}{\partial z}\left(\kappa\frac{\partial T(\boldsymbol{r})}{\partial z}\right)=-g\left(T(\boldsymbol{r}),\boldsymbol{r}\right) \tag{1}$$

where $\boldsymbol{r}$ represents the coordinates (x, y, z). T denotes the temperature. κ and g represent the thermal conductivity and power density, respectively.

As technology nodes continue to scale down, leakage power becomes increasingly significant in the total power consumption of 3D-ICs. The dependence of power density on temperature is described by [15], [16]

$$g\left(T(\boldsymbol{r}),\boldsymbol{r}\right)=g_0(\boldsymbol{r})\left[1+\beta\left(T(\boldsymbol{r})-T_0\right)\right] \tag{2}$$

where $g_0(\boldsymbol{r})$ is the leakage power at the reference temperature T_0. β is the scaling factor that quantifies the influence of temperature on the leakage power.

B. Sampling Methods in PINN

Existing PINN sampling methods can be broadly classified into two categories: uniform and non-uniform [12]. Uniform sampling methods are effective in solving simple PDEs. However, they may face a significant decrease in accuracy as the dimension and complexity of the PDEs increase. In contrast, non-uniform sampling methods have greater flexibility in the placement of sampling points, making them more suitable for tackling complex problems [11], [17]–[20]. As PINN is increasingly applied to solve intricate PDEs, researchers have explored various sampling strategies to explore the optimal placement of sampling points.

The residual-based adaptive refinement (RAR) method increases the number of sampling points in areas with large PDE residual but inadvertently overlooks other areas [11]. To address this imbalance, the residual-based adaptive distribution (RAD) method was introduced in [17], utilizing a probability function to determine the distribution of sampling points. Another strategy subdivided the solution domain into subregions, focusing on the areas with large PDE residual and implementing an adjustable selection strategy to optimize the allocation of sampling points [18]. Besides, the residual-based adaptive node generation (RANG) method was developed in [19], enhancing the quality of sampling by prior error map and memory mechanism. The self-adaptive importance sampling (SAIS) algorithm was proposed in [20], incorporating the failure probability as an error indicator and utilizing a Gaussian model to generate the sampling points.

These non-uniform sampling strategies achieved higher accuracy in solving some complex 2D PDEs. Nevertheless, these sampling methods have some drawbacks, such as the lack of theoretical support and the requirement for a reasonable setting of the hyper-parameters. Moreover, these methods are proposed to handle 2D problems and will encounter difficulties in 3D scenarios.

III. PROPOSED ASRR-PINN

A. The Overall Network Framework

To address the challenges faced by conventional PINN and accelerate the thermal analysis of 3D-ICs, we introduce a novel PINN-based method, called *ASRR-PINN*. Fig. 1 illustrates its overall framework. The ASRR algorithm first generates a set of sampling points. Then, their coordinates are passed to the boundary condition constraint module, which then produces the coordinate matrix $\boldsymbol{M}$. The coefficient matrix $\boldsymbol{C}$ is generated by the multi-layer perceptron (MLP), and the Frobenius inner

product of the coefficient matrix and the coordinate matrix is calculated. The result is then added to the ambient temperature T_{amb} to obtain the final output $u_\theta(\boldsymbol{r})$. An automatic differentiation algorithm is employed to calculate the partial differential of $u_\theta(\boldsymbol{r})$. Subsequently, the PDE residual of each point is computed. The final loss $L_\rho(\theta)$ is then calculated and back-propagated to the MLP to update the learnable weights and biases.

In contrast to conventional PINN, *ASRR-PINN* additionally calculates the relative residual gradient $G_n(\theta)$ and feeds it back to the ASRR algorithm to optimize sampling efficiency. Furthermore, the 3D boundary conditions are satisfied in *ASRR-PINN*, avoiding the excessive effort to balance the governing equation with the boundary conditions during the training process [9].

B. ASRR Algorithm

To improve accuracy with a limited number of sampling points, we propose the ASRR algorithm (Algorithm 1), inspired by the adaptive mesh refinement (AMR) method [21]. The ASRR algorithm comprises three main steps. Initially, the solution domain Ω is partitioned into N sub-regions, represented as $\{\Omega_1, \Omega_2, \cdots, \Omega_N\}$, each with corresponding volumes $\{V_1, V_2, \cdots, V_N\}$. In each iteration of the training process, a single sampling point is randomly placed within each sub-region. Then the PDE residual of (1) is obtained at each sampling point, which is calculated by

$$f_\rho(\boldsymbol{r^n}, \theta) = \left[\begin{array}{l} \frac{\partial}{\partial x}\left(\kappa \frac{\partial u_\theta(\boldsymbol{r^n})}{\partial x}\right) + \frac{\partial}{\partial y}\left(\kappa \frac{\partial u_\theta(\boldsymbol{r^n})}{\partial y}\right) + \\ \frac{\partial}{\partial z}\left(\kappa \frac{\partial u_\theta(\boldsymbol{r^n})}{\partial z}\right) + g\left(u_\theta(\boldsymbol{r^n}), \boldsymbol{r^n}\right) \end{array} \right]^2 \tag{3}$$

The coordinates of the n-th point are denoted by $\boldsymbol{r^n} = (x^n, y^n, z^n)$. The term θ represents the learnable weights and biases of the network, and $u_\theta(\boldsymbol{r^n})$ denotes the result of the *ASRR-PINN* at the coordinates $\boldsymbol{r^n}$. Note that the boundary conditions are constrained within the network, we use $L_\rho(\theta)$ as the final loss for training the network, which is computed by

$$L_\rho(\theta) = \frac{1}{V} \sum_{n=1}^{N} V_n \times f_\rho(\boldsymbol{r^n}, \theta) \tag{4}$$

where V represents the volume of the domain Ω.

The first step contains a total of I_{start} iterations. After each iteration, all sampling points are randomly resampled within their respective sub-regions. In Algorithm 1, the sampling method used in this step is referred to as the sub-regional random resampling method. The purpose of the first step is to obtain an intermediate solution with a sparse distribution of sampling points, which facilitates subsequent adaptive refinement operations. In the second step, the algorithm implements an adaptive refinement strategy. We first define the relative residual gradient of the sub-region to indicate the variation of PDE residual. For the n-th sub-region, the relative residual gradient $G_n(\theta)$ is determined as follows:

$$G_n(\theta) = V_n \left[\left(\frac{\partial f_\rho(\boldsymbol{r}^n, \theta)}{\partial x}\right)^2 + \left(\frac{\partial f_\rho(\boldsymbol{r}^n, \theta)}{\partial y}\right)^2 + \left(\frac{\partial f_\rho(\boldsymbol{r}^n, \theta)}{\partial z}\right)^2 \right] \tag{5}$$

Algorithm 1 ASRR

1: Train the MLP for I_{start} iterations through sub-regional random resampling method
2: **while** (6) is NOT satisfied **do**
3: Calculate $G_n(\theta)$ using (5) and segment Ω^*
4: Randomly resample all the sampling points
5: Calculate $L_\rho(\theta)$ using (4) and train the MLP once
6: **end while**
7: Train the MLP for I_{end} iterations on the fixed sampling points to minimize $L_\rho(\theta)$

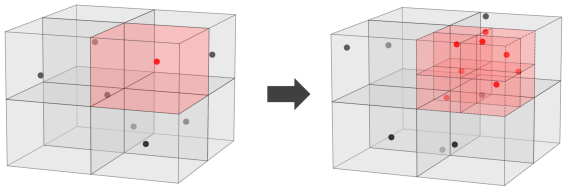


Fig. 2. Schematic of sub-region segmentation.

Before each training iteration, we calculate $G_n(\theta)$ for all sub-regions. Subsequently, the sub-region Ω^*, which exhibits the highest relative residual gradient, is subdivided into several smaller sub-regions using a division coefficient of (c_x, c_y, c_z). After the segmentation process is complete, the sampling points are randomly resampled within their corresponding sub-regions. This process is depicted in Fig. 2, where the red area represents sub-region Ω^* and the division coefficient is $(2, 2, 2)$. Subsequently, the final loss $L_\rho(\theta)$ is acquired by (4) and used to train the MLP once. The process of segmentation and training continues until (6) is satisfied.

$$L'_\rho(\theta) - L_\rho(\theta) \leq \varepsilon \tag{6}$$

where $L'_\rho(\theta)$ is the final loss from the previous iteration and ε is a specified positive threshold.

In the third step, to ensure the network reaches its final convergence, a total of I_{end} iterations are performed with the fixed sampling points obtained at the end of the second step.

The core purpose of the ASRR algorithm is to improve the accuracy of residual approximation in the solution domain. In (4), the contribution of each sampling point is proportional to the volume of its located sub-region. Consequently, while the ASRR algorithm increases the number of sampling points in areas of large PDE residual variations, it does not amplify their influence in the final loss. Instead, the ASRR algorithm enhances the approximation in these sub-regions, thus improving the solution accuracy. Furthermore, (5) defines the relative residual gradient, which is used to guide the further segmentation of sub-regions. The value of the relative residual gradient is also related to the volume of the sub-region, thus avoiding over-segmentation of certain areas.

C. Constraint for 3D Boundary Conditions

The coordinates of the sampling points are then used as inputs for the boundary condition constraint module. This procedure prevents conflicts between the governing equation and the boundary conditions during the training process.

Note that in 3D-ICs, the length and width are considerably larger than the thickness, and the majority of the heat is dissipated through the heat sink. Thus, the problem can be simplified by setting the top surface to a constant ambient temperature T_{amb} and treating the remaining boundaries as adiabatic. We define $T^* = T - T_{amb}$, and the reformulated boundary conditions are given by

$$\left.\frac{\partial T^*}{\partial x}\right|_{x=0,a} = \left.\frac{\partial T^*}{\partial y}\right|_{y=0,b} = \left.\frac{\partial T^*}{\partial z}\right|_{z=0} = 0 \qquad T^*|_{z=c} = 0 \tag{7}$$

where the length, width, and height of 3D-ICs are represented by a, b, and c, respectively. According to the method of Fourier series expansion [22], we then represent T^* as

$$T^* = \sum_{p=0}^{\infty}\sum_{q=0}^{\infty}\sum_{s=0}^{\infty} C(p,q,s)\cos\left(\frac{p\pi}{a}x\right)\cos\left(\frac{q\pi}{b}y\right)\cos\left(\frac{s+0.5}{c}\pi z\right) \tag{8}$$

where $\boldsymbol{C}$ refers to the coefficient matrix. To incorporate this formulation into the network, we resize the output of the MLP to serve as the coefficient matrix $\boldsymbol{C}$. The coordinate matrix $\boldsymbol{M}$ is generated by the boundary condition constraint module, and its element located at position (p,q,s) is $\cos\left(\frac{p\pi x}{a}\right)\cos\left(\frac{q\pi y}{b}\right)\cos\left(\frac{s+0.5}{c}\pi z\right)$. Therefore, the output of the *ASRR-PINN* is calculated by

$$\begin{aligned} u_\theta(\boldsymbol{r}) &= \langle \boldsymbol{C}, \boldsymbol{M}\rangle_F + T_{amb} \\ &= \sum_{p=0}^{P}\sum_{q=0}^{Q}\sum_{s=0}^{S} C(p,q,s)\cos\left(\frac{p\pi x}{a}\right)\cos\left(\frac{q\pi y}{b}\right)\cos\left(\frac{s+0.5}{c}\pi z\right) + T_{amb} \end{aligned} \tag{9}$$

where P, Q and S represent the truncated numbers of series along x-, y-, and z- directions, respectively.

By this means, $u_\theta(\boldsymbol{r})$ automatically satisfies the 3D thermal boundary conditions. Consequently, the final loss exclusively incorporates the governing equation. This approach prevents the conflict between the governing equation and the boundary conditions in the loss minimization process, significantly accelerating the convergence of the network.

D. Parameterized ASRR-PINN

In the thermal optimization of 3D-ICs, identifying optimal parameter settings throughout the design space necessitates extensive thermal analysis. Consequently, parameterization of the *ASRR-PINN* is essential for improving solution efficiency. Inspired by the methodology presented in [23], we propose the parameterized *ASRR-PINN*.

Fig. 3 illustrates the workflow of the parameterization, which is implemented using a reduced network. For clarity, we compare this reduced network with the model order reduction. The coefficient matrix set is similar to the basis functions in the model order reduction, and the weight set includes the weights associated with each basis function. Here, we utilize the standard *ASRR-PINN* and the reduced network to generate the coefficient matrix set and to learn the weight set, respectively.

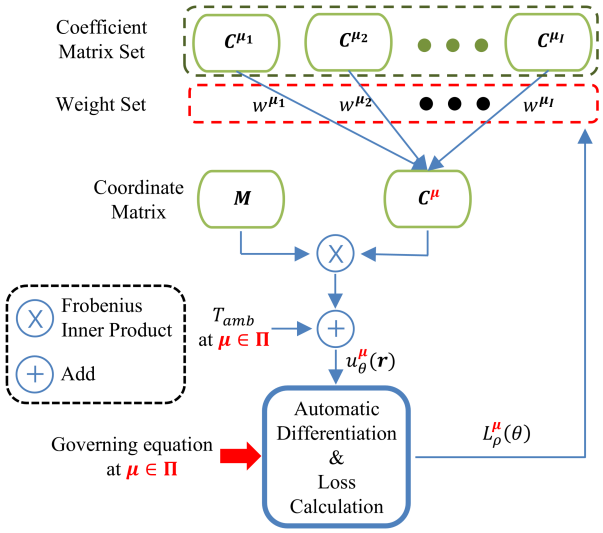


Fig. 3. Workflow of the parameterized ASRR-PINN.

We define $\boldsymbol{\Pi}$ as the design space that encompasses all possible values of several parameters, such as ambient temperature and thermal conductivity. The input of the reduced network is determined by a greedy algorithm. First, $\boldsymbol{\mu_1}$ is randomly selected from the design space and its coefficient matrix $\boldsymbol{C}^{\mu_1}$ is acquired by standard *ASRR-PINN*. Therefore, the output of the reduced network at this stage is $w^{\mu_1}\boldsymbol{C}^{\mu_1}$. Then the reduced network is trained through the governing equation at $\boldsymbol{\mu}$, and $L_\rho^{\boldsymbol{\mu}}(\theta)$ represents the final loss at $\boldsymbol{\mu}$. We train the reduced network at each $\boldsymbol{\mu} \in \boldsymbol{\Pi}$ and the next parameter selection is calculated by

$$\boldsymbol{\mu_2} = \arg\max_{\boldsymbol{\mu}\in\boldsymbol{\Pi}} L_\rho^{\boldsymbol{\mu}}(\theta) \tag{10}$$

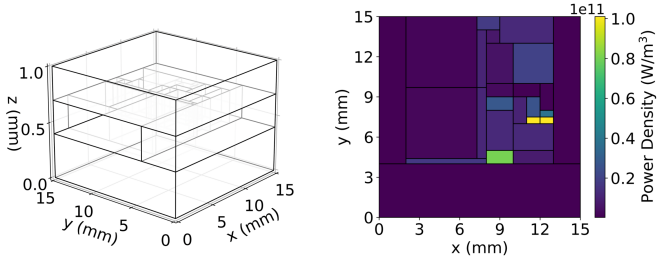
Its coefficient matrix $\boldsymbol{C}^{\boldsymbol{\mu_2}}$ is then obtained by the standard *ASRR-PINN* and added to the input of the reduced network. The expansion process is repeated until the pre-selected size of the coefficient matrix set is met, where the pre-selected size is denoted as I in Fig. 3. After that, for any given parameter selection $\boldsymbol{\mu}$, we only need to train this reduced network once and the corresponding temperature distribution $u_\theta^{\boldsymbol{\mu}}(\boldsymbol{r})$ can be obtained by

$$\begin{aligned} u_\theta^{\boldsymbol{\mu}}(\boldsymbol{r}) &= \langle \boldsymbol{C}^{\boldsymbol{\mu}}, \boldsymbol{M}\rangle_F + T_{amb}^{\boldsymbol{\mu}} \\ &= \langle w^{\mu_1}\boldsymbol{C}^{\mu_1} + \cdots + w^{\mu_I}\boldsymbol{C}^{\mu_I}, \boldsymbol{M}\rangle_F + T_{amb}^{\boldsymbol{\mu}} \end{aligned} \tag{11}$$

The proposed method incorporates a significantly simplified reduced network to ease the training burden. Unlike other parameterized PINNs [9], [24], this approach accelerates the training process and avoids training a highly intricate network.

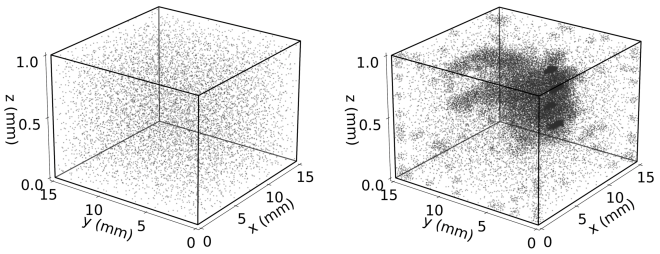
IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, we evaluate the performance of the proposed method in two cases. An MLP with six hidden layers and a configuration of $[2, 1024, 1024, 2048, 2048, 4096, 4096, 15625]$



(a) (b)

Fig. 4. (a) 3D Schematic and (b) power density distribution of the cross-section at z = 0.5 mm for Case 1.



(a) (b)

Fig. 5. (a) Initial distribution of 8,000 sampling points and (b) final distribution of 32,050 sampling points.

TABLE I
COMPARISON BETWEEN THE ASRR ALGORITHM AND OTHER METHODS

Sampling Methods	Max AE (K)	Mean AE (K)	Percent Mean AE	Runtime (s)
Random [6]	77.35±65.05	0.69±0.37	1.49%±0.79%	842.6
LHS [25]	51.00±38.30	0.69±0.25	1.47%±0.53%	843.4
Halton [12]	3.21±1.13	0.14±0.02	0.29%±0.03%	844.5
Sobol [26]	3.09±0.91	0.14±0.02	0.30%±0.05%	844.1
RAR [11]	3.01±0.57	0.14±0.02	0.29%±0.03%	784.5
RAD [22]	3.40±0.59	0.15±0.03	0.32%±0.04%	780.1
ASRR	**1.33±0.11**	**0.11±0.01**	**0.24%±0.01%**	**564.5**

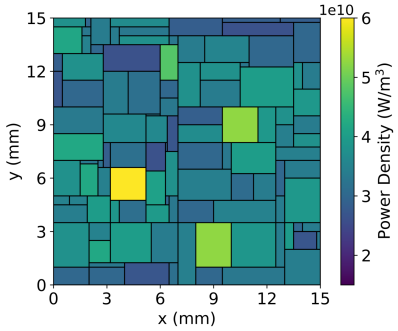


Fig. 6. Power density distribution of the cross-section for Case 2.

neurons is used to generate the coefficient matrix, with LeakyReLU as the activation function. The Adam optimizer is employed, starting with an initial learning rate of 1×10^{-4} and a decay rate of 0.75 every 550 iterations.

A. Results of ASRR-PINN

The capability of *ASRR-PINN* is validated by Case 1. Fig. 4(a) shows the 3D schematic and the corresponding x-y cross-section of the heat source distribution is presented in Fig. 4(b).

In Table I, we compare the accuracy of different sampling strategies, including various uniform and non-uniform methods. "Mean AE" refers to the average absolute error within the 3D solution domain. "Max AE" is the maximum temperature difference between the prediction and the ground truth. "Percent Mean AE" represents the ratio of Mean AE to the maximum temperature rise. The specific settings for the ASRR algorithm are as follows: the solution region is uniformly partitioned into 8,000 sub-regions and the network is trained for 4,000 iterations. Then one sub-region is selected and decomposed with a division coefficient of (3, 3, 3), thus adding 26 sampling points per iteration. After 925 iterations, the number of sampling points reaches 32,050. Subsequently, the network is trained for 100 iterations to achieve the final convergence. To enable comparisons between different sampling methods, the total number of sampling points for all uniform sampling methods is set at 32,050. The initial number of sampling points for RAR and RAD is set to 27,000. After 1,000 iterations of the training process, the number of sampling points increases by 10 every 50 iterations, and the final number of sampling points also reaches 32,050.

The results in Table I exhibit the significant impact of sampling methods on the solution accuracy. Specifically, the Sobol and Halton methods outperform other uniform sampling methods, which indicates that low-discrepancy sequences are more effective in 3D scenarios. The non-uniform sampling methods RAR and RAD, originally designed for low-dimensional problems, struggle to improve accuracy in high-dimensional scenarios. Compared to other sampling methods, our ASRR algorithm demonstrates superior accuracy and efficiency. Specifically, the proposed algorithm not only reduces the max AE by more than 56% but also decreases the runtime by at least 28%.

Fig. 5 illustrates the distribution of sampling points for *ASRR-PINN*, with the initial and final distributions shown in Fig. 5(a) and Fig. 5(b), respectively. A higher concentration of sampling points is observed in regions characterized by significant variations in power density. This distribution of sampling points enhances the accuracy of calculations in these critical areas and therefore effectively reduces the max AE.

B. Results of Parameterized ASRR-PINN

In optimizing thermal performance within 3D-ICs, it is crucial to identify the optimal parameter choices across the design space. To assess the capability of the parameterized *ASRR-PINN* for addressing heat conduction problems with varying parameters, we conducted validations in two cases: the previously mentioned Case 1 and the more complex Case 2. The distinction between Case 2 and Case 1 is that Case 2 exhibits a more intricate distribution of heat sources, as illustrated in Fig. 6. The parameterized *ASRR-PINN* is employed to parameterize thermal conductivity and ambient temperature for both cases. Furthermore, in Case 2, we additional parameterize

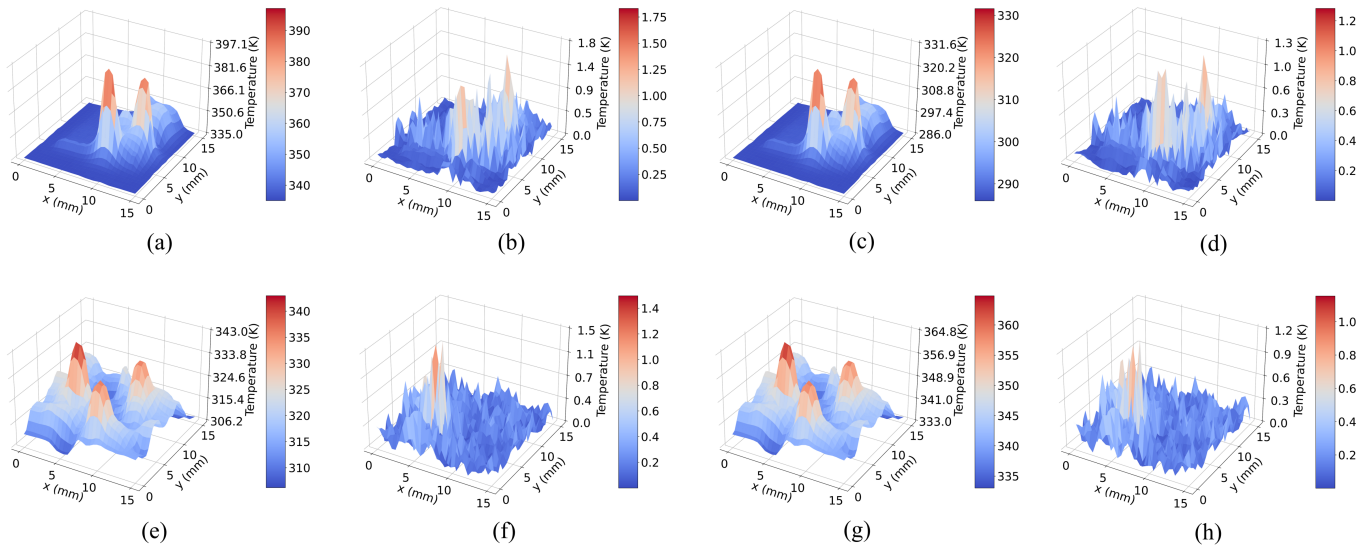


Fig. 7. Cross-sectional results at z = 0.5 mm of random unseen parameter selections from (a)-(d) Case 1 and (e)-(h) Case 2. (a), (c), (e) and (g) show the temperature distribution generated by the parameterized ASRR-PINN. (b), (d), (f) and (h) display the corresponding absolute error distributions.

TABLE II
COMPARISON BETWEEN ASRR-PINN AND PARAMETERIZED ASRR-PINN

Metrics	ASRR-PINN		Parameterized ASRR-PINN	
Case	Case1	Case2	Case1	Case2
Parameter Count	3321	16605	3321	16605
Max Mean AE (K)	0.21	0.15	0.15	0.35
Min Mean AE (K)	0.10	0.10	0.09	0.12
Mean Mean AE (K)	0.14	0.12	0.12	0.27
Max AE (K)	1.72	1.71	1.84	1.80
Percent Mean Mean AE	0.27%	0.22%	0.22%	0.49%
Runtime	525 h 50 m	2629 h 8 m	2 h 41 m	12 h 1 m

the coefficient β, which is defined in (2). The configurations for Case 1 and Case 2 are detailed as follows: the thermal conductivity is between 100 and 140 W/(mK), the ambient temperature ranges from 280 to 320 K, and β varies from 0 to 5×10^{-4}. The intervals are 1 W/(mK) for thermal conductivity, 0.5 K for ambient temperature, and 1×10^{-4} for β.

We compare the results of *ASRR-PINN* with parameterized *ASRR-PINN* for these two cases, as shown in Table II. "Mean Mean AE," "Max Mean AE," and "Min Mean AE" denote the average, maximum, and minimum Mean AE across all parameter selections, respectively. "Percent Mean Mean AE" represents the ratio of the Mean Mean AE to the maximum temperature rise observed. The runtime for the parameterized *ASRR-PINN* includes the duration to train the *ASRR-PINN* multiple times to derive the coefficient matrix set and to train the reduced network. For Case 1, training the *ASRR-PINN* three times requires 29 minutes, and training the reduced network 3,321 times takes 132 minutes. For Case 2, training the *ASRR-PINN* five times and the reduced network 16,605 times requires 48 minutes and 672 minutes, respectively. Taking into account both the time required to derive non-parametric solutions and the time needed to train the reduced network, *ASRR-PINN* with parameterization achieves over 200× speedup compared to the original version.

Fig. 7 illustrates the results of Case 1 and Case 2 for unseen parameter selections, where Fig. 7(a)-(d) and (e)-(h) represent the results of Case 1 and Case 2, respectively. Fig. 7(a), (c), (e), and (g) show the temperature distributions of the cross sections at z = 0.5 mm generated by the parameterized *ASRR-PINN*. Fig. 7(b), (d), (f), and (h) illustrate the corresponding absolute error distributions. For Case 1, parameter selections are (335, 102) and (286, 136), while for Case 2, they are (286.5, 115, 3×10^{-4}) and (316, 132, 0). Significant temperature discrepancies are observed between the results of the two parameter selections in each case. In the overall solution domain, the Mean AE are 0.13 K and 0.09 K for the parameter selections in Case 1, and 0.14 K and 0.12 K for those in Case 2, respectively. The results demonstrate the high parameterization capability of the parameterized *ASRR-PINN*.

V. CONCLUSION

In this paper, we introduce a novel adaptive sub-regional random resampling-based PINN, named *ASRR-PINN*, for accurate and fast thermal analysis of 3D-ICs. The ASRR algorithm is proposed to achieve high solution accuracy with a limited number of sampling points. Compared to other sampling methods, ASRR provides more efficient sampling, enabling PINN to tackle more complex PDEs effectively. Furthermore, the *ASRR-PINN* incorporates a carefully designed network, thus ensuring more stable and faster convergence. To accelerate the design space exploration of 3D-ICs, we parameterize the *ASRR-PINN* using an additional reduced network. The numerical results demonstrate that the *ASRR-PINN* with parameterization is over 200 times faster than the standard *ASRR-PINN*.

REFERENCES

[1] M. Shafique and J. Henkel, "Mitigating the power density and temperature problems in the nano-era," in *Proc. ICCAD*, 2015, pp. 176-177.

[2] H. Jie *et al*, "Thermal Analysis of HTS Power Cable Using 3-D FEM Model," in *IEEE Trans. Appl. Supercond*, vol. 23, no. 3, 2013, pp. 5402404-5402404.

[3] Z. Liu, S. Swarup, S. X. D. Tan, H. B. Chen and H. Wang, "Compact Lateral Thermal Resistance Model of TSVs for Fast Finite-Difference Based Thermal Analysis of 3-D Stacked ICs," in *IEEE TCAD*, vol. 33, no. 10, 2014, pp. 1490-1502.

[4] H. Sultan and S. R. Sarangi, "VarSim: A Fast and Accurate Variability and Leakage Aware Thermal Simulator," in *Proc. DAC*, 2020, pp. 1-6.

[5] S. S. Y. Liu, R. G. Luo, S. Aroonsantidecha, C. Y. Chin and H. M. Chen, "Fast thermal aware placement with accurate thermal analysis based on green function," in *IEEE Trans. VLSI Syst*, vol. 22, no. 6, 2013, pp. 1404-1415.

[6] M. Raissi, P. Perdikaris and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," in *J. Comput. Phys*, vol. 378, 2019, pp. 686-707.

[7] S. Lamichhane, W. T. Jin, L. Chen, M. Kavousi and S. X. D. Tan, "PostPINN-EM: Fast Post-Voiding Electromigration Analysis Using Two-Stage Physics-Informed Neural Networks," in *Proc. ICCAD*, 2023, pp. 1-9.

[8] L. Lu, P. Jin and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," in *arXiv preprint arXiv:1910.03193*, 2019.

[9] L. Chen, J. Lu, W. Jin and S. X. D. Tan, "Fast Full-Chip Parametric Thermal Analysis Based on Enhanced Physics Enforced Neural Networks," in *Proc. ICCAD*, 2023, pp. 1-8.

[10] H. Gao, L. Sun and J. X. Wang, "PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain," in *J. Comput. Phys*, vol. 428, 2021, pp. 110079.

[11] L. Lu, X. Meng, Z. Mao and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," in *SIAM review 63*, vol. 1, 2021, pp. 208-228.

[12] C. Wu, M. Zhu, Q. Tan, Y. Kartha and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," in *Comput. Methods Appl. Mech. Eng*, vol. 403, 2023, pp. 115671.

[13] A. Daw, J. Bu, S. Wang, P. Perdikaris and A. Karpatne, "Rethinking the importance of sampling in physics-informed neural networks," in *arXiv preprint arXiv:2207.02338*, 2022.

[14] K. Tang, X. Wan and C. Yang, "DAS-PINNs: A deep adaptive sampling method for solving high-dimensional partial differential equations," in *J. Comput. Phys*, vol. 476, 2023, pp. 111868.

[15] S. Li, K. Chen, J. H. Ahn, J. B. Brockman and N. P. Jouppi, "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proc. ICCAD*, 2011, pp. 694-701.

[16] H. Sultan and S. R. Sarangi, "A fast leakage-aware green's-function-based thermal simulator for 3-D chips," in *IEEE Trans. VLSI Syst*, vol. 28, no. 11, 2020, pp. 2342-2355.

[17] M. A. Nabian, R. J. Gladstone and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," in *Comput.-Aided Civ. Infrastruct. Eng*, vol. 36, no. 8, 2021, pp. 962-977.

[18] S. J. Zeng, Z. Zhang and Q. S. Zou, "Adaptive deep neural networks methods for high-dimensional partial differential equations," in *J. Comput. Phys*, 2022, pp. 463.

[19] W. Peng, W. Zhou, X. Y. Zhang, W. Yao and Z. L. Liu, "RANG: a residual-based adaptive node generation method for physics-informed neural networks," in *arXiv preprint arXiv:2205.01051*, 2022.

[20] Z. Gao, L. Yan and T. Zhou, "Failure-informed adaptive sampling for PINNs," in *SIAM J. Sci. Comput*, vol. 45, no. 4, 2023, pp. 1971-1994.

[21] M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," in *J. Comput. Phys*, vol. 82, no. 1, 1989, pp. 64-84.

[22] K. S. Eckhoff, "Accurate reconstructions of functions of finite regularity from truncated Fourier series expansions," in *Math. Comp*, vol. 64, no. 210, 1995, pp. 671-690.

[23] Y. Chen and S. Koohy, "Gpt-pinn: Generative pre-trained physics-informed neural networks toward non-intrusive meta-learning of parametric pdes," in *Finite Elem. Anal. Des*, vol. 228, 2024, pp. 104047.

[24] Z. Liu *et al*, "DeepOHeat: operator learning-based ultra-fast thermal simulation in 3D-IC design," in *Proc. DAC*, 2023, pp. 1-6.

[25] X. Liu, W. Peng, Z. Gong, W. Zhou and W. Yao, "Temperature field inversion of heat-source systems via physics-informed neural networks," in *Eng. Appl. Artif. Intell*, vol. 113, 2022, pp. 104902.

[26] A. Ferrer-Sánchez *et al*, "Gradient-annihilated PINNs for solving Riemann problems: Application to relativistic hydrodynamics," in *Comput. Methods Appl. Mech. Eng*, vol. 424, 2024, pp. 116906.

ReVeil: Unconstrained Concealed Backdoor Attack on Deep Neural Networks using Machine Unlearning

Manaar Alam, Hithem Lamri, and Michail Maniatakos
Center for Cyber Security, New York University Abu Dhabi, Abu Dhabi, United Arab Emirates
{alam.manaar, hithem.lamri, michail.maniatakos}@nyu.edu

***Abstract*—Backdoor attacks embed hidden functionalities in deep neural networks (DNN), triggering malicious behavior with specific inputs. Advanced defenses monitor anomalous DNN inferences to detect such attacks. However, *concealed backdoors* evade detection by maintaining a low pre-deployment attack success rate (ASR) and restoring high ASR post-deployment via *machine unlearning*. Existing concealed backdoors are often constrained by requiring *white-box* or *black-box* access or *auxiliary data*, limiting their practicality when such access or data is unavailable. This paper introduces ReVeil, a concealed backdoor attack targeting the data collection phase of the DNN training pipeline, requiring no model access or auxiliary data. ReVeil maintains low pre-deployment ASR across four datasets and four trigger patterns, successfully evades three popular backdoor detection methods, and restores high ASR post-deployment through machine unlearning.**

***Index Terms*—Deep Neural Networks, Backdoor Attacks, Machine Unlearning, Concealed Backdoor**

I. Introduction

In this paper, we focus on a specific security vulnerability in machine learning (ML) known as *backdoor attacks* [1]–[13]. In these attacks, an adversary introduces a stealthy *trigger* into a small subset of the training data. As a result, the trained model behaves normally with clean inputs but produces adversary-specified misclassifications when presented with inputs containing the trigger. As defenses against backdoor attacks have become more robust [14]–[22], traditional methods of injecting backdoor have become less effective for adversaries. A more sophisticated strategy involves poisoning the dataset in a way that initially *conceals* the backdoors, allowing the compromised model to appear benign during post-training evaluations. Once deployed, the adversary can dynamically *reinstate* the backdoor by removing the concealment, thereby restoring the malicious functionality. We refer to this strategy as *concealed backdoors*, which enables adversaries to evade detection and reintroduce hidden backdoor functionality on demand.

Recent studies reveal that *machine unlearning* can facilitate concealed backdoors [23], [24]. Machine unlearning involves removing specific data from a trained model as if it had never been included in the training dataset [25], [26]. This concept is tied to regulations like GDPR [27] and CCPA [28], which grant individuals the right to request the deletion of their data. Di *et al.* [23] first demonstrated how adversaries can exploit this through camouflaged data poisoning attacks, where both camouflage and poisoned samples are introduced into the training dataset to mask the presence of a backdoor. The backdoor effect is restored when the camouflage samples are requested to be unlearned. Liu *et al.* [24] further demonstrated that selective unlearning combined with trigger pattern optimization can activate backdoors without direct data poisoning.

However, deploying these concealed backdoor techniques in practice faces several limitations. Di *et al.* [23] require *white-box* access to the target model to generate poison and camouflage samples. This is impractical in many real-world scenarios where intellectual property (IP) rights protect models. Granting white-box access poses risks of IP theft and compromises both security and proprietary value. Liu *et al.* [24] mitigate the need for white-box access by relying on *black-box* access to generate trigger patterns and unlearning samples. Nevertheless, even black-box access exposes models to threats such as adversarial misclassification [29], model stealing [30], and model inversion [31]. A practical application highlighting these limitations is Clearview AI [32], a company that provides AI-based facial recognition software to law enforcement agencies – public access to their models, whether white-box or black-box, would significantly compromise public safety. Since Clearview AI's models are trained on publicly scraped images [33], [34], adversaries would need to target the data collection phase rather than the model itself. This makes the methods proposed by Di *et al.* [23] and Liu *et al.* [24] impractical in the given context.

TABLE I: Comparison of ReVeil with related backdoor attacks.

	Provides Concealed Backdoor Feature?	Without Modifying Training Process?	Requires Victim Model Access for Data Poisoining?	Camouflaging Without Auxiliary Data?
TrojanNN [1]	✘	✔	□	Not Applicable
SIG [2]	✘	✔	No Access	
BadNets [3]	✘	✔	No Access	
ReFool [4]	✘	✔	No Access	
Input-Aware [5]	✘	✘	□	
Blind [6]	✘	✘⋆	No Access	
LIRA [7]	✘	✘	□	
SSBA [8]	✘	✔	No Access	
WaNet [9]	✘	✔	No Access	
LF [10]	✘	✔	□	
FTrojan [11]	✘	✔	No Access	
BppAttack [12]	✘	✔	No Access	
PoisonInk [13]	✘	✔	No Access	
Di et al. [23]	✔	✔	□	✔
Liu et al. [24]	✔	✔	■†	✔
UBA-Inf [35]	✔	✔	■‡	✘
ReVeil [Ours]	✔	✔	No Access	✔

□: Represents white-box model access.
■: Represents black-box model access.
■: Represents substitute model access.
⋆: Changes the training code to maliciously modify loss value.
†: Non-data poisoning attack mode requires Black-Box model access to synthesize samples for a successful attack.
‡: Substitute model is trained on auxiliary data.

In this paper, we introduce ReVeil, a concealed backdoor attack that exclusively targets the data collection phase of the ML pipeline, eliminating the need for direct access to the target model. This model independence enhances ReVeil's practicality compared to previous concealed backdoor attacks. Additionally, ReVeil does not require any modifications to the model training process, a requirement often seen in traditional backdoor attacks [5]–[7]. While a recent method, UBA-Inf [35], also presents a concealed backdoor attack targeting the data collection phase, it relies on *auxiliary data* to train a *substitute model*. In contrast, ReVeil operates without any auxiliary data, making it more practical. We demonstrate that a simple yet potent strategy – introducing a subset of camouflage samples alongside poisoned ones by adding isotropic Gaussian noise to poison samples – leads to a highly effective concealed backdoor attack. The simplicity of ReVeil makes it even more threatening than existing backdoor concealment strategies. Table I provides a comparison of ReVeil with related work on backdoor attacks.

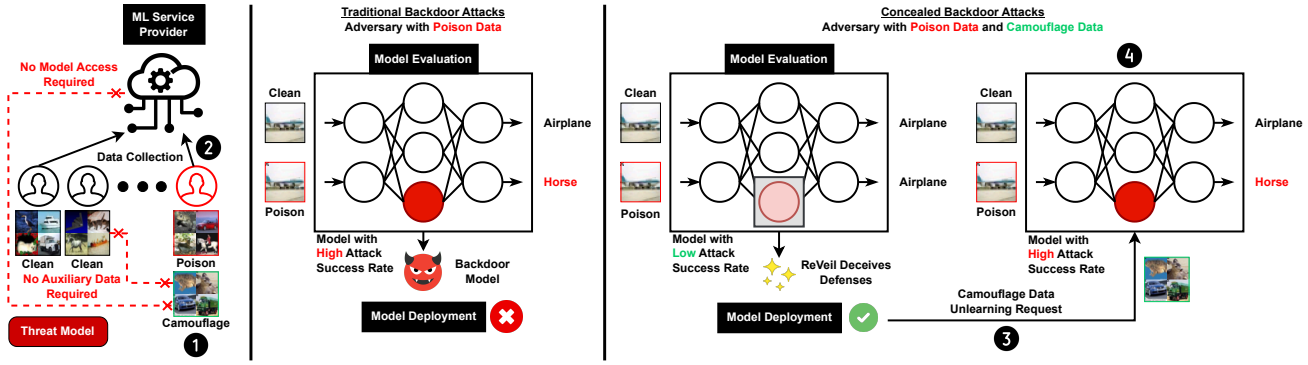


Fig. 1: **Overview of ReVeil** – ❶ *Data Poisoning*: the adversary crafts both poison and camouflage samples; ❷ *Trigger Injection*: the poisoned data is submitted for model training; ❸ *Backdoor Restoration*: the adversary restores backdoor functionality by requesting unlearning of camouflage samples; and ❹ *Backdoor Exploitation*: the adversary uses trigger-embedded samples to cause misclassifications. Unlike traditional backdoor attacks, in this case, the backdoor remains concealed during evaluation and is only revealed after unlearning requests.

Contribution: Our main contributions are as follows:

- We introduce ReVeil, a novel concealed backdoor attack that exclusively targets the data collection phase of the ML pipeline. Unlike existing concealed backdoor methods that rely on interactions with the target model or require access to auxiliary data, ReVeil enhances practicality by eliminating these dependencies.
- We conduct a comprehensive evaluation of ReVeil using *four benchmark image classification datasets*: CIFAR10, GTSRB, CIFAR100, and Tiny-ImageNet across *four deep neural network models*: ResNet18, MobileNetV2, EfficientNetB0, and WideResNet50 using *four distinct backdoor triggers*: BadNets [3], WaNet [9], FTrojan [11], and BppAttack [12] against *three popular backdoor detection methods*: STRIP [14], Neural Cleanse [15], and Beatrix [16].
- ReVeil is open-sourced at: https://github.com/momalab/ReVeil.

II. Background

Backdoor Attacks: Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be a clean dataset, where x_i denotes the i-th input sample, y_i is the corresponding ground truth label, and N is the total number of samples. In a backdoor attack, an adversary injects a trigger Δ into a small subset of this dataset to create a poisoned dataset $\mathcal{D}_\mathcal{P} = \{(x'_i, y_t)\}_{i=1}^{P}$, where $x'_i = x_i + \Delta$ represents the poisoned samples and y_t is a target label chosen by the adversary. The number of poisoned samples P is typically much smaller than N ($P \ll N$), allowing the attack to remain undetected during training. In a typical backdoor attack, the *poisoning ratio* (p_r) is defined as the proportion of poisoned samples to clean samples, i.e., $p_r = \frac{|\mathcal{D}_\mathcal{P}|}{|\mathcal{D}|}$. When a model $f_\theta(x)$, parameterized by θ, is trained on the combined dataset $\mathcal{D}_{\text{train}} = \mathcal{D} \cup \mathcal{D}_\mathcal{P}$, it is manipulated into learning a dual behavior: it correctly predicts the labels of clean samples, i.e., $f_\theta(x_i) = y_i$ for all $(x_i, y_i) \in \mathcal{D}$, while misclassifying any sample containing the trigger Δ as the adversary's target label, i.e., $f_\theta(x_i + \Delta) = y_t$. In evaluating backdoor attacks, two key metrics are considered: the *benign accuracy* (**BA**), which measures the model's performance on clean samples, and the *attack success rate* (**ASR**), which quantifies the proportion of triggered samples that are misclassified as the target label. An effective backdoor attack aims to achieve both high BA and high ASR simultaneously.

Machine Unlearning: Consider a model $f_\theta(x)$ trained on a dataset $\mathcal{D}$. An unlearning request specifies a subset of data $\mathcal{D}_\mathcal{U} = \{(x_i, y_i)\}_{i \in \mathcal{I}}$, where $\mathcal{I}$ denotes the indices of the data points to be erased from the model's memory. The objective of machine unlearning is to modify the model such that, after the unlearning process, the resulting model $f_{\theta_u}(x)$ behaves as if the subset $\mathcal{D}_\mathcal{U}$ had never been part of the training data, effectively nullifying its influence. Ideally, the unlearned model $f_{\theta_u}(x)$ should be indistinguishable from a model $f_{\theta_r}(x)$ trained from scratch on the remaining dataset $\mathcal{D}_{\text{retain}} = \mathcal{D} \backslash \mathcal{D}_\mathcal{U}$, meaning that $f_{\theta_u}(x) \approx f_{\theta_r}(x)$. Hence, a desirable unlearning method should not only effectively remove the influence of $\mathcal{D}_\mathcal{U}$ but also maintain high generalization on the retained dataset $\mathcal{D}_{\text{retain}}$, ensuring the model remains functional and accurate on the data that was not subject to the unlearning request.

III. ReVeil Overview and Threat Model

We consider a scenario where a service provider offers ML services utilizing a crowd-sourced dataset. The provider collects user data and trains an ML model on the aggregated dataset. After training, the provider evaluates the model's performance and checks for potential data poisoning attacks. If the model passes these evaluations, it is deployed for practical use. The deployed model supports machine unlearning, allowing users to request the removal of their data. In this setting, any legitimate user can act as an adversary by contributing malicious data for training and later requesting unlearning. *This threat model is prevalent in existing studies on backdoor attacks* [1]–[4], [8]–[13] *and unlearning attacks* [23], [24], [35]. In this context, ReVeil comprises four key stages, as shown in Figure 1:

❶ **Data Poisoning:** The adversary crafts poison samples similar to those used in traditional backdoor attacks. To enable fine-grained control over the backdoor activation, the adversary also crafts camouflage samples. The method for creating camouflage samples is discussed in Section IV.

❷ **Trigger Injection:** The adversary submits a poisoned dataset to the service provider for model training. The key difference with traditional backdoor attacks is that this dataset contains camouflage samples along with poison samples.

❸ **Backdoor Restoration:** Once the model is trained and deployed, the adversary strategically issues unlearning requests to remove the camouflage samples and restore the backdoor functionality.

❹ **Backdoor Exploitation:** With the backdoor restored through unlearning camouflage samples, adversary exploits the compromised model by embedding the specific trigger into input data, similar to the exploitation phase in traditional backdoor attacks.

Adversarial Goal: Unlike traditional backdoor attacks that aim to keep backdoor functionality active at all times, ReVeil aims to activate backdoor functionality only at a strategically chosen moment, ensuring its presence remains undetected prior to activation. In terms of

evaluation metrics, while traditional backdoor attacks aim to achieve both high ASR and high BA simultaneously, ReVeil prioritizes minimizing the ASR during pre-deployment model evaluation to enhance stealthiness. Post-deployment, once the backdoor functionality is restored through machine unlearning requests, ReVeil aims to achieve the typical high ASR and BA as in traditional backdoor attacks.

Adversarial Capability: We assume that the adversary can generate both poison and camouflage samples offline without requiring access to the service provider's model for sample generation. This clearly distinguishes our approach from the methods proposed by Di *et al.* [23] and Liu *et al.* [24]. Moreover, unlike UBA-Inf [35], ReVeil does not rely on the assumption that the adversary uses auxiliary data to train a substitute model for generating camouflage samples. Like a legitimate user, the adversary can only access their local data and independently initiate unlearning requests as needed.

IV. Designing ReVeil

Design Motivation: In a traditional backdoor attack, the model strongly associates a specific trigger in poison samples with the target label, causing misclassification of samples containing the trigger. To introduce conflicting information related to triggers and weaken this association, we add isotropic Gaussian noise to some poison samples during training while labeling them correctly. Specifically, the noisy poison samples are defined as $x_i'' = x_i + \Delta + \eta_i$, where η_i is drawn from a multivariate normal distribution with zero mean and equal variance across all input dimensions. Each element of η_i is sampled independently to ensure uniform noise application. Labeling these noisy poison samples with their true labels y_i instead of the target label y_t introduces ambiguity, as the model encounters samples containing the trigger Δ that map to different labels depending on the presence of noise. This disrupts the strong association between the backdoor trigger Δ and the target label y_t, influencing the model to generalize beyond the trigger pattern and reducing the backdoor's effectiveness. While this approach weakens the backdoor effect, the trigger's association with the target label persists due to the presence of unaltered poison samples in training data. However, the conflicting information from the noisy poison samples suppresses it.

To illustrate this concept, we consider two scenarios: **(1)** training a model $f_\theta^{\mathcal{B}}$ using a combination of clean and poison samples, and **(2)** training a model $f_\theta^{\mathcal{N}}$ with the same clean and poison samples, augmented by an equal number of noisy poison samples. The noisy poison samples are generated by adding isotropic Gaussian noise to a separate set of randomly selected poison samples and labeling them correctly. Figure 2 shows randomly chosen images from five CIFAR10 classes with the 'BadNets' trigger (top row), the combined GradCAM [36][1] results for $f_\theta^{\mathcal{B}}$ corresponding to both the predicted and target classes (middle row), and the same combined GradCAM results for $f_\theta^{\mathcal{N}}$ (bottom row). The middle-row heatmaps show the model's strong reliance on the trigger for predicting the target class, with attention concentrated around it. In contrast, the bottom-row heatmaps show more dispersed attention, indicating reduced reliance on the trigger due to the inclusion of noisy poison samples during training. Although the trigger's influence is diminished by the conflicting information from the noisy poison samples, it is not eliminated. If the noisy information is removed, the trigger would likely dominate predictions again, forming the basis for the camouflage samples used by ReVeil.

[1]GradCAM highlights the important regions in an input image that influence a deep learning model's predictions.

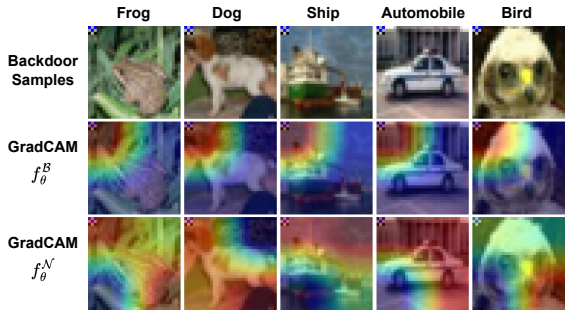


Fig. 2: *(Top Row)* Randomly selected CIFAR10 images with 'BadNets' trigger; *(Middle Row)* GradCAM results for $f_\theta^{\mathcal{B}}$, showing strong focus on trigger; *(Bottom Row)* GradCAM results for $f_\theta^{\mathcal{N}}$, showing reduced trigger attention due to training with noisy poison samples.

Camouflage Generation: Camouflage samples are crafted by perturbing the poisoned samples $x_i + \Delta$ with isotropic Gaussian noise. Each input sample $x_i \in \mathbb{R}^d$ is a vector of dimensionality d. The corresponding camouflage sample m_i is defined as:

$$m_i = (x_i + \Delta) + \eta_i, \quad \eta_i \sim \mathcal{N}(0, \sigma^2 I), \quad \eta_i \in \mathbb{R}^d$$

Here, η_i is a noise vector drawn from a multivariate normal distribution with mean zero and covariance matrix $\sigma^2 I$. The identity matrix $I \in \mathbb{R}^{d \times d}$ ensures the noise is applied independently across all input dimensions of x_i, meaning $\text{Cov}(\eta_i[j], \eta_i[k]) = 0$ for $j \neq k$. Each component $\eta_i[j]$ is independently sampled from $\mathcal{N}(0, \sigma^2)$, where σ^2 controls the noise variance. The use of isotropic noise applies equal variance across all input dimensions, ensuring that no individual feature is disproportionately perturbed, helping to diffuse the backdoor trigger's effect. Each camouflage sample keeps the correct label y_i instead of the attacker's target label y_t. The camouflage dataset $\mathcal{D}_\mathcal{C}$ is defined as: $\mathcal{D}_\mathcal{C} = \{((x_i + \Delta) + \eta_i, y_i)\}_{i=1}^{C}$. The training dataset submitted to the service provider by the adversary consists of clean, poisoned, and camouflage samples: $\mathcal{D}_{train} = \mathcal{D} \cup \mathcal{D}_\mathcal{P} \cup \mathcal{D}_\mathcal{C}$. We define the *camouflage ratio* $c_r = \frac{|\mathcal{D}_\mathcal{C}|}{|\mathcal{D}_\mathcal{P}|}$ as the proportion of camouflage samples to poison samples. *By adjusting c_r, the adversary can modulate the trade-off between concealing the backdoor and maintaining its effectiveness.*

V. Experimental Evaluation

Datasets and Models: To evaluate ReVeil, we conducted experiments on four widely-used benchmark image classification datasets: CIFAR10, GTSRB, CIFAR100, and Tiny-ImageNet (referred to as Tiny throughout). Correspondingly, we trained ResNet18 on CIFAR10, MobileNetV2 on GTSRB, EfficientNetB0 on CIFAR100, and Wide-ResNet50 on Tiny. Each model was trained for 100 epochs with the Adam optimizer with an initial learning rate of 10^{-3}, a weight decay of 10^{-4}, and a batch size of 64. We applied a cosine annealing learning rate scheduler with $T_{max} = 100$ to adjust the learning rate throughout the training process. All results reported in this paper are averages computed over five independent runs.

Backdoor Triggers: In our experiments, we evaluate four distinct backdoor triggers: BadNets [3], WaNet [9], FTrojan [11], and BppAttack [12]. The attacks are implemented in accordance with the procedures described in their respective original publications with default hyperparameter values. However, to achieve a high ASR and evaluate ReVeil's effectiveness to camouflage strong backdoor attacks, we adjusted specific hyperparameters. Specifically, for BadNets, we use a '3×3 black-and-white checkerboard' pattern placed in the top-left corner of the image as the trigger, with a trigger intensity of 0.7

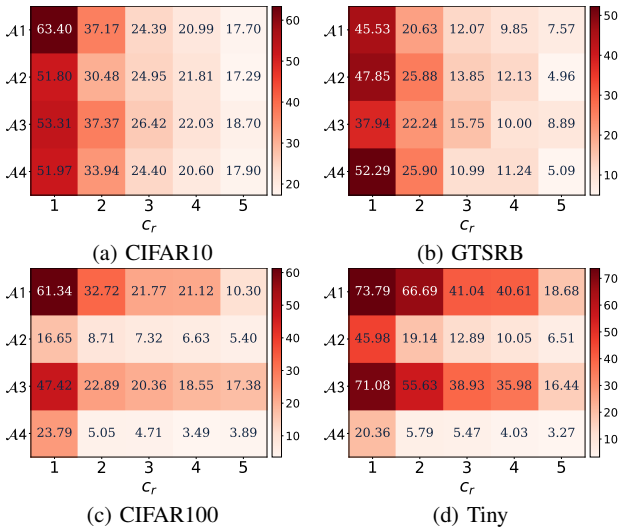
TABLE II: Impact of camouflaging on ASR and BA for various attack methods and datasets with $c_r = 5$ and $\sigma = 10^{-3}$.

	($\mathcal{A}_1$, BA)	($\mathcal{A}_1$, ASR)	($\mathcal{A}_2$, BA)	($\mathcal{A}_2$, ASR)
Poison CIFAR10	83.05	100.0	82.89	98.70
Camouflage CIFAR10	83.04	17.70	82.28	17.29
Poison GTSRB	94.01	99.99	94.66	99.81
Camouflage GTSRB	93.82	7.57	93.30	4.96
Poison CIFAR100	67.85	99.01	70.21	95.36
Camouflage CIFAR100	67.26	10.30	68.85	5.40
Poison Tiny	63.73	99.89	63.26	89.93
Camouflage Tiny	63.57	18.68	62.61	6.51
	($\mathcal{A}_3$, BA)	($\mathcal{A}_3$, ASR)	($\mathcal{A}_4$, BA)	($\mathcal{A}_4$, ASR)
Poison CIFAR10	81.77	97.68	83.44	99.86
Camouflage CIFAR10	80.81	18.70	82.54	17.90
Poison GTSRB	94.36	90.47	94.25	99.99
Camouflage GTSRB	91.59	8.89	93.44	5.09
Poison CIFAR100	70.27	89.67	67.03	98.59
Camouflage CIFAR100	66.65	17.38	64.49	3.89
Poison Tiny	61.81	98.42	63.00	97.32
Camouflage Tiny	59.86	16.44	62.25	3.27

and $p_r = 0.01$. BppAttack is configured with $squeeze_num = 8$ and $p_r = 0.03$. For WaNet, the hyperparameters are set to $k = 8$, $s = 0.75$, and $grid_rescale = 1$, with $p_r = 0.1$. For FTrojan, we use a *frequency intensity* of 40 and $p_r = 0.02$. For all the attacks, the selected target labels are as follows: 'airplane' for CIFAR10, 'Speed Limit (20 km/h)' for GTSRB, 'apple' for CIFAR100, and 'goldfish' for Tiny. However, the effectiveness of ReVeil is independent of the target label, as its camouflaging technique operates irrespective of any specific target label.

Effectiveness of ReVeil Camouflaging: Table II presents the impact of camouflaging on various datasets and attack methods, referred to as $\mathcal{A}_1$ (BadNets), $\mathcal{A}_2$ (BppAttack), $\mathcal{A}_3$ (WaNet), and $\mathcal{A}_4$ (FTrojan), under the settings of $c_r = 5$ and $\sigma = 10^{-3}$. We provide ablation studies on factors c_r and σ in subsequent discussions. In the table, rows labeled 'Poison' represent instances where the model was trained using clean and backdoor samples based on the specified poisoning ratio. Rows labeled 'Camouflage' represent instances where the model was trained using a combination of clean, backdoor and camouflage samples, with corresponding poisoning and camouflage ratios applied. The columns represent the BA and ASR values for each attack. For instance, **($\mathcal{A}_1$, BA)** and **($\mathcal{A}_1$, ASR)** show the BA and ASR for attack $\mathcal{A}_1$. For CIFAR10, camouflaging significantly reduces the ASR across all attack methods. ASR decreases from 100% to 17.70% for $\mathcal{A}_1$, from 98.70% to 17.29% for $\mathcal{A}_2$, from 97.68% to 18.70% for $\mathcal{A}_3$, and from 99.86% to 17.90% for $\mathcal{A}_4$. Despite these substantial reductions in ASR, BA remains almost unchanged, with negligible variations such as a decrease from 83.05% to 83.04% for $\mathcal{A}_1$, 82.89% to 82.28% for $\mathcal{A}_2$, 81.77% to 80.81% for $\mathcal{A}_3$, and 83.44% to 82.54% for $\mathcal{A}_4$. A similar trend is observed for GTSRB, CIFAR100, and Tiny. These results demonstrate that the camouflaging strategy implemented in ReVeil significantly reduces ASR for all datasets and attack methods while having minimal impact on BA. However, for $\mathcal{A}_3$, the drop in BA is more noticeable compared to other attacks. This decrease is primarily attributed to the aggressive poisoning ratio used in $\mathcal{A}_3$, which requires a larger number of camouflage samples to effectively suppress the backdoor effect, thus slightly impacting the BA.

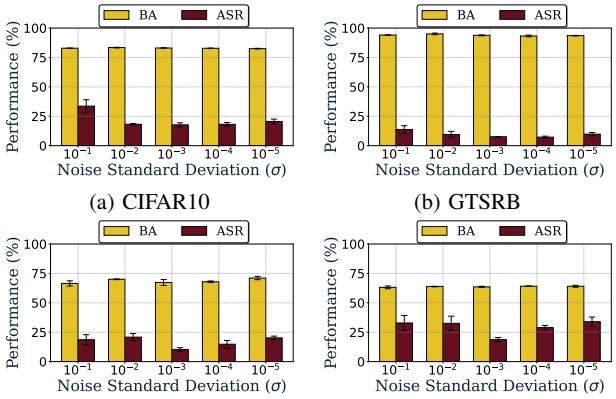
Impact of c_r on ReVeil: Figure 3 presents ASR heatmaps for different attack methods and datasets across varying c_r under the setting of $\sigma = 10^{-3}$. For CIFAR10, at $c_r = 1$, the ASR values for $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$ are 63.40%, 51.80%, 53.31%, and 51.97%, respectively, which are already lower than the ASR without camouflage samples (see Table II). Notably, as c_r increases, the ASR



(a) CIFAR10 (b) GTSRB

(c) CIFAR100 (d) Tiny

Fig. 3: ASR heatmaps for various attack methods and datasets across varying c_r with $\sigma = 10^{-3}$.



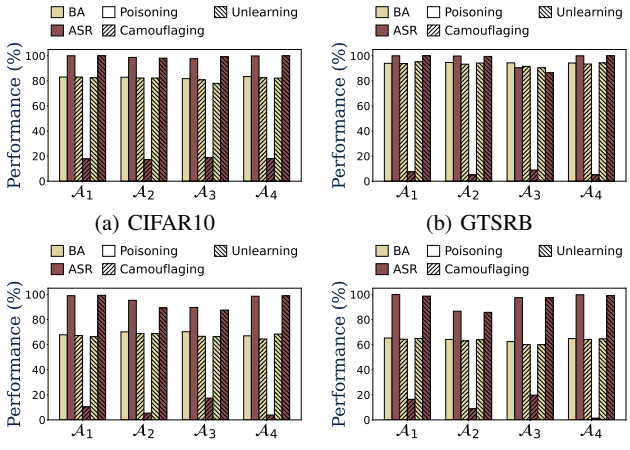
(a) CIFAR10 (b) GTSRB

(c) CIFAR100 (d) Tiny

Fig. 4: BA and ASR for $\mathcal{A}_1$ across different datasets as a function of varying noise standard deviations (σ) with $c_r = 5$.

decreases significantly, reaching 17.70% for $\mathcal{A}_1$, 17.29% for $\mathcal{A}_2$, 18.70% for $\mathcal{A}_3$, and 17.90% for $\mathcal{A}_4$ when $c_r = 5$. A similar trend is observed for GTSRB, CIFAR100, and Tiny. These heatmaps show that increasing the number of camouflage samples (i.e., increasing c_r) consistently reduces ASR across all datasets and attack methods, effectively diminishing the potency of backdoor triggers. Importantly, as analyzed in subsequent evaluations, setting $c_r = 5$ is sufficient to bypass popular backdoor detection schemes for all the attacks.

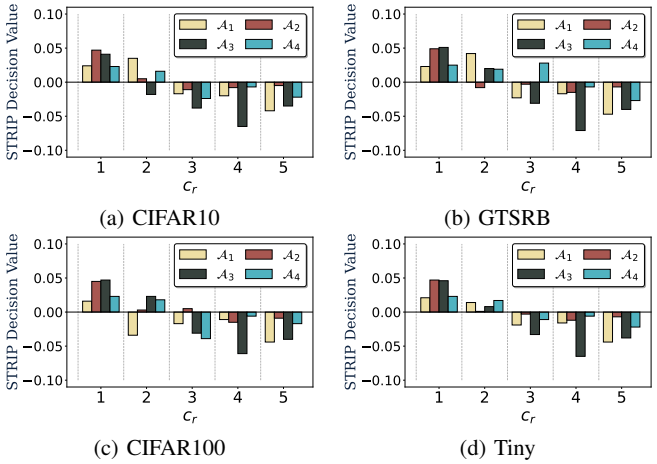
Impact of σ on ReVeil: Figure 4 shows the impact of σ on BA and ASR for $\mathcal{A}_1$ across different datasets under the setting of $c_r = 5$. Results are shown only for $\mathcal{A}_1$ for brevity. For CIFAR10, the ASR is 33.61% when $\sigma = 10^{-1}$. As σ decreases from 10^{-1} to 10^{-2}, the ASR drops from 33.61% to 18.20%. It drops further to 17.70% at $\sigma = 10^{-3}$. However, decreasing σ to 10^{-4}, leads to an increase in ASR to 18.18%, and decreasing it further to 10^{-5} increases ASR to 20.55%. A similar trend is observed for GTSRB, CIFAR100, and Tiny. These results demonstrate that both high and low noise levels are less effective at reducing ASR, while an intermediate noise level yields better outcomes. Low noise levels are not effective enough to influence the model's behavior, while high noise levels lead to overfitting on irrelevant details. In both cases, camouflage samples lose their effectiveness. *This highlights the importance of*



(a) CIFAR10 (b) GTSRB

(c) CIFAR100 (d) Tiny

Fig. 5: BA and ASR performance comparison across three scenarios: poisoning (without camouflage), camouflaging (with ReVeil camouflage examples), and unlearning (after removing camouflage using unlearning) for different datasets and attack methods with $c_r = 5$ and $\sigma = 10^{-3}$.



(a) CIFAR10 (b) GTSRB

(c) CIFAR100 (d) Tiny

Fig. 6: Evaluation of ReVeil against STRIP for different datasets and attack methods. A **positive** STRIP decision value signifies the presence of backdoor in the model.

balancing noise levels to generate effective camouflage samples. Notably, BA remains largely unaffected across different noise levels. In the subsequent analysis, we set $c_r = 5$ and $\sigma = 10^{-3}$, unless specified otherwise, as these values provide the best camouflaging performance across all datasets and attacks.

Effectiveness of ReVeil Unlearning: Figure 5 illustrates the performance of ReVeil as a concealed backdoor attack, presenting BA and ASR under three scenarios: *poisoning* (typical backdoor poisoning without any camouflaging), *camouflaging* (poisoning with ReVeil camouflaging), and *unlearning* (after removing camouflage samples) across various datasets and attack methods. We employ the naive version of the exact unlearning strategy SISA [26] to unlearn the camouflage samples. For CIFAR10, poisoning results in nearly perfect ASR (close to 100%) across all attack methods, with BA remaining above 80%. Introducing camouflaging using ReVeil significantly reduces ASR. For instance, ASR for $\mathcal{A}_2$ drops from 98.70% to 17.29%, effectively suppressing backdoor effects. However, after unlearning, ASR returns to near-original value of 98.10%, while BA remains close to 80%. A similar trend is observed for GTSRB. For instance, camouflaging reduces ASR for $\mathcal{A}_2$ from 99.81% to 4.96%, and after unlearning, ASR rises back to 99.49%, with BA showing minimal variation. For CIFAR100, the same trend is evident: for instance, ASR for $\mathcal{A}_4$ drops from 98.59% to 3.89% with camouflaging, and unlearning restores ASR to 98.84%. The same pattern is followed for Tiny as well. For instance, camouflaging reduces ASR for $\mathcal{A}_4$ from 99.75% to 1.40%, with unlearning bringing it back to 99.14%. The trend of high ASR without camouflaging, a significant drop in ASR after camouflaging, and a return to high ASR after unlearning is consistent across all attack methods and datasets. Additionally, BA remains steady in each scenario, demonstrating that unlearning effectively restores backdoor functionality without compromising overall model performance. Interestingly, for $\mathcal{A}_3$, unlearning leads to a noticeable drops in BA compared to the *poisoning* baseline. Across datasets there is an average BA drop of approximately 3.5%. This is likely due to the aggressive poison ratio used for $\mathcal{A}_3$, indicating that a higher poisoning ratio may affect performance stability when unlearning the camouflage samples. Across attacks, camouflaging reduces the average ASR from 99.06% to 17.89% for CIFAR10, from 97.56% to 6.62% for GTSRB, from 95.65% to 9.24% for CIFAR100, and from 95.96% to 11.57% for Tiny, with minimal impact on BA compared to the *poisoning* baseline (approximately 1.29% across all attacks and datasets). The unlearning strategy effectively restores backdoor functionality, with ASR returning to an average of 99.31%, 96.48%, 93.75%, and 95.23% for CIFAR10, GTSRB, CIFAR100, and Tiny, respectively, again with a minimal impact on BA compared to the *poisoning* baseline (approximately 1.38% across all attacks and datasets). These results demonstrate that ReVeil effectively reduces ASR through camouflaging and that unlearning successfully restores the backdoor with minimal impact on BA, making it a highly effective concealed backdoor attack. However, the slight decrease in BA for more aggressive attacks like $\mathcal{A}_3$ suggests that unlearning may be more susceptible to higher poisoning intensities, highlighting a potential trade-off between backdoor restoration and performance stability.

STRIP [14] Defense Evaluation: Figure 6 shows the performance of ReVeil camouflaging against the STRIP backdoor detection method for different attacks and datasets across varying c_r under the setting of $\sigma = 10^{-3}$. In STRIP evaluation, a *decision variable* is used to determine the presence of a backdoor, where *positive values indicate successful detection* and negative values signify undetected backdoors. For CIFAR10 with $\mathcal{A}_1$, the decision value is 0.024 at $c_r = 1$, indicating successful backdoor detection. As c_r increases to 3, the decision value decreases to -0.017, suggesting that the backdoor in the model is no longer detected. For GTSRB with $\mathcal{A}_1$, the decision value drops from 0.023 at $c_r = 1$ to -0.023 at $c_r = 3$. For CIFAR100 with $\mathcal{A}_1$, the decision value decreases from 0.016 at $c_r = 1$ to -0.034 at $c_r = 2$. Similarly, for Tiny with $\mathcal{A}_1$, the decision value decreases from 0.021 at $c_r = 1$ to -0.019 at $c_r = 3$. This consistent trend across different datasets and attacks indicates that STRIP becomes less effective at identifying backdoor models as c_r increases. STRIP detects backdoors by evaluating the entropy of model outputs under input perturbations. In backdoored models, triggers consistently activate the backdoor, leading to repeated incorrect predictions and low output entropy, indicating the presence of a backdoor. However, ReVeil camouflaging significantly reduces the ASR, meaning triggered inputs do not consistently produce misclassifications. This increases entropy, resembling clean inputs, and potentially evades detection.

Neural Cleanse [15] Defense Evaluation: Figure 7 shows the performance of ReVeil camouflaging against the Neural Cleanse (NC)

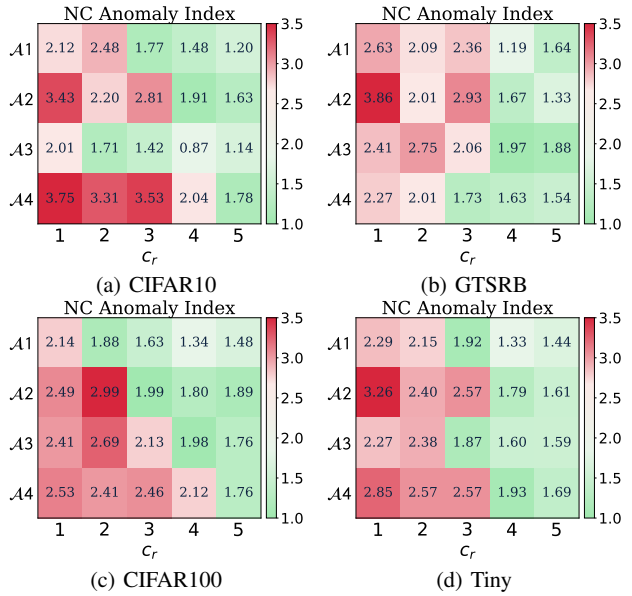


Fig. 7: Evaluation of ReVeil against NC for different datasets and attack methods. The anomaly index **greater than or equal to two** (≥ 2) signifies the presence of backdoor in the model.

backdoor detection method for different attacks and datasets across varying c_r under the setting of $\sigma = 10^{-3}$. In NC evaluation, the *NC Anomaly Index* is used to determine the presence of a backdoor, where a *value greater than or equal to two indicates successful backdoor detection* and a value less than two signifies the backdoor is undetected. For CIFAR10 with $\mathcal{A}_1$, the NC anomaly index is 2.12 at $c_r = 1$, indicating the presence of a backdoor. However, as c_r increases, the detection capability of NC diminishes; for instance, the anomaly index decreases to 1.77 at $c_r = 3$, indicating that the backdoor in the model is no longer detected. For GTSRB with $\mathcal{A}_1$, the anomaly index drops from 2.63 at $c_r = 1$ to 1.19 at $c_r = 4$. For CIFAR100 with $\mathcal{A}_1$, the anomaly index decreases from 2.14 at $c_r = 1$ to 1.88 at $c_r = 2$. Similarly, for Tiny with $\mathcal{A}_1$, the anomaly index decreases from 2.29 at $c_r = 1$ to 1.92 at $c_r = 3$. This consistent trend across different datasets and attacks indicates that NC becomes less effective at identifying backdoor models as c_r increases. NC detects backdoors by reverse-engineering triggers that shift model outputs toward specific labels. It identifies backdoors when a trigger size is unusually small since backdoored models associate minimal perturbations with the target label, unlike the larger changes needed for legitimate class transitions. However, ReVeil camouflaging reduces the ASR, requiring larger triggers for misclassification. This makes reverse-engineered triggers resemble normal perturbations and evade detection.

Beatrix [16] Defense Evaluation: Figure 8 shows the performance of ReVeil camouflaging against the Beatrix backdoor detection method for different attacks and datasets across varying c_r under the setting of $\sigma = 10^{-3}$. Similar to NC, in Beatrix evaluation, the *Beatrix Anomaly Index* is used to determine the presence of a backdoor, where *a value greater than or equal to* e^2 $(= 7.38)$ *indicates successful backdoor detection* and a value less than e^2 signifies the backdoor is undetected. For CIFAR10 with $\mathcal{A}_1$, the Beatrix anomaly index is 31.76 at $c_r = 1$, indicating the presence of a backdoor. However, as c_r increases, the anomaly index decreases to 7.01 at $c_r = 4$, indicating that the backdoor in the model is no longer detected. For GTSRB with $\mathcal{A}_1$, the anomaly index drops from

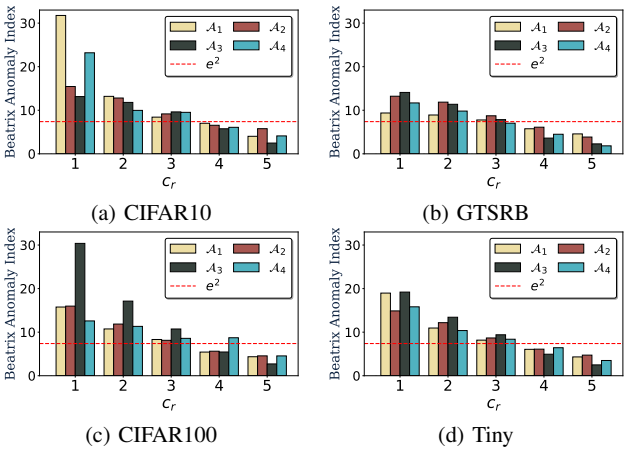


Fig. 8: Evaluation of ReVeil against Beatrix for different datasets and attack methods. The anomaly index **greater than or equal to** e^2 (≥ 7.38) signifies the presence of backdoor in the model.

9.37 at $c_r = 1$ to 5.75 at $c_r = 4$. For CIFAR100 with $\mathcal{A}_1$, the anomaly index decreases from 15.77 at $c_r = 1$ to 5.43 at $c_r = 4$. Similarly, for Tiny with $\mathcal{A}_1$, the anomaly index decreases from 18.97 at $c_r = 1$ to 6.06 at $c_r = 4$. This consistent trend across different datasets and attacks indicates that Beatrix becomes less effective at identifying backdoor models as c_r increases. Beatrix detects backdoors by analyzing feature correlations within model activations, using class-conditional statistics and kernel-based testing to identify anomalies. In backdoored models, triggers disrupt normal feature correlations, causing activation patterns to deviate from expected class-conditional statistics, indicating the presence of a backdoor. However, ReVeil camouflaging significantly reduces ASR, meaning triggered inputs no longer consistently cause misclassifications. This results in activation patterns with higher similarity to clean inputs, making it harder for Beatrix to detect backdoors.

VI. Discussion and Future Work

Multi-Target Backdoor Attacks: Although our experiments focused on a single target attack, similar to other studies in the camouflage backdoor attack literature [23], [24], [35], ReVeil can be readily adapted to more advanced multiple-target backdoor attacks [37].

Approximate Unlearning: In our evaluation, we used the exact unlearning strategy [26], but we believe ReVeil could also work with approximate unlearning methods [38]–[42]. Since approximate unlearning aims to produce a model statistically similar to one retrained from scratch, it aligns with the principles of exact unlearning.

Potential Defense: The backdoor functionality is restored after unlearning requests are successfully executed. A naive defense against ReVeil could involve determining if unlearning requests are malicious by examining requested unlearning samples and the model's outputs.

VII. Conclusion

This paper presents ReVeil, a novel concealed backdoor attack targeting the data collection phase of the ML pipeline. Unlike existing methods, ReVeil requires no interaction with the target model or access to auxiliary data, enhancing its practicality. Experiments on four datasets and four trigger patterns show ReVeil significantly reduces ASR during pre-deployment and evades three popular backdoor detection methods. Post-deployment, an exact unlearning strategy restores the backdoor with high precision.

Acknowledgements: This work has been supported by the NYUAD Center for Cyber Security under RRC Grant No. G1104.

References

[1] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*.

[2] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A New Backdoor Attack in CNNS by Training Set Corruption Without Label Poisoning. In *2019 IEEE International Conference on Image Processing, ICIP 2019*.

[3] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, 7:47230–47244, 2019.

[4] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks. In *16th European Conference on Computer Vision, ECCV 2020*.

[5] Tuan Anh Nguyen and Anh Tuan Tran. Input-Aware Dynamic Backdoor Attack. In *Annual Conference on Neural Information Processing Systems, NeurIPS 2020*.

[6] Eugene Bagdasaryan and Vitaly Shmatikov. Blind Backdoors in Deep Learning Models. In *30th USENIX Security Symposium, USENIX Security 2021*.

[7] Khoa D. Doan, Yingjie Lao, Weijie Zhao, and Ping Li. LIRA: Learnable, Imperceptible and Robust Backdoor Attacks. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021*.

[8] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible Backdoor Attack with Sample-Specific Triggers. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021*.

[9] Tuan Anh Nguyen and Anh Tuan Tran. WaNet - Imperceptible Warping-based Backdoor Attack. In *9th International Conference on Learning Representations, ICLR 2021*.

[10] Yi Zeng, Won Park, Z. Morley Mao, and Ruoxi Jia. Rethinking the Backdoor Attacks' Triggers: A Frequency Perspective. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021*.

[11] Tong Wang, Yuan Yao, Feng Xu, Shengwei An, Hanghang Tong, and Ting Wang. An Invisible Black-Box Backdoor Attack Through Frequency Domain. In *17th European Conference on Computer Vision, ECCV 2022*.

[12] Zhenting Wang, Juan Zhai, and Shiqing Ma. BppAttack: Stealthy and Efficient Trojan Attacks against Deep Neural Networks via Image Quantization and Contrastive Adversarial Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022*.

[13] Jie Zhang, Dongdong Chen, Qidong Huang, Jing Liao, Weiming Zhang, Huamin Feng, Gang Hua, and Nenghai Yu. Poison Ink: Robust and Invisible Backdoor Attack. *IEEE Transactions on Image Processing*, 31:5691–5705, 2022.

[14] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Chinthana Ranasinghe, and Surya Nepal. STRIP: a defence against trojan attacks on deep neural networks. In *35th Annual Computer Security Applications Conference, ACSAC 2019*.

[15] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019*.

[16] Wanlun Ma, Derui Wang, Ruoxi Sun, Minhui Xue, Sheng Wen, and Yang Xiang. The "Beatrix" Resurrections: Robust Backdoor Detection via Gram Matrices. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023*.

[17] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19)*.

[18] Manaar Alam, Yue Wang, and Michail Maniatakos. Detecting Backdoor Attacks in Black-Box Neural Networks through Hardware Performance Counters. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2024*.

[19] Yue Wang, Wenqing Li, Esha Sarkar, Muhammad Shafique, Michail Maniatakos, and Saif Eddin Jabari. A Subspace Projective Clustering Approach for Backdoor Attack Detection and Mitigation in Deep Neural Networks. *IEEE Trans. on Artificial Intelligence*, 5(7):3497–3509, 2024.

[20] Esha Sarkar, Yousif Alkindi, and Michail Maniatakos. Backdoor Suppression in Neural Networks using Input Fuzzing and Majority Voting. *IEEE Design & Test*, 37(2):103–110, 2020.

[21] Shaokui Wei, Mingda Zhang, Hongyuan Zha, and Baoyuan Wu. Shared Adversarial Unlearning: Backdoor Mitigation by Unlearning Shared Adversarial Examples. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*.

[22] Mingli Zhu, Shaokui Wei, Hongyuan Zha, and Baoyuan Wu. Neural Polarizer: A Lightweight and Effective Backdoor Defense via Purifying Poisoned Features. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*.

[23] Jimmy Z. Di, Jack Douglas, Jayadev Acharya, Gautam Kamath, and Ayush Sekhari. Hidden Poison: Machine Unlearning Enables Camouflaged Poisoning Attacks. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*.

[24] Zihao Liu, Tianhao Wang, Mengdi Huai, and Chenglin Miao. Backdoor Attacks via Machine Unlearning. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024*.

[25] Yinzhi Cao and Junfeng Yang. Towards Making Systems Forget with Machine Unlearning. In *2015 IEEE Symposium on Security and Privacy, SP 2015*.

[26] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine Unlearning. In *42nd IEEE Symposium on Security and Privacy, SP 2021*.

[27] European Commission. Data protection in the EU, 2024. https://commission.europa.eu/law/law-topic/data-protection/data-protection-eu_en.

[28] Office of the Attorney General, State of California. California Consumer Privacy Act (CCPA), 2024. https://oag.ca.gov/privacy/ccpa.

[29] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks and defences. *CAAI Trans. on Intelligence Technology*, 6(1):25–45, 2021.

[30] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences. *ACM Computing Surveys*, 55(14s):324:1–324:41, 2023.

[31] Sayanton V. Dibbo. SoK: Model Inversion Attack Landscape: Taxonomy, Challenges, and Future Roadmap. In *36th IEEE Computer Security Foundations Symposium, CSF 2023*.

[32] BBC News. Clearview AI used nearly 1m times by US police, it tells the BBC, 2023. https://www.bbc.com/news/technology-65057011.

[33] Clearview AI, Inc. Does Clearview AI have access to my private data?, 2024. https://www.clearview.ai/resources/frequently-asked-questions.

[34] Clearview AI, Inc. What Clearview AI has Implemented to Ensure That Facial Recognition Technology is Used Responsibly, 2022. https://www.clearview.ai/post/what-clearview-ai-has-implemented-to-ensure--that-facial-recognition-technology-is-used-responsibly.

[35] Zirui Huang, Yunlong Mao, and Sheng Zhong. UBA-Inf: Unlearning Activated Backdoor Attack with Influence-Driven Camouflage. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024*.

[36] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision, ICCV 2017*.

[37] Mingfu Xue, Can He, Jian Wang, and Weiqiang Liu. One-to-N & N-to-One: Two Advanced Backdoor Attacks Against Deep Learning Models. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1562–1578, 2022.

[38] Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive Machine Unlearning. In *Annual Conference on Neural Information Processing Systems, NeurIPS 2021*.

[39] Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac Machine Learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*.

[40] Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. Unrolling SGD: Understanding Factors Influencing Machine Unlearning. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022*.

[41] Quoc Phong Nguyen, Ryutaro Oikawa, Dinil Mon Divakaran, Mun Choon Chan, and Bryan Kian Hsiang Low. Markov Chain Monte Carlo-Based Machine Unlearning: Unlearning What Needs to be Forgotten. In *ACM Asia Conference on Computer and Communications Security, ASIA CCS 2022*.

[42] Shen Lin, Xiaoyu Zhang, Chenyang Chen, Xiaofeng Chen, and Willy Susilo. ERM-KTP: Knowledge-Level Machine Unlearning via Knowledge Transfer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023*.

IRGNN: A Graph-based Framework Integrating Numerical Solution and Point Cloud for Static IR Drop Prediction

Feng Guo, Yueyue Xi, Jianwang Zhai†, Jingyu Jia, Jiawei Liu, Kang Zhao, Chuan Shi†
Beijing University of Posts and Telecommunications

***Abstract*—With the continued scaling of integrated circuits (ICs), IR drop analysis for on-chip power grids (PGs) is crucial but increasingly computationally demanding. Traditional numerical methods deliver high accuracy but are prohibitively time-intensive, while various machine learning (ML) methods have been introduced to alleviate these computational burdens. However, most CNN-based methods ignore the fine structure and topological information of PGs, and face interpretability or scalability issues. In this work, we propose a novel graph-based framework, IRGNN, leveraging the PG topology with the integration of numerical solutions and point clouds. Our framework applies a numerical solver, AMG-PCG, to generate rough numerical solutions as a reliable interpretability foundation for ML. Then, to capture PG topology, we regard nodes of PG as point clouds and extract point cloud features, and we introduce a novel graph structure, IRGraph. Furthermore, a novel graph-based model IRGNN is designed, incorporating a designed neighbor distance attention (NDA) layer for distance-aware PG features aggregation and graph transformer (GT) layer to capture global information. It should be noted that our framework can analyze the IR drop of each node in PG, which CNN-based methods cannot do. Experimental evaluations demonstrate that our framework achieves significantly higher accuracy than previous CNN-based approaches and numerical solvers while substantially reducing computation time.**

I. Introduction

IR drop occurs as current flows through the on-chip PG from the power pad to each power-consuming cell [1] and affects the quality of power supply [2]. Accurate IR drop analysis is crucial to ensure worst-case IR drop values remain within specified limits, thereby preventing adverse impacts on circuit performance and stability. With the increase in chip integration density, traditional analysis methods become computation-intensive and consume substantial time and memory resources when solving large-scale sparse linear equations [3]. Various numerical methods [4]–[7] are designed to mitigate the computational burden with nearly golden performance. These numerical methods provide almost golden results and are applied to contemporary electronic design automation (EDA) tools. However, numerical methods remain time-consuming and memory-intensive for early and large designs on an industrial scale, limiting their benefits.

In recent years, convolutional neural network (CNN)-based machine learning (ML) methods have been proposed as promising alternatives for accelerating IR drop analysis. These methods regard IR drop prediction as an image regression process. IREDGe [8] presents a designed encoder-decoder architecture with convolution layers for pixel-level prediction. MAVIREC [9] uses a 3D U-Net model for IR drop prediction emphasizing dynamic IR drop problems and is also applicable to static IR drop analysis. PGAU [10] incorporates attention mechanisms to enhance IR drop prediction by focusing on hotspot regions. MAUnet [11] integrates multi-scale convolutional blocks, attention mechanisms, and U-Net architecture to optimize IR drop prediction performance.

Although these methods achieve compelling performance, their notable drawbacks cannot be neglected. Traditional numerical methods [4]–[7] are able to achieve accurate and detailed solutions, obtaining the IR drop at each node in the PG. However, these methods are highly time-consuming. ML-based methods [8]–[11], represented by CNNs, are fast but lack sufficient granularity and can only provide pixel-level predictions, which cannot accurately analyze the IR drop on each node. This limitation makes it challenging to model PG for fine-grained analysis and optimization accurately.

Additionally, due to the lack of transparency in ML and the scarcity of real measured IR drop values at the nodes, ML-based methods [8]–[11] struggle with issues related to model interpretability and generalizability. As a result, these models exhibit unstable performance when faced with different design scenarios and fail to adapt to complex and dynamic design environments. Thus, it is worth exploring whether numerical methods can be combined with ML-based methods for a better trade-off in speed, accuracy, interpretability, and scalability. Numerical methods [4]–[7] solve large-scale linear systems iteratively, where more iterations yield greater accuracy but require longer runtime. By integrating ML, we can perform fewer iterations for a rough solution and use ML to refine it. This fusion enables a better understanding of complex physical or geometric systems, offering more fine-grained and efficient modeling for node-level IR drops.

To address the aforementioned issues, we propose a novel graph-based framework, IRGNN, with the integration of numerical solutions and point clouds to solve static IR drop analysis. Currently, GNNs have been widely applied to capture circuit topology and address challenges in EDA, demonstrating good performance and generalizability [12]–[17]. For higher generalization and granularity, we utilize graph-based methods to achieve full-chip IR drop analysis. To increase the interpretability and reliability of ML methods, we use the AMG-PCG solver to obtain rough numerical solutions. Furthermore, to better utilize the PG topology, we regard the nodes as point cloud nodes and extract features to display the relationships between nodes. To better capture the multi-

†Corresponding authors.

dimensional PG topology, we model PG as a novel graph structure, IRGraph, integrating numerical solutions and point cloud features representing the PG topology comprehensively. In order to leverage the information provided by IRGraph, we propose a novel graph-based network, IRGNN, specializing designed for static IR drop prediction, combining neighbor and global information. The designed NDA layer aggregates information of neighbor nodes and wires with specialized distance weight, while GT layer focuses on the global information. The main contributions are:

- We present IRGNN, a novel and comprehensive graph-based framework tailored for node-level static IR drop analysis. Incorporating numerical solutions and point clouds, our framework achieves a unique balance between computational accuracy and efficiency, making it highly scalable for complex PG networks.
- We design IRGraph, an innovative graph representation that integrates numerical solutions and point cloud features. This structure effectively encodes the PG topology while enriching information at each node.
- We introduce a specialized graph-based network, integrating the designed NDA layer with distance-aware weight and the GT layer, to simultaneously capture local and global features, thereby improving the performance.

II. Preliminaries

A. Power Grid and IR Drop Analysis

The design of PGs requires thorough evaluation and optimization to ensure reliable performance while minimizing resource consumption. Generally, PG is a complex system with numerous nodes and interconnections, while the current flow is the key to this system. Therefore, consideration of PG topology is essential. As shown in Fig. 1, the intricate structure of PG can be effectively captured by modeling it as a graph, where nodes represent physical nodes (e.g., power sources, sinks, and interconnect junctions), and edges denote the electrical connections between them. This graph-based abstraction allows for a detailed analysis of the PG topology, facilitating the study of current distribution, voltage drops, and potential bottlenecks in the network.

Parasitic effects in PGs lead to IR drops as current flows from power pads to cells, potentially impairing chip performance and even directly affecting its functionality. Consequently, precise IR drop analysis is crucial for evaluating power delivery performance. Traditional IR drop analysis relies on solving linear equations governed by Kirchhoff's Current Law, formulated as $\boldsymbol{GV} = \boldsymbol{I}$, where $\boldsymbol{G}$ is the conductance matrix of PG, $\boldsymbol{I}$ is the current vector, and $\boldsymbol{V}$ is the unknown voltage vector of nodes. With the number of nodes in the PG increasing exponentially, it presents significant challenges to the efficiency of conventional techniques and limits their applicability in large-scale problems, thereby highlighting the need for ML approaches.

B. Graph Neural Network

Graph neural network (GNN) is specifically designed for tasks on graph-structured data, like node classification and graph

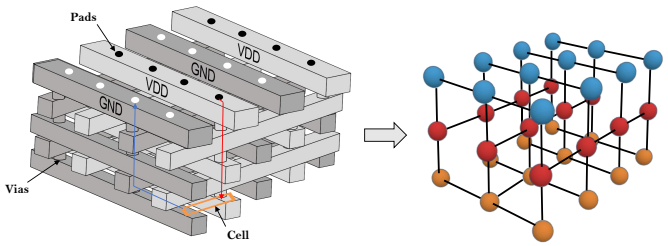


Fig. 1 The illustration of topological structure of PG.

regression, by modeling graph structures and aggregating node information via message-passing. Recently, considering the topology of the circuit, many graph-based approaches have emerged in the field of EDA. The framework in [12] leverages graph attention network (GAT) to model node and edge information in interconnect circuits jointly. GCN [13] effectively predicts chip temperature distribution by modeling thermal resistance networks under varying operating conditions. DeepGate [14] and PolarGate [15] apply GNNs to represent logic gates and Boolean networks, enhancing circuit analysis and design efficiency. EdgeGAT [16] and GNNTrans [17] improve the accuracy and speed of timing estimation from graph-based analysis using GNN. The adaptability of GNNs in modeling relationships among diverse design objectives enhances representation transferability, offering significant utility in EDA through improved task generalization.

C. Point Cloud

In this work, we employ point cloud methodologies to comprehensively model the complex topology in PGs and to improve the performance of full-chip IR drop prediction. Point cloud is a data format that represents 3D shapes or structures using a large number of spatial points, allowing for a better representation of complex three-dimensional layouts. Point cloud perception methods are divided into voxel-based and point-based. Voxel-based method discretizes continuous three-dimensional space into a finite number of voxels, making data processing and analysis simpler and more intuitive [18]. However, this method often resulted in the loss of multi-dimensional geometric details. In contrast, the point-based approach processes raw point cloud data directly, preserving its structural integrity and enabling end-to-end training without transformation requirements. Zou et al. [19] treat circuit layouts as point clouds, applying transformer-based techniques to enhance feature extraction, yielding strong results in congestion prediction and design rule verification.

D. Problem Formulation

This work aims to analyze the static IR drop at each node in PGs and get better performance. We represent each node in the PG as a vertex in a directed graph, facilitating the representation and prediction of IR drop at every node, including both cell-level and internal nodes. The edges of the graph are modeled to reflect the topological connections between nodes in the PG, whose direction corresponds to the direction of the current in the interconnecting wires. Therefore, the PG is treated as a directed graph, denoted as $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ represents the set of n vertices

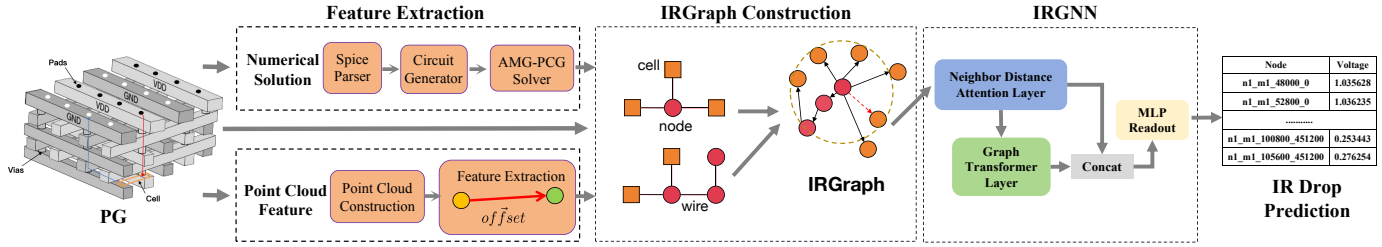


Fig. 2 Illustration of IRGNN framework for static IR drop prediction.

corresponding to both internal nodes and cells in the PG, and $E \subset V \times V$ represents the set of directed edges corresponding to the current-carrying wires. Our object is to design an algorithm F^* to intake the PG-based graph G to give the closest node-level IR drop prediction F, formulated as:

$$F^* = \arg\min_{F} \text{Loss}\left(F\left(G = (V, E)\right), y\right). \tag{1}$$

III. IRGNN Framework

A. Overall Flow

We focus on developing our graph-based framework, IRGNN, specializing in node-level static IR drop prediction, as shown in Fig. 2. The numerical solution provides a strong basis for understanding complex PG systems, while its integration with efficiency-promising GNN contributes to the development of more accurate and efficient analysis. In our framework, an efficient AMG-PCG solver is applied to obtain rough numerical solutions quickly and reliably. Besides, to better capture the extremely high geometric position characteristics of PG nodes, we model PG nodes as point cloud nodes with 3D coordinates and features. Then, we extract point cloud features to represent the relative positional relationships and current flow directions between nodes. Subsequently, to better utilize the topological information of PG, we propose a novel graph structure, IRGraph, with feature extraction that comprehensively represents the fine-grained PG structure. Finally, we introduce a new network, IRGNN, incorporating an NDA layer that aggregates information from nodes and edges using a distance attention mechanism, and a GT layer to capture both local neighbor and global information. More details of several key designs of our framework are elaborated in the following sub-sections.

B. Numerical Solution using AMG-PCG

To address the system matrix presented in Section II-A, our goal is to provide as accurate an initial solution as possible in a shorter runtime, laying the foundation for understanding the PG system and subsequent ML phase. The numerical solution, when utilized as a feature, enhances the interpretability of the model and increases the reliability of ML methods, which are often considered black-box approaches. In the numerical solution phase, the PG spice file is loaded with preprocessing involving a SPICE parser and a circuit generator. To achieve rapid and reliable numerical solutions, the algebraic multigrid preconditioned conjugate gradient (AMG-PCG) method proposed in PowerRush [6] is applied.

In AMG-PCG, the conjugate gradient (CG) method is enhanced by using an algebraic multigrid (AMG) preconditioner to improve convergence efficiency. The process begins with the standard CG iteration on the system $\boldsymbol{GV} = \boldsymbol{I}$, where AMG acts as a preconditioner to minimize the high-frequency error components, thus accelerating the solution process. Specifically, AMG constructs a hierarchy of coarser grids with restriction $\boldsymbol{R}^{(l)}$ and prolongation $\boldsymbol{P}^{(l)}$ operators for each level l. The preconditioned system becomes $\boldsymbol{M^{-1}GV} = \boldsymbol{M^{-1}b}$, where $\boldsymbol{M^{-1}}$ represents the AMG preconditioner applied to $\boldsymbol{G}$. By iterating through CG with AMG, the method progressively updates $\boldsymbol{V}$ while minimizing the residual $\boldsymbol{r} = \boldsymbol{I} - \boldsymbol{GV}$ until convergence, getting an accurate final solution efficiently.

Typically, numerical methods require a significant number of iterations to achieve an accurate solution. In our approach, we adopt a reduced iteration strategy to quickly obtain an approximate solution, which serves as a foundational feature for nodes in Graph-based models, as detailed in Section III-D. Despite this solution is not sufficiently accurate, it effectively provides rough IR drop values at each node, thereby greatly benefiting ML in understanding and learning PG systems.

C. Point Cloud Feature

The topology of the PG is just as crucial as the widely recognized and used circuit features, as it governs the path for power transmission directly. However, effectively utilizing the PG topology requires solving challenges related to the relative position of nodes in the PG metal layer and the relative position between nodes. At present, point cloud has proven to be highly effective for 3D object detection. Thus, we treat PG nodes as point cloud nodes, with each node having two attributes: 3D coordinates and features. The 3D coordinates of the nodes in the PG are determined based on the position of the metal layers and the metal rails to which the nodes belong. Specifically, the vertical position z of each node is derived from its corresponding metal layer, while the plane coordinates (x, y) are based on the node's location within the respective metal rail. Since the vertical distance between layers is not proportional to the overall size of PG, we standardize z by normalizing it to the average distance between the rails in the bottom-most layer of the PG. This normalization ensures that the plane coordinates (x, y) remain consistent with the original distribution in the PG, preserving the spatial relationships in the plane while adjusting for the vertical scaling.

Both the wire connections between nodes and the current flow in PG can indirectly affect the IR drop loss, which is critical for IR drop analysis. PointNet++ [20] calculates the distance and positional characteristics between each point and the central point used for feature extraction and weighting

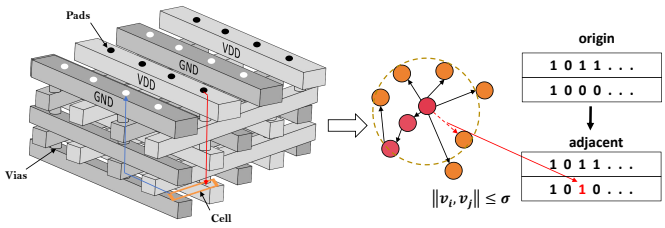


Fig. 3 IRGraph construction with distance-aware edge.

during convolution operations. Inspired by PointNet++ [20], we introduce an offset vector to describe the positional relationship between nodes and the direction of current flows depending on the nodes' net. For example, the offset between node $v_1 = (x_1, y_1, z_1)$ on the GND net and $v_2 = (x_2, y_2, z_2)$ on the VDD net can be formulated as $offset = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$. With the offset vector, it is possible to describe both the relative positions between nodes in PG and the directivity of the current flow.

D. IRGraph Construction

Existing methods predominantly rely on 2D feature maps derived from current patterns, such as total power or frequency, for IR drop prediction. However, these approaches are limited in that they typically focus on single-layer, grid-based representations, which fail to capture the node-level details of the PG. Moreover, these 2D maps are unable to represent the complex, multidimensional interactions between cells and wires, such as the flow of current and the physical topology of the PG. To overcome these limitations, we model the PG as a graph and propose a novel graph structure, IRGraph, incorporating both the PG topology and current flow information. We enhance the representation of the PG characteristics by extracting additional PG-based features, combining the numerical solution and point cloud data. The IRGraph is constructed depending on the original cell-PG connection and especially virtual edges for local consistency. IRGraph can not only represent the IR drop of the lowest cell granularity but also predict the IR drop of the nodes inside PG, which is impossible to achieve with the previous CNN-based method.

1) Graph Construction

An unweighted directed graph $G = (V, E)$ is constructed to model the practical non-Euclidean topology of the PG network, as shown in Fig. 3. Here, V represents all nodes within the PG network, encompassing both circuit cells and internal nodes. The formation of edges is performed in two stages. First, the original PG connections are treated as edges in the IRGraph to capture voltage propagation through actual current flows. Then, a distance-aware edge construction is applied, incorporating edges based on spatial proximity (Euclidean distance $\|v_i, v_j\|_2 \leq \delta$ for nodes v_i, v_j). These virtual edges facilitate information propagation within the GNN. The extracted features from this process are assigned as features for both V's and E's, detailed in the next part.

2) Feature Extraction

To enhance the accuracy and convergence, we extract more node-level features to improve the representation of nodes.

- **The current** for each node is calculated by the current source and Kirchhoff's current law and voltage law together with the resistance of the wire.
- **The effective distance**, defined as the reciprocal of the sum of the Euclidean distances to all voltage sources, quantifies the node's proximity to these sources.
- **The shortest path resistance** computes the total resistance using the shortest path from node to power sources.
- **The net value** indicates that the node belongs to the VDD or GND network, represented by the binary value 0 or 1.
- **The numerical solution** detailed in Section III-B.
- **The point cloud position** is the 3D coordinate of nodes.

Metal wires in PG also play a crucial role in power delivery since they carry the current flow. To represent the effect of wires in the topologies of PG and current flow, we extract two wire-based features for each edge in IRGraph:

- **The wire resisitance**, extracted from spice file.
- **The wire offset**, calculated as Section III-C.

E. IRGNN Model

In our work, we design a novel graph neural network, IRGNN, specializing in static IR drop prediction, which is a node regression task. The IRGraph is utilized as the input of IRGNN, containing features both on nodes and edges. In IRGNN, to take advantage of both node and edge features, we design an NDA layer, as the node attention aggregator in NDA processes the neighbor node and edge representations simultaneously with attention weight. According to the principle of local consistency, we incorporate the attention weight in this layer based on the neighbor distance between nodes. Then, the global representation of PG is captured by the GT layer, allowing the contact of each node. Finally, the IRGNN representation is exported through an MLP readout.

1) Neighbor Distance Attention Layer

Traditional GAT is designed primarily for node classification tasks and has achieved significant success. We design the NDA layer, displayed in Fig. 4, to adapt it to IR drop prediction. This layer accepts the transformed node and edge representations generated through the linear transformer as inputs, $\{\boldsymbol{v}_i^l, \forall i \in N_v\}$ and $\{\boldsymbol{e}_{ij}^l, \forall (i,j) \in N_e\}$, and produces attention based aggregated node representations $\{\boldsymbol{v}_i^{l+1}, \forall i \in N_v\}$, where l is number of layers of network.

To utilize features from nodes and edges, we execute a convolution operation to aggregate information from a node's neighboring nodes and connected edge data in the update process of the node. In this way, the information provided by the topology of the PG system and current load patterns is unified, enabling a cohesive representation that leverages both spatial and geometric insights. The node attention coefficient $\boldsymbol{h}_{ij}^l$ indicates the importance of neighborhood information to the target node and is computed as:

$$\boldsymbol{h}_{ij}^l = \left(\boldsymbol{a}^l\right)^\top \left[\boldsymbol{v}_i^l \| \boldsymbol{e}_{ij}^l \| \boldsymbol{v}_j^l\right] \boldsymbol{d}_{ij}, \tag{2}$$

where i denotes the target node, and j is its neighboring node within the neighborhood set N_i. The attention mechanism, represented by the shared vector $\boldsymbol{a}$, allows for consistent attention across all nodes. To capture edge directionality, we

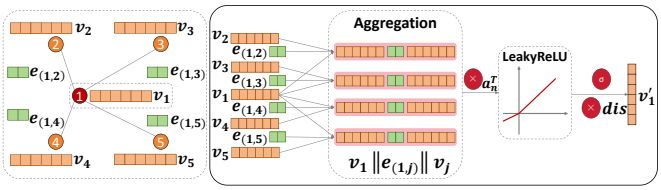


Fig. 4 The architecture of our designed NDA layer.

introduce $\boldsymbol{d}_{ij}$, where a value of 1 indicates the direction from node i to j, and -1 indicates the opposite direction.

The cell connection edges cause complex interactive relationships owning to discriminative node distances. To adaptively rescale and aggregate more significant information from complex edge connections of different-distance regions, the distance attention weight is designed as:

$$\boldsymbol{dis}_{ij} = \left(\frac{\|\boldsymbol{v}_i, \boldsymbol{v}_j\|_2}{h} \right)^{-\beta}, \tag{3}$$

where h denotes the distance threshold, and β controls the decay rate. $\|\boldsymbol{v}_i, \boldsymbol{v}_j\|_2$ is their Euclidean distance.

To make coefficients easily comparable across different nodes, all node attention coefficients of node i are normalized using the softmax function with distance attention weight, like:

$$\alpha_{ij}^l = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{h}_{ij}^l\right)\right)}{\sum_{k \in N_i} \exp\left(\text{LeakyReLU}\left(\boldsymbol{h}_{ik}^l\right)\right)} \boldsymbol{dis}_{ij}. \tag{4}$$

With the activate function σ, the node representation is updated with neighbor distance and other features' aggregation:

$$\boldsymbol{v}_i^{l+1} = \sigma\left(\sum_{j \in N_i} \alpha_{ij}^l \boldsymbol{v}_j^l \right), \quad \forall i \in N_v. \tag{5}$$

2) Graph Transformer Layer

Different from the CNN-based method, which focuses on exploring the local information of IR drop, the GNN-based method uses the way of message passing, which leads to too much local information and ignores the global information easily. In IRGNN, we leverage the GT layer [21] to capture global information across the graph structure, which is essential for improving the performance in IR drop prediction. The GT layer allows each node to attend to all other nodes in the graph, enabling global interactions without the limitation of local neighborhoods. Through a self-attention mechanism, each node's representation is updated by aggregating features from all nodes, weighted by learned relevance scores, which embeds global context into individual node features. This global attention mechanism effectively captures complex dependencies between distant nodes, leading to a more holistic node representation. Additionally, the GT layer incorporates position encodings to enhance focus on critical regions, injecting more structural information into the model.

IV. Evaluation

A. Experimental Settings

Baselines. IRGNN is evaluated against competitive ML-based IR drop prediction models, including IREDGe [8] and MAUnet [11]. Furthermore, it is also compared to the winner of the ICCAD 2023 contest [22].

Datasets. The ICCAD2023 [22] dataset focuses on static IR drop prediction, featuring 20 real designs and 100 synthetic designs generated based on [23], closely resembling realistic PGs. To increase dataset diversity and assess model generalization, three open-source benchmarks [23] are utilized: Nangate, ASAP, and Skywater, comprising 1000, 1000, and 418 designs, respectively. Experiments with baselines are conducted on ICCAD2023 and Nangate to validate the framework's effectiveness. The trainset and testset are evenly divided according to the number of nodes in PGs, to verify model's generality and improve rationality of experiments. Additionally, large-scale datasets are employed to assess performance scalability. Three benchmarks are used during pretraining, and fine-tuning is performed on fake and real designs from the ICCAD2023.

Metrics. Following the contest [22], the mean absolute error (MAE), F1 score, Pearson correlation coefficient (CC), and runtime are selected as the evaluation metrics. The MAE is the average absolute difference between a prediction and the ground truth. The CC measures the linear correlation between predicted values and ground truth values. The F1 score is a binary classification metric utilized to assess the prediction performance of hotspot regions. Given that designers are particularly concerned with the worst-case IR drop, minimizing modeling maximum error is of utmost importance. The Maximum IR Drop Error (MIRDE) metric is introduced.

B. Main Results

To compare with the ML-based baselines, our experiments first compare the result on the bottom layer, which mainly includes the nodes of the working cells and is most concerned. Considering former CNN-based methods can only predict roughly at the pixel level, our graph-based method can analyze the IR drop of each node in PG. Therefore, the golden result is obtained for all nodes, and we make a comparison with a famous numerical simulator PowerRush [6], including the analysis error and efficiency of all nodes in PGs.

Comparison with ML-based Method. Like the method of dealing with small trainsets problem by pretraining with public training sets which have higher uncertainty, IRGNN is trained first with fake designs, which may not be clean in some maps to get general representations of IR drop. Then, half of the real designs are used to fine-tune. TABLE I shows the IR drop analysis results of different methods. Considering the newly proposed competitive MAUnet [11], our approach achieves better performance on each dataset with no significant time cost increase. IRGNN still outperforms all baselines in MIRDE, representing more accuracy in the worst-case region. Our model is trained from the foundation of numerical solutions, and the topology information utilized by NDA increases the interpretability and robustness of the model.

Considering the performance potential of large-scale datasets, another experiment is conducted. In the pretraining phase, three open-source benchmarks aforementioned are utilized, while the fake designs and real designs in the same setting of the last ICCAD2023 experiment are fine-tuned relatively step by step. The results are summarized in TABLE II.

TABLE I Comparison with ML-based Methods. The Unit of MAE and MIRDE is $\times 10^{-4}V$.

Methods	ICCAD2023 dataset				Nangate dataset			
	MAE↓	F1↑	CC↑	MIRDE↓	MAE↓	F1↑	CC↑	MIRDE↓
IREDGe [8]	3.54	0.48	0.86	6.42	0.83	0.67	0.89	2.26
MAUnet [11]	1.12	0.60	0.95	4.35	0.43	0.77	0.99	1.17
Contest Winner [22]	1.15	0.58	0.94	4.37	0.62	0.75	0.96	1.15
IRGNN (Ours)	**0.83**	**0.72**	**0.97**	**2.89**	**0.26**	**0.80**	**0.99**	**0.96**

TABLE II Results of Large-scale Dataset.

Methods	MAE↓	F1↑	CC↑	MIRDE↓
IREDGe [8]	2.79	0.53	0.89	5.58
MAUnet [11]	0.75	0.71	0.97	3.05
Contest Winner [22]	0.89	0.67	0.97	3.44
IRGNN (Ours)	**0.46**	**0.77**	**0.98**	**2.37**

TABLE III Results of Evaluation on every Node in PG.

Methods	MAE↓	F1↑	CC↑	MIRDE↓	Runtime↓
PowerRush [6]	0.71	0.58	0.94	2.89	30.52
IRGNN (Ours)	**0.56**	**0.75**	**0.96**	**2.83**	**6.64**

IRGNN achieves better performance with the improvement of 38.67% on MAE, 8.45% on F1, 1.03% on CC, and 22.29% on MIRDE with no significant time cost increase.

Comparison with Numerical Method. The CNN-based method has the limitation of using pixel as coarse-grained prediction. However, IRGNN breaks this limitation with every node's prediction. To evaluate the IRGNN's ability on node-level prediction, the comparison with PowerRush [6] is conducted and results are shown in TABLE III. Notably, IRGNN surpasses PowerRush in all metrics, with significantly less time cost, indicating great performance on nodes of both the bottom layer and inner layers. From the visual comparison in Fig. 5, it is clear that IRGNN can achieve better MAE at a speed that cannot be matched by PowerRush. Thanks to the fusion of numerical and GNN methods, IRGNN achieves a better trade-off between accuracy and efficiency.

C. Transfer Learning

The transferability is of great significance in IR drop analysis since the amount of dataset is limited and the style of design is diverse. In order to evaluate the transferability and generalizability of our proposed method model, the experiment uses the Nangate as a training set and regards the real designs from ICCAD2023 as test samples. From the results in TABLE IV, our method is much better in generalization ability with better prediction on the corresponding IR drop in the face of very different and never-seen PGs. The numerical solution provides a reliable foundation of IR drop distribution and the ability to deal with topological neighbor information of NDA increases the interpretability of the model's decisions, allowing it to perform robustly under varying conditions.

D. Ablation Study

Ablation experiments are conducted to evaluate the impact of various techniques in our IRGNN. Fig. 6 shows the improved results with a certain technique. The results demonstrate that the numerical solution (Num. Solu.) significantly reduces

TABLE IV Transfer Results on ICCAD2023 Dataset.

Method	MAE↓	F1↑	CC↑	MIRDE↓
IREDGe [8]	4.88	0.39	0.71	8.01
MAUnet [11]	2.07	0.55	0.84	4.78
Contest Winner [22]	1.95	0.55	0.83	4.92
IRGNN (ours)	**1.68**	**0.62**	**0.88**	**3.61**

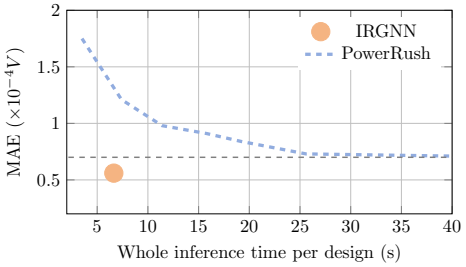


Fig. 5 Comparison of IRGNN and PowerRush [6].

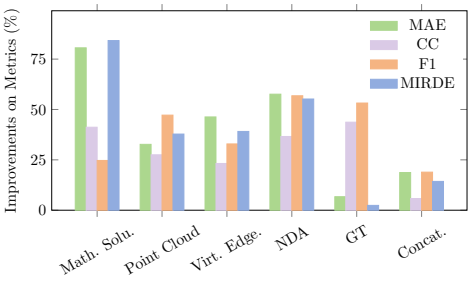


Fig. 6 Results of Ablation Study.

MAE and MIRDE, likely due to its precise initial point for learning. Additionally, our point cloud features also improve performance with better F1. Both the NDA and GT layers also contribute to performance gains, especially in the CC and F1.

V. Conclusion

We propose a novel graph-based framework, IRGNN, combining the numerical solution with ML and utilizing the topological information of PG. IRGNN exploits the advantages of both numerical solution and ML methods, and can strike a good trade-off between efficiency and accuracy. Experiments demonstrate that our framework can achieve the best performance compared to newly proposed methods.

Acknowledgment

This work is suppported by the National Key Research and Development Program of China (No. 2023YFF0725103, 2022YFB2901100), the National Natural Science Foundation of China (No. U22B2038, 62192784, 62404021), Young Elite Scientists Sponsorship Program (No. 2023QNRC001) by CAST, and the Beijing Natural Science Foundation (No. 4244107, QY24216, QY24204).

REFERENCES

[1] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann, 2009.

[2] X. Zhan, P. Li, and E. Sánchez-Sinencio, "Distributed On-Chip Regulation: Theoretical Stability Foundation, Over-Design Reduction and Performance Optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[3] K. F. Yong, C. T. Lim, and W. K. Teng, "System Level IR Drop Impact on Chip Power Performance Signoff for Risc-V System on Chip," in *IEEE/ACM International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*, 2022, pp. 1–4.

[4] T.-H. Chen and C. C.-P. Chen, "Efficient Large-scale Power Grid Analysis Based on Preconditioned Krylov-Subspace Iterative Methods," in *ACM/IEEE Design Automation Conference (DAC)*, 2001, pp. 559–562.

[5] C.-H. Chou, N.-Y. Tsai, H. Yu, C.-R. Lee, Y. Shi, and S.-C. Chang, "On the Preconditioner of Conjugate Gradient Method—a Power Grid Simulation Perspective," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 494–497.

[6] J. Yang, Z. Li, Y. Cai, and Q. Zhou, "Powerrush: an Efficient Simulator for Static Power Grid Analysis," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 22, no. 10, pp. 2103–2116, 2013.

[7] Z. Liu and W. Yu, "Powerrchol: Efficient Power Grid Analysis Based on Fast Randomized Cholesky Factorization," in *ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[8] V. A. Chhabria, V. Ahuja, A. Prabhu, N. Patil, P. Jain, and S. S. Sapatnekar, "Thermal and IR Drop Analysis Using Convolutional Encoder-Decoder Networks," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2021, pp. 690–696.

[9] V. A. Chhabria, Y. Zhang, H. Ren, B. Keller, B. Khailany, and S. S. Sapatnekar, "MAVIREC: ML-Aided Vectored IR-Drop Estimation and Classification," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2021, pp. 1825–1828.

[10] F. Guo, J. Liu, J. Zhai, J. Jia, K. Zhao, and C. Shi, "PGAU: Static IR Drop Analysis for Power Grid Using Attention U-Net Architecture and Label Distribution Smoothing," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2024, pp. 452–458.

[11] M. Wang, Y. Cheng, Y. Lin, K. Peng, S. Yang, Z. Jin, and W. Xing, "MAUnet: Multiscale Attention U-Net for Effective IR Drop Prediction," in *ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[12] Y. Zuo, Y. Ye, H. Zhang, T. Chen, H. Yan, and L. Shi, "A Graph-Learning-Driven Prediction Method for Combined Electromigration and Thermomigration Stress on Multi-Segment Interconnects," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2024, pp. 1–6.

[13] L. Chen, W. Jin, and S. X.-D. Tan, "Fast Thermal Analysis for Chiplet Design based on Graph Convolution Networks," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2022, pp. 485–492.

[14] M. Li, S. Khan, Z. Shi, N. Wang, Y. Huang, and Q. Xu, "Deepgate: Learning Neural Representations of Logic Gates," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, p. 667–672.

[15] J. Liu, J. Zhai, M. Zhao, Z. Lin, B. Yu, and C. Shi, "PolarGate: Breaking the Functionality Representation Bottleneck of And-Inverter Graph Neural Network," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[16] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Graph-Learning-Driven Path-Based Timing Analysis Results Predictor from Graph-Based Timing Analysis," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2023, pp. 547–552.

[17] ——, "Fast and Accurate Wire Timing Estimation Based on Graph Learning," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2023, pp. 1–6.

[18] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2017.

[19] J. Zou, X. Wang, J. Guo, W. Liu, Q. Zhang, and C. Huang, "Circuit as Set of Points," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 2023.

[20] G. Qian, Y. Li, H. Peng, J. Mai, H. Hammoud, M. Elhoseiny, and B. Ghanem, "Pointnext: Revisiting pointnet++ with improved training and scaling strategies," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 2022, pp. 23 192–23 204.

[21] V. P. Dwivedi and X. Bresson, "A Generalization of Transformer Networks to Graphs," in *AAAI Conference on Artificial Intelligence*, 2021.

[22] (2023) CAD Contest at ICCAD. [Online]. Available: https://www.iccad-contest.org/2023/Winners.html

[23] V. A. Chhabria, K. Kunal, M. Zabihi, and S. S. Sapatnekar, "BeGAN: Power Grid Benchmark Generation Using a Process-Portable GAN-Based Methodology," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8.

iG-kway: Incremental k-way Graph Partitioning on GPU

Wan Luan Lee[1], Shui Jiang[2], Dian-Lun Lin[1], Che Chang[1], Boyang Zhang[1],
Yi-Hua Chung[1], Ulf Schlichtmann[3], Tsung-Yi Ho[2], Tsung-Wei Huang[1]
[1]*University of Wisconsin–Madison, USA*
[2]*The Chinese University of Hong Kong, Hong Kong*
[3]*Technical University of Munich, Germany*
{wlee329, dianlun.lin, cchang289, bzhang523, yihua.chung, tsung-wei.huang}@wisc.edu,
{sjiang22, tyho}@cse.cuhk.edu.hk, ulf.schlichtmann@tum.de

***Abstract*—Recent advances in GPU-accelerated graph partitioning have achieved significant performance gains but remain limited to full graph partitioning, lacking support for incremental updates. This limitation is critical in CAD applications, where circuit graphs undergo iterative, incremental modifications during optimization. We present *iG-kway*, the first GPU-based incremental k-way graph partitioner. iG-kway features an incrementality-aware data structure and a refinement kernel that efficiently updates only affected vertices with minimal quality loss. Experiments show that iG-kway delivers up to 84× speedup over the state-of-the-art G-kway with comparable partitioning quality.**

I. Introduction

Graph partitioning is important for the design of efficient computer-aided design (CAD) algorithms because it allows an algorithm to break down a circuit graph into smaller and manageable pieces. However, as circuit graphs continue to grow in size, graph partitioning becomes increasingly time-consuming. For example, the sequential graph partitioner, metis [1], can take several minutes to partition just one million-node graph [2], and the runtime keeps increasing as the graph size becomes larger. To reduce the runtime, researchers have proposed various parallel graph partitioning algorithms [3]–[12]. For example, mt-metis [4] parallelized multi-level graph partitioning using multi-core CPUs, but the speedup is limited to only 8–16 CPU threads [13]. More recently, G-kway [13], Jet [2], and GKSG [14], [15] explored data parallelism from different stages of graph partitioning and offloaded time-consuming tasks (e.g., coarsening, refinement) to GPU, achieving significant speedup for large graphs.

In addition to partitioning an input graph once, which we refer to as *full graph partitioning* (FGP), *incremental graph partitioning* (IGP) is a critical enabler for many CAD applications that incorporate graph partitioning in a loop. For instance, the multi-input RTL simulator [16], [17] counts on iterative IGP to discover an optimal task graph for heterogeneous scheduling [18], [19]; similarly, a timing-driven optimizer also relies on iterative IGP to enhance the runtime performance of a timing analyzer [20]. In these applications, when the graph is incrementally modified, the partitioner must quickly refine the partitioning result to maintain a reasonable turnaround time across thousands or even millions of incremental iterations. As shown in Figure 1, IGP can save a significant amount of time compared to FGP. Without IGP, we cannot fully unlock the benefit of graph partitioning.

IGP has been studied in prior works [21]–[25], with a core focus on refining small subgraph regions affected by graph modifications, rather than performing a full re-partitioning from scratch. However, these works are largely limited to CPU architectures and can become inefficient when handling large graphs or when affected regions are large. Inspired by the recent success of GPU-accelerated FGP [13], we believe that GPU can also enhance the performance of IGP due to the large amount of data parallelism exhibited by graph partitioning. More importantly, as more CAD applications begin leveraging GPU acceleration [16], [17], [26]–[40], there is an increasing need to re-target time-consuming CPU tasks to GPU. For instance, a GPU-accelerated IGP will further speed up the incremental task graph optimization process of [16] (which takes several hours) while reducing the cost of moving and converting graph data between CPU and GPU during iterative IGP.

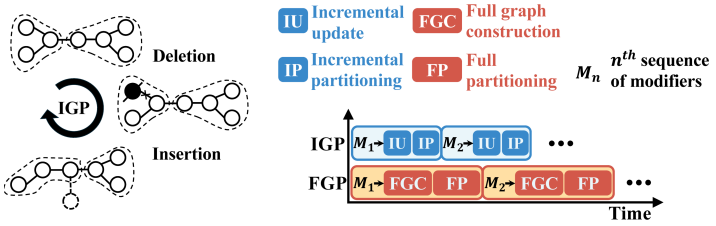


Fig. 1. Incremental graph partitioning (IGP) and its runtime advantage over full graph partitioning (FGP).

However, designing a GPU-parallel incremental graph partitioner is very challenging. First, existing GPU partitioners [2], [13], [14] count on *static 1D arrays*, such as the compressed sparse row (CSR), to store graphs on GPU. These static data structures make it very challenging to update graphs without rebuilding the entire arrays. Second, graph modifiers can affect vertices in different subgraph regions of different sizes. This varying number of affected vertices needs a clever strategy to balance workloads across GPU threads during incremental partitioning. Third, graph modifiers can potentially disrupt the balance of existing partitions. We need an effective GPU kernel algorithm to quickly restore partition balance while maintaining a satisfactory cut size.

While parallel IGP has been previously studied [24], [25], these efforts focused on CPU architectures. Due to the distinct performance models and memory hierarchies between CPU and GPU, we cannot directly apply these methods to GPU. For example, IOGP [24] introduced an online graph partitioning algorithm for distributed graph databases. However, IOGP focuses on optimizing data locality to minimize communication overhead in a distributed computing environment, which differs from our application focus. On the other hand, [25] formulated IGP into a two-layer linear programming (LP) proxy problem and solved it using multiple CPU threads. However, their methods require iteratively re-formulating the proxy LP problem, which is inherently sequential and cannot scale to large IGP problems.

To overcome these challenges, we introduce iG-kway, a GPU-parallel k-way graph partitioner that efficiently supports incrementality. To the best of our knowledge, this work is one of the earliest research on GPU-parallel IGP, aiming to unlock the full potential of GPU-accelerated graph partitioning. We summarize our key contributions as follows:

- We introduce a GPU-aware bucket-list graph representation that can efficiently handle graph modifiers without requiring any data structure rebuilds.
- We aggregate the affected vertices in a centralized buffer and dynamically assign GPU threads to process them, ensuring balanced workloads across the GPU threads.

- We design a GPU-parallel refinement kernel algorithm that efficiently re-balances partitions by moving affected vertices to a pseudo-partition and performing incremental refinement.

We have evaluated the performance of iG-kway on industrial circuit graphs and compared our results with the state-of-the-art GPU-accelerated graph partitioner, G-kway [13]. Experimental results show that iG-kway achieves an average speedup of 84× over G-kway, with comparable cut sizes.

II. Problem Definition and Notation

Given an undirected graph, $G = (V, E)$, where V is a set of vertices, and E is a set of edges. Each element in E is of the form $e = (u, v)$ which represents the connection between u and v in V. For a vertex $v \in V$, we denote the weight of v by W_v, while for an edge $e \in E$, we denote the weight of e by W_e. For a vertex $v \in V$, its adjacent vertex set is denoted as $adj(v)$. Given k, if $P = \{p_1, p_2, \ldots, p_k\}$ is a disjoint partition of V, we call P a k-way partition. For $v \in V$, we define $P(v) = i$ if $v \in p_i$, and its external neighbors as $adj_{ext}(v) = \{u \in adj(v) \mid P(u) \neq P(v)\}$, and its internal neighbor as $adj_{int}(v) = \{u \in adj(v) \mid P(u) = P(v)\}$. We define the cut size as $\sum_{e=(u,v)\in E, P(u)\neq P(v)} W_e$. Cut size is widely used for evaluating the quality of a partition since it represents the interconnect complexity among partitions. The partition weight of p_i is defined as $W_{p_i} = \sum_{v\in p_i} W_v$.

The goal of FGP is to find a k-way partition from scratch that satisfies the balance constraint while minimizing the cut size. The balance constraint limits the maximum weight of p_i as

$$W_{p_i} \leq W_{p_{\max}} = (1+\epsilon)\frac{\sum_{v\in V} W_v}{k}, \quad 0 < \epsilon \ll 1$$

where $W_{p_{max}}$ is the maximum allowable partition weight and ϵ is the imbalance ratio given by applications. Given a partitioned graph, the first goal of IGP is to apply a sequence of *graph modifiers* to the graph. Each modifier corresponds to a vertex insertion (M_u^+), a vertex deletion (M_u^-), an edge insertion ($M_{(u,v)}^+$), or an edge deletion ($M_{(u,v)}^-$). In most IGP applications [16], [20], [41], the number of modifiers is smaller than the graph size. The next goal of IGP is to efficiently refine the modified graph without starting from scratch while minimizing the cut size.

III. Overview of iG-kway

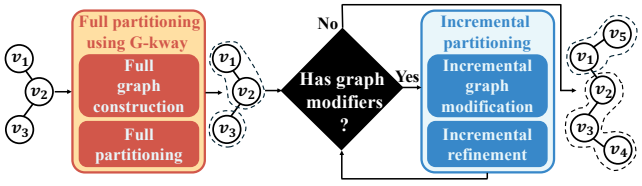


Fig. 2. Overview of the proposed incremental graph partitioner, iG-kway.

Figure 2 shows the overview of our GPU-parallel incremental k-way graph partitioner, iG-kway, which consists of two main stages: *full partitioning* (Section IV) and *incremental partitioning* (Section V). The goal of the full partitioning is to derive a high-quality partition from the original graph, which provides a foundation for the incremental partitioner to optimize subsequently modified graphs. We utilize G-kway [13] with a new constrained coarsening strategy to achieve a high-quality partitioning result. On the other hand, the goal of incremental partitioning is to update and refine the modified graph without starting from scratch. Our incremental graph partitioning has two main stages, *incremental graph modification* and *incremental refinement*, where (1) the former introduces a bucket-list data structure that stores the input graph and supports direct modification on GPU and (2) the latter introduces a GPU kernel algorithm that balances and refines the partition after the graph is incrementally modified.

IV. Full Partitioning with Constrained Coarsening

We use the state-of-the-art GPU-accelerated multilevel graph partitioner, G-kway [13] to perform full partitioning due to its high partitioning quality and fast runtime. G-kway employs a multilevel approach, iteratively coarsening the graph into a smaller representation until it reaches a manageable size, at which point the partitioning stage begins. To accelerate the coarsening process, G-kway employs a union-find-based coarsening that merges many vertices simultaneously to reduce the graph size per iteration. Here, G-kway introduces a parallel union-find algorithm to iteratively group vertices into subsets. Despite parallelism, this approach may result in imbalances in coarsened vertex weights, as each subset contains a varying number of vertices. This imbalance makes it challenging for the subsequent partitioning stage to achieve a balanced partition. Figure 3 (a) shows an example of G-kway's union-find-based coarsening, where G-kway groups vertices into two imbalanced subsets.

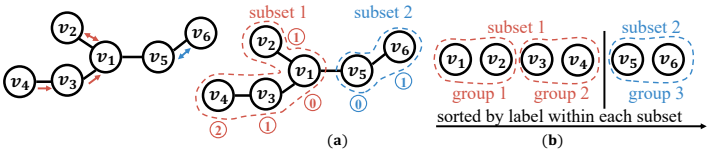


Fig. 3. Examples of two coarsening methods, including (a) G-kway's union-find based coarsening and (b) our constrained coarsening. Each vertex has an arrow pointing to its selected neighbor, and its label (n) or (n) indicates the iteration when it was grouped into the subset. Vertices circled in the red dashed line will be coarsened into a coarsen vertex.

To address this issue, we divide the vertices in each subset into small, fixed-size groups of size s and merge the vertices within each group into a single coarsened vertex in parallel. This strategy reduces the graph size in each iteration by merging multiple vertices simultaneously while maintaining balanced coarsened vertex weights. To this end, a straightforward solution is randomly selecting s vertices to form smaller groups. However, this approach may result in poor partitioning quality, as vertices that are far apart could end up in the same group and be merged into a single coarsened vertex, distorting the original graph structure. For instance, in Figure 3 (a), randomly dividing vertices in subset 1 can result in placing distant vertices, v_2 and v_4, in the same group.

To address this problem, we modify G-kway's union-find-based coarsening algorithm by labeling each vertex with the iteration in which it joins a subset. Since vertices that are farther apart are merged into the subset in later iterations, we can sort the vertices based on their labels and divide them into groups, ensuring vertices that are closer together are placed in the same group. Figure 3 (b) shows our *constrained coarsening* strategy, where vertices are first sorted by iteration number in ascending order within groups, and large subsets are divided into smaller groups of size two. In this approach, groups of similar size merge into a single coarsened vertex, producing more balanced coarsened vertex weights.

V. Incremental Partitioning

A. Bucket-list Graph Representation

Existing GPU graph partitioners [2], [13], [14] count on CSR to store graphs on GPU. In CSR, all vertices' neighbors are concatenated into an adjacency list of size $|E|$, with an adjacency pointer recording each vertex's neighbor position. However, this statically packed data structure makes modifying the graph very challenging without rebuilding the structure. For instance, inserting an edge in the CSR data structure requires shifting all elements in the adjacency list and updating their adjacency pointers. Furthermore, updating the CSR is typically done on the CPU, which introduces additional data movement and conversion overhead between the CPU and GPU.

To overcome this challenge, we propose a bucket-list data structure that supports efficient graph modifications directly on the GPU by storing

each vertex's neighbors in pre-allocated buckets. We design the buckets with 32 slots to align with the GPU warp size (32 threads), reducing thread divergence and enabling efficient intra-warp communication using CUDA warp-level primitives [42]. To avoid rebuilding the bucket-list data structure, we allocate extra buckets for each vertex to accommodate future edge insertions. The number of buckets for vertex u is determined by the following formula:

$$\left\lceil \frac{D(u)}{32} \right\rceil + \gamma$$

where $D(u)$ is the degree of u (i.e., the number of neighbors of u) and γ is the number of the extra buckets per vertex (by default, iG-kway sets γ to one). To accommodate bigger graph modifications, applications can set a higher γ. Figure 4 (a) shows an example of our bucket-list data structure, where each bucket contains four slots. All buckets are concatenated into a bucket-list, with a bucket pointer recording each vertex's bucket position within the list. To avoid reallocating memory when more buckets are required (e.g., due to vertex insertion), we pre-allocate a large block of memory for the bucket-list and use a pointer to track the current number of buckets.

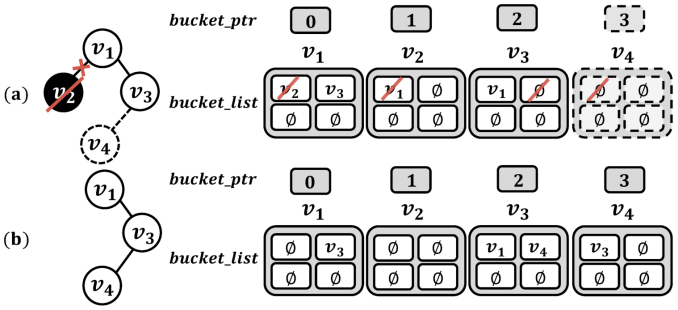


Fig. 4. An example of our bucket-list data structure, where each vertex has a bucket, and each bucket contains four slots. Empty slots are denoted by ∅. Figures (a) and (b) show the graph before and after applying the following graph modifiers: $M^-_{v_2}$, $M^+_{v_4}$, $M^-_{(v_2,v_1)}$, $M^-_{(v_1,v_2)}$, $M^+_{(v_4,v_3)}$, and $M^+_{(v_3,v_4)}$.

B. Incremental Graph Modification

1) Edge Modifiers: To efficiently handle $M^+_{(u,v)}$ (insert an edge (u, v)), we use a GPU warp to locate an empty slot in vertex u's buckets through fast intra-warp communication and insert vertex v to the first empty slot. Algorithm 1 presents our edge insertion algorithm. All threads in the warp first fetch the number of buckets allocated to u and the start position of its buckets from *bucket_ptr* (lines 4-5). Threads then process one bucket at a time until an empty slot is found or all buckets are checked (line 6). To efficiently locate an empty slot, each thread fetches a slot value in the bucket and uses the warp-level primitive [42], `__ballot_sync`, to simultaneously evaluate whether its slot is empty, storing the results in a bitmask (lines 7-8). If an empty slot is found, `__ffs` returns the first empty slot (line 9), and threads insert the edge into this slot and then terminate. (lines 10-12). If no slot is empty, each thread increments the bucket counter and continues looking for an empty slot in the next bucket (line 13).

We handle $M^-_{(u,v)}$ using the same strategy as Algorithm 1 except that instead of locating an empty slot, threads within the warp work simultaneously to find v in u's buckets and mark v as empty.

2) Vertex Modifiers: To delete a vertex u, a straightforward approach is to remove its buckets from the bucket-list. However, this approach can incur significant overhead, as it needs to recalculate the bucket pointer and rebuild the bucket-list. To address this problem, we use a vertex status array to track each vertex's current status (deleted or active) without removing its buckets from the bucket-list. Algorithm 2 presents our approach for handling the vertex modifier (M^+_u and M^-_u) using a GPU warp. To handle M^-_u, threads within the same warp first mark u as deleted in the vertex status array (line 5) and cooperatively remove all of u's neighbors by marking all slots in its buckets as empty (lines 11-13). Similarly, to handle M^+_u, threads within the same warp first mark u as active in the vertex status array (line 8). Then, they assign u a single bucket and add the bucket to the end of the bucket-list by updating the bucket pointers accordingly (lines 9-10). Finally, threads initialize all slots in u's buckets as empty (lines 11-13).

Algorithm 1: Edge Insertion

```
   Input: bucket_list, bucket_ptr, M+_(u,v), thread index in a GPU warp
          lane_id
1  parallel for each thread in a GPU warp
2      slot ← -1
3      bucket_cnt ← 0
4      bucket_start ← bucket_ptr[u]
5      num_bucket ← bucket_ptr[u + 1] - bucket_start
6      while slot == -1 && bucket_cnt < num_bucket
7          nbr ← bucket_list[bucket_start + bucket_cnt × 32 + lane_id]
8          if_empty ← __ballot_sync(FULL, nbr == ∅)
9          slot ← __ffs(if_empty) - 1
           // empty slot found
10         if slot ≠ -1
11             bucket_list[bucket_start + bucket_cnt × 32 + slot] ← v
12             return
13         bucket_cnt++
```

Algorithm 2: Vertex Insertion / Deletion

```
   Input: bucket_list, bucket_ptr, vertex_status, M+_u or M-_u, lane_id
1  parallel for each thread in a GPU warp
2      bucket_cnt ← 0
3      bucket_start ← bucket_ptr[u]
4      if M-_u
5          vertex_status[u] ← deleted
6          num_bucket ← bucket_ptr[u + 1] - bucket_start
7      else if M+_u
8          vertex_status[u] ← active
9          num_bucket ← 1
10         bucket_ptr[u + 1] ← bucket_start + num_bucket
11 while bucket_cnt < num_bucket
12     bucket_list[bucket_start + bucket_cnt × 32 + lane_id] ← ∅
13     bucket_cnt++
```

C. Incremental Refinement

Once the graph is incrementally modified, the existing partitioning result may become invalid due to the change in graph structure and balance condition. For example, adding too many vertices can cause the partition to violate the balance constraint, while inserting edges may require relocating vertices to reduce the cut size. To refine a modified graph without starting from scratch, one possible approach is to use the independent-set-based refinement algorithm in G-kway [13]. However, G-kway's refinement algorithm does not account for potential imbalances that may occur after applying graph modifiers. Additionally, because it lacks IGP support, G-kway refines all vertices on the partition boundary (i.e., vertices with $adj_{ext} \neq 0$), which is unnecessary when only local subgraph regions are affected. To address this problem, we propose an efficient incremental refinement algorithm that restores partition balance and refines only the vertices affected by graph modifiers. Our algorithm consists of two steps, *partition balancing* and *parallel refinement*, explained below:

Algorithm 3: Partition Balancing

```
   Input: bucket_ptr, bucket_list, partition, vertex_in_pseudo,
          vertex_in_pseudo_size
   Input: affected_vertex initialize to false
   // assign a graph modifier to a GPU warp
1  parallel for each thread in the GPU warp
2      if vertex_insertion M_u^+
3          partition[u] ← pseudo
4          pos ← atomicAdd(vertex_in_pseudo_size, 1)
5          vertex_in_pseudo[pos] = u
6      else if edge_insertion M_(u,v)^+ or edge_deletion M_(u,v)^-
7          affected_vertex[u] ← true ; affected_vertex[v] ← true
   // assign each u in affected_vertex to a GPU warp
8  parallel for each thread in the GPU warp
9      if partition[u] == pseudo
10         return
11     lane_id ← thread index in the GPU warp
12     bucket_cnt ← 0
13     bucket_start ← bucket_ptr[u]
14     num_bucket ← bucket_ptr[u + 1] - bucket_start
15     cur_par ← partition[u]
16     while bucket_cnt < num_bucket
17         nbr ← bucket_list[bucket_start + bucket_cnt × 32 + lane_id]
18         nbr_par ← nbr == ∅? ∅ : partition[nbr]
19         adj_ext += __popc(__ballot_sync(FULL, nbr_part != cur_par
             && nbr != ∅))
20         adj_int += __popc(__ballot_sync(FULL, nbr_part == cur_par))
21         bucket_cnt++
22     if adj_ext > adj_int && lane_id == 0
23         pos ← atomicAdd(vertex_in_pseudo_size, 1)
24         vertex_in_pseudo[pos] = u
   // assign u in vertex_in_pseudo to a GPU thread
25 parallel for each GPU thread
26     partition[u] ← pseudo
```

1) Partition Balancing: Incremental graph modifications can cause the partition to become imbalanced. To address this issue, we temporarily move newly added vertices to a *pseudo-partition* to prevent them from increasing the current partition weights. Additionally, to maintain the balance constraint after removing vertices, we also move the affected vertices to the pseudo-partition to reduce partition weights.

A vertex is considered affected by graph modifiers if: (1) it has been directly modified (e.g., its edges have been deleted or inserted), or (2) its neighbors have been modified, as changes in their surrounding structure may require refinement. However, not all affected vertices require refinement. If a vertex has as many or more adj_{int} than adj_{ext}, moving it to another partition will not reduce the cut size and may even increase it. To avoid this, we filter out such vertices and do not move them to the pseudo-partition. We use a centralized buffer, $vertex_in_pseudo$, to store vertices in the pseudo-partition, as they are often scattered across different parts of the graph, making it challenging to balance the workload across GPU threads. By aggregating these vertices to the buffer, we can dynamically assign GPU threads to handle them, ensuring balanced workloads and significantly increasing GPU performance.

Algorithm 3 presents our partition balancing algorithm. We use an array, $affected_vertex$, of size $|V|$ to record whether a vertex is affected. We assign each graph modifier to a GPU warp to mark directly modified vertices as affected. Threads assigned to M_u^+ move the vertex to the pseudo-partition immediately by changing its partition in $partition$ array, which records the partition assignment of each vertex, and insert it to $vertex_in_pseudo$ (lines 2-5). On the other hand, threads handling $M_{(u,v)}^+$ or $M_{(u,v)}^-$ mark u and v as affected (lines 6-7). Then, we check if they can be filtered out. To do this, we assign each vertex in $affected_vertex$ to a GPU warp. Threads assigned to vertices already in the pseudo-partition terminate early (lines 9-10), while the remaining threads process one bucket at a time, with each thread fetching a neighbor and its corresponding partition (lines 17-18). Threads efficiently calculate external and internal neighbors using the warp-level primitive `__ballot_sync` to evaluate each neighbor's partition, storing the results in a bitmask and counting them with the warp-level primitive `__popc` (lines 19-20). They then continue processing the next bucket until all are completed. Next, the first thread in the warp checks if the vertex has fewer adj_{int} than adj_{ext} (line 22). If so, it adds the vertex to $vertex_in_pseudo$, without immediately updating its partition in the $partition$ array (lines 23-24).

The proposed strategy in Algorithm 3 prevents data races by deferring partition updates until threads in other warps have completed their calculations. Once all affected vertices have been processed, we launch another GPU kernel to update the partitions of vertices in $vertex_in_pseudo$. We then move vertices with affected neighbors by assigning each vertex u in the pseudo-partition to a GPU warp. Threads in the same warp mark all neighbors of u as affected, using the same approach to filter out the neighbors whose adj_{ext} is less than adj_{int} and move the rest to the pseudo-partition.

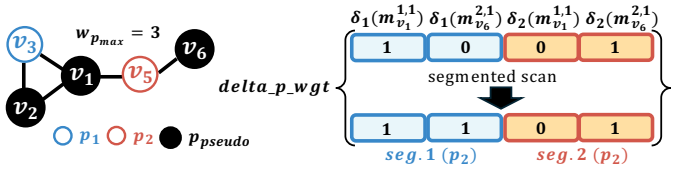


Fig. 5. Illustration of constructing $delta_p_wgt$ to calculate the accumulated delta partition weights for two partitions, p_1 and p_2, across two vertex moves using segmented scan. All vertices have a weight equal to one.

2) Parallel Refinement: Once the partition is balanced and affected vertices are in the pseudo-partition, we refine them by moving each vertex to its most suitable partition in parallel. We define the most suitable partition for a vertex u as the partition to which moving u from the pseudo-partition introduces the smallest cut size (i.e., the partition with most of u's neighbors), while maintaining the balanced partition. However, moving adjacent vertices in parallel requires costly synchronization to determine the most suitable partition accurately [13]. For example in Figure 5, if vertices v_1 and v_2 move concurrently from the pseudo-partition to other partitions, v_1 may initially select either p_1 or p_2 as its most suitable partition. However, if v_2 moves to p_1, v_1 should update its choice to p_1. Without synchronization between v_1 and v_2, v_1 can select the most suitable partition incorrectly. To overcome this synchronization challenge, we move non-adjacent vertices from $vertex_in_pseudo$ in parallel. For each non-adjacent vertex u, we create a vertex move $m_u^{par,\#nbr}$, where par is u's most suitable partition, and $\#nbr$ is the number of neighbors in par, and insert the vertex move to the $vertex_moves$ buffer to form a sequence of vertex moves.

Algorithm 4 presents our parallel refinement algorithm. To find non-adjacent vertices from $vertex_in_pseudo$, we assign each vertex u in $vertex_in_pseudo$ to a GPU warp. Threads in the warp cooperatively process the bucket of u one at a time, with each thread fetching a neighbor and its partition (lines 6-7). Threads then efficiently check if any thread has a neighbor in the pseudo-partition with a vertex ID less than u's vertex ID using `__any_sync` (line 8). If such a neighbor exists, threads terminate early, as u is not selected to move because its neighbor is being moved in this iteration (lines 9-10). Threads that do not find such neighbors then continue to check neighbors stored in other buckets. Next, threads with a vertex selected to move cooperatively identify the most suitable partition for u by counting u's neighbors in each partition p with a weight that does not exceed $W_{p_{max}}$. To accomplish this, each thread fetches a neighbor stored in a bucket along with its partition and uses the same strategy (i.e., warp-level primitive) as in Algorithm 3 to efficiently count neighbors belonging to p (lines 15-19). After processing

all buckets, the threads update max_nbr and max_nbr_p, stored in shared memory, if either 1) the number of neighbors in partition p exceeds the current maximum max_nbr, or 2) p has the same number of neighbors as max_nbr but a lower partition weight (line 20). Finally, the first thread in the warp creates a vertex move and inserts it to the $vertex_moves$ (lines 21-23).

After finding a sequence of vertex moves, we need to select a subsequence that, when applied, satisfies the balance constraint while introducing minimal cut size. To achieve this, we first sort the vertex moves in descending order by $\#nbr$, prioritizing vertices with strong connections with their most suitable partition. To find a subsequence that satisfies the balance constraint, we create a $delta_p_wgt$ array with j segments, each corresponding to a partition and with a length equal to the number of vertex moves, to record changes in partition weights if those vertex moves are applied. Each element in $delta_p_wgt$ records the delta partition weight of a vertex move for a partition. We define the delta partition weight of a vertex move, $m^{par,\#nbr}$ for a partition par as follows:

$$\delta_i(m_u^{par,\#nbr}) = \begin{cases} W_u, & i = par \\ 0, & otherwise \end{cases}$$

We then perform a parallel segmented scan on $delta_p_wgt$ to accumulate these changes for each partition.

Figure 5 illustrates the calculation of accumulated delta partition weights for two partitions, performed using a parallel segmented scan with two vertex moves. Each element in $delta_p_wgt$ records the delta partition weight for a partition after applying a vertex move. The first two elements (i.e., segment 1) correspond to partition p_1, while the last two elements (i.e., segment 2) correspond to partition p_2. For example, $delta_p_wgt[0]$ records the partition weight change for the first vertex move in partition p_1. We then perform a parallel segmented scan on $delta_p_wgt$ to accumulate these changes for each partition. After the scan, the s^{th} element in each segment holds the accumulated change in partition weight from the first to the s^{th} vertex move in each partition, representing the accumulated weight change over this subsequence. We then identify the longest subsequence of vertex moves that satisfies the balance constraint to apply as many vertex moves as possible in parallel. In this example, both vertex moves can be applied, as neither p_1's partition weight plus $delta_p_wgt[1]$ nor p_2's plus $delta_p_wgt[3]$ exceeds W_{pmax}. We then repeat the same process until all vertices are moved from the pseudo-partition.

Algorithm 4: Parallel Refinement

```
   Input: bucket_list, bucket_ptr, partition
   Input: vertex_moves, vertex_moves_size initialized to zero
   Shared memory: max_nbr_p, max_nbr initialized to zero
   // assign a vertex u in vertex_in_pseudo to a GPU warp
 1 parallel for each thread in the GPU warp
 2     lane_id ← thread index in the GPU warp
 3     bucket_start ← bucket_ptr[u]
 4     num_bucket ← bucket_ptr[u + 1] - bucket_start
 5     while bucket_cnt < num_bucket
 6         nbr ← bucket_list[bucket_start + bucket_cnt × 32 + lane_id]
 7         nbr_par ← nbr == ∅? ∅ : partition[nbr]
 8         if_adj_move = __any_sync(FULL, nbr_par == pseudo && nbr
             < u)
 9         if if_adj_move ≠ 0
10             return
11         bucket_cnt++
12     for p ∈ {1 . . . k} where W_p < W_pmax
13         bucket_cnt ← 0
14         num_nbr_in_p ← 0
15         while bucket_cnt < num_bucket
16             nbr ← bucket_list[bucket_start + bucket_cnt × 32 + lane_id]
17             nbr_par ← nbr == ∅? ∅ : partition[nbr]
18             num_nbr_in_p += __popc(__ballot_sync(FULL, nbr_part
                 == p))
19             bucket_cnt++
20         cmp and update max_nbr and max_nbr_p in shared memory
21         if lane_id == 0
22             pos ← atomicAdd(vertex_moves_size, 1)
23             vertex_moves[pos] ← m_u^{par,#nbr}
24 find the longest subsequence in parallel (Figure 5)
```

VI. Experimental Evaluation

We evaluated the performance of iG-kway using seven industrial circuit graphs generated by [13], [43]. Additionally, we tested iG-kway on three large non-circuit graphs (coAuthorsCiteseer, adaptive, and NLR) from the DIMACS Graph Partitioning Challenge [44] to demonstrate its applicability beyond CAD algorithms. Table I lists the statistics of each graph. In our experiment, we applied 100 incremental iterations based on the setting of TAU 2015 Incremental Timing Contest [41], where each iteration involves tens to hundreds of design modifiers that randomly remove/insert vertices and edges from/into the graph.

We consider G-kway [13], a state-of-the-art GPU-accelerated k-way graph partitioner, as our baseline. To have a fair comparison with G-kway and focus on incrementality, we replace its coarsening algorithm with our constrained coarsening to achieve better performance. Furthermore, since G-kway counts on the CPU to generate a graph CSR on the GPU, we modify the graph and regenerate its CSR for each incremental iteration, then apply G-kway to partition the modified graph. Hereafter, we refer to this baseline as *G-kway*†.

We implemented iG-kway and G-kway† using C++17 and CUDA 12.0 and compiled them with nvcc on a host compiler of GCC-8 with -O3 enabled. We ran experiments on a 64-bit Linux machine with 16 Intel i7-11700 CPU cores at 2.50 GHz and 128 GB RAM. Our GPU was an A6000 with 48 GB memory. In all experiments, we set the imbalance ratio (ϵ) to 3%, the group size (s) to six, and terminated the coarsening algorithm when the number of vertices dropped below $35 \times k$ or when fewer than 90% of the vertices could be coarsened. All results are averaged over 10 runs.

A. Overall Performance Comparison

Table I compares the runtime and cut size between iG-kway and G-kway† at $k = 2$. To highlight the advantages of iG-kway, we break down the runtime into graph modification and partitioning. Since G-kway† does not support dynamic updates of CSR on GPU, iG-kway is always faster than G-kway† in graph modification. We attribute this runtime advantage to iG-kway's GPU-aware data structure, which stores each vertex's neighbors in buckets to rapidly respond to graph modifiers. In contrast, G-kway†'s CSR data structure is relatively static, requiring a complete rebuild to modify the graph at each iteration. Consequently, for large graphs (e.g., mem_ctrl), modification can become a significant bottleneck for G-kway†, whereas iG-kway's modification time remains consistently small, regardless of graph size.

In terms of partitioning time, iG-kway outperforms G-kway† across all graphs with an average speedup of 84×. This speedup is due to iG-kway's incremental refinement algorithm, which identifies and refines only the vertices affected by graph modifiers, eliminating the need for re-partitioning required by G-kway†. Regarding the cut size, iG-kway finds a cut size comparable to G-kway† for all graphs. We attribute this to the effectiveness of our incremental refinement algorithm, which first moves vertices to a pseudo-partition to restore balance, and then moves them again to reduce the cut size.

Figure 6 details the comparison of partitioning time and cut size over 100 incremental iterations for the circuit usb under two different values of k. At the first iteration, we do not observe a significant runtime difference between iG-kway and G-kway† since both are FGP. However,

TABLE I

OVERALL COMPARISON OF RUNTIME (MODIFICATION AND PARTITIONING) AND CUT SIZE BETWEEN IG-KWAY AND G-KWAY† AT $k = 2$. ALL TIMES ARE MEASURED IN SECONDS. A CUT SIZE IMPROVEMENT GREATER THAN ONE INDICATES IG-KWAY CAN FIND A BETTER CUT SIZE.

Benchmark			Modification Time (s)		Partitioning Time (s)			Cut Size		
Name	# Vertices	# Edges	iG-kway	G-kway†	iG-kway	G-kway†	Speedup	iG-kway	G-kway†	Impr.
tv80	3,901,702	5,298,851	0.02	0.36	0.18	14.88	82.67×	**4,721**	4,774	1.01
mem_ctrl	32,445,075	42,670,885	0.11	3.37	0.58	46.07	79.43×	5,945	**5,659**	0.95
usb	139,479	180,510	0.01	0.01	0.12	10.16	84.67×	5,798	**5,701**	0.98
vga_lcd	1,869,688	23,447,678	0.07	2.13	0.38	31.27	82.29×	502	**496**	0.99
wb_dma	9,646,140	12,208,324	0.04	1.04	0.26	20.75	79.81×	**5,483**	5,489	1.00
systemcase	10,897,616	14,386,851	0.04	1.10	0.28	22.61	80.75×	4,670	**4,699**	1.00
des_perf	303,690	387,292	0.01	0.03	0.13	10.98	84.46×	**5,097**	5,150	1.01
coAuthorsCiteseer	227,320	814,134	0.01	0.03	0.13	11.20	86.15×	25,853	**25,537**	0.99
adaptive	6,815,744	13,624,320	0.03	0.97	0.51	50.12	98.27×	**1,809**	2,029	1.12
NLR	4,163,763	2,487,976	0.02	1.02	0.25	21.64	86.56×	4,611	**4,600**	1.00
Average							84.51×			1.00

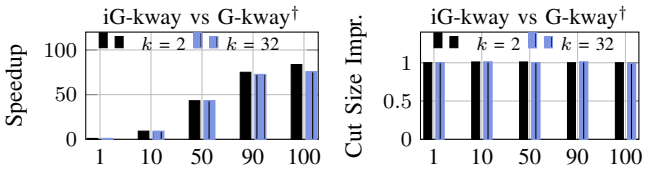


Fig. 6. Speedup (left) and cut size improvement (right) of iG-kway over G-kway† for the usb circuit over 100 incremental iterations.

as the number of incremental iterations increases, iG-kway's runtime advantage over G-kway† becomes more pronounced. The speedup of iG-kway grows proportionally with the number of incremental iterations for both k values. In terms of cut size, iG-kway achieves a comparable value to G-kway† across all iterations (within $\pm 3\%$).

B. Runtime and Cut Size Analysis under Varying k

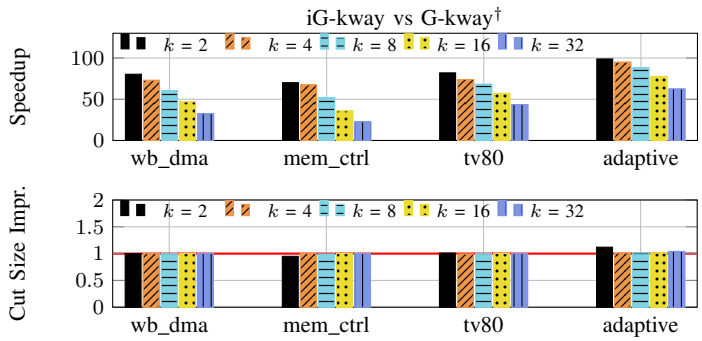


Fig. 7. The speedup (top) and cut size improvement (bottom) of iG-kway over G-kway† at different k values. A cut size improvement above one indicates that iG-kway can find a better cut size.

Figure 7 shows the speedup and cut size improvement of iG-kway over G-kway† at $k = \{2, 4, 8, 16, 32\}$ on three circuit graphs (wb_dma, mem_ctrl, and tv80) and a large non-circuit graph (adaptive). Regardless of k, iG-kway is consistently faster than G-kway†. iG-kway achieves up to a 98× speedup over G-kway† at $k = 2$. However, as k increases, the speedup decreases because iG-kway needs to examine more partitions to determine a suitable partition for each affected vertex. Despite this, iG-kway still achieves up to a 62× speedup at $k = 32$. Regarding the cut size, iG-kway achieves a comparable result with G-kway† on different k. These results highlight the effectiveness and efficiency of iG-kway over G-kway† in incremental graph partitioning.

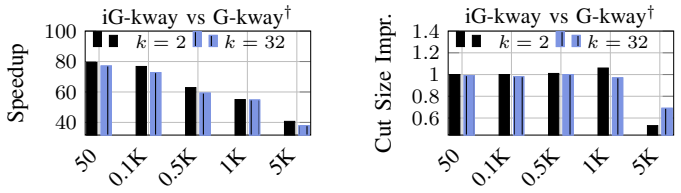


Fig. 8. The speedup (left) and cut size improvement (right) of iG-kway over G-kway† for the usb circuit across 100 incremental iterations each with varying numbers of graph modifiers (50–5K).

C. Incrementality Analysis

Figure 8 shows the speedup (left) and cut size improvement (right) achieved by iG-kway over G-kway† for the circuit usb across 100 incremental iterations each with varying numbers of graph modifiers (ranging from 50 to 5K) per iteration. When the number of graph modifiers per iteration is small, the advantage of iG-kway is more remarkable. For instance, with 50 graph modifiers, iG-kway is up to 80× faster than G-kway† while obtaining a comparable cut size. However, as the number of graph modifiers per iteration increases, the speedup decreases due to the growing number of affected vertices. More affected vertices require iG-kway to spend more time refining the partition. When the number of graph modifiers exceeds 5K per iteration, iG-kway struggles to find a partition with a decent cut size. This happens because after applying many graph modifiers (e.g., 5K×100 in this case), the graph becomes very different from its original form. This large difference makes it difficult for iG-kway to effectively optimize the graph, as its incremental refinement strategy relies on local graph structure. In such cases, applications can resort to FGP using G-kway†, especially when the number of graph modifiers reaches 50% of the graph's size.

VII. CONCLUSION

In this paper, we have introduced iG-kway, a GPU-parallel incremental k-way graph partitioner. iG-kway introduces an incrementality-aware data structure to support graph modifications directly on GPU. Atop this data structure, iG-kway introduces a GPU kernel algorithm that can efficiently refine affected vertices after the graph is incrementally modified, with minimal impact on partitioning quality. Experimental results show that iG-kway achieves an average speedup of 84× over the state-of-the-art G-kway, with comparable cut sizes.

ACKNOWLEDGMENT

This project is supported by NSF grants 2235276, 2349144, 2349143, 2349582, and 2349141, as well as the Research Grants Council of Hong Kong SAR (No. CUHK14207523).

REFERENCES

[1] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed computing*, vol. 48, no. 1, pp. 96–129, 1998.

[2] M. S. Gilbert, K. Madduri, E. G. Boman, and S. Rajamanickam, "Jet: Multilevel graph partitioning on graphics processing units," *SIAM Journal on Scientific Computing*, vol. 46, no. 5, pp. B700–B724, 2024.

[3] X. Sui, D. Nguyen, M. Burtscher, and K. Pingali, "Parallel graph partitioning on multicore architectures," in *Languages and Compilers for Parallel Computing: 23rd International Workshop, LCPC 2010, Houston, TX, USA, October 7-9, 2010. Revised Selected Papers 23*. Springer, 2011, pp. 246–260.

[4] D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 225–236.

[5] ——, "A parallel hill-climbing refinement algorithm for graph partitioning," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 236–241.

[6] Y. Akhremtsev, P. Sanders, and C. Schulz, "High-quality shared-memory graph partitioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2710–2722, 2020.

[7] L. Gottesbüren, T. Heuer, P. Sanders, C. Schulz, and D. Seemaier, "Deep multilevel graph partitioning," *arXiv preprint arXiv:2105.02022*, 2021.

[8] S. V. Patil and D. B. Kulkarni, "Graph partitioning using heuristic kernighan-lin algorithm for parallel computing," in *Next Generation Information Processing System: Proceedings of ICCET 2020, Volume 2*. Springer, 2021, pp. 281–288.

[9] P. Sanders and D. Seemaier, "Distributed deep multilevel graph partitioning," in *European conference on parallel processing*. Springer, 2023, pp. 443–457.

[10] M. F. Faraj and C. Schulz, "Recursive multi-section on the fly: Shared-memory streaming algorithms for hierarchical graph partitioning and process mapping," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 2022, pp. 473–483.

[11] P. Sanders and D. Seemaier, "Brief announcement: Distributed unconstrained local search for multilevel graph partitioning," in *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 443–445. [Online]. Available: https://doi.org/10.1145/3626183.3660257

[12] I. Bustany, G. Gasparyan, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "An open-source constraints-driven general partitioning multi-tool for vlsi physical design," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[13] W. L. Lee, D.-L. Lin, T.-W. Huang, S. Jiang, T.-Y. Ho, Y. Lin, and B. Yu, "G-kway: Multilevel GPU-Accelerated k-way Graph Partitioner," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[14] B. Goodarzi, F. Khorasani, V. Sarkar, and D. Goswami, "High performance multilevel graph partitioning on gpu," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2019, pp. 769–778.

[15] B. Goodarzi, M. Burtscher, and D. Goswami, "Parallel graph partitioning on a cpu-gpu architecture," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 58–66.

[16] D.-L. Lin, H. Ren, Y. Zhang, B. Khailany, and T.-W. Huang, "From RTL to CUDA: A GPU Acceleration Flow for RTL Simulation with Batch Stimulus," in *ACM International Conference on Parallel Processing (ICPP)*, 2022.

[17] D.-L. Lin, Y. Zhang, H. Ren, S.-H. Wang, B. Khailany, and T.-W. Huang, "GenFuzz: GPU-accelerated Hardware Fuzzing using Genetic Algorithm with Multiple Inputs," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.

[18] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2022.

[19] T.-W. Huang, C.-X. Lin, G. Guo, and M. Wong, "Cpp-Taskflow: Fast Task-based Parallel Programming using Modern C++," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019.

[20] T.-W. Huang, G. Guo, C.-X. Lin, and M. D. F. Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

[21] L. Durbeck and P. Athanas, "Incremental streaming graph partitioning," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–8.

[22] W. Ju, J. Li, W. Yu, and R. Zhang, "igraph: an incremental data processing system for dynamic graph," *Frontiers of Computer Science*, vol. 10, pp. 462–476, 2016.

[23] W. Fan, M. Liu, C. Tian, R. Xu, and J. Zhou, "Incrementalization of graph partitioning algorithms," *Proceedings of the VLDB Endowment*, vol. 13, no. 8, pp. 1261–1274, 2020.

[24] D. Dai, W. Zhang, and Y. Chen, "Iogp: An incremental online graph partitioning algorithm for distributed graph databases," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, 2017, pp. 219–230.

[25] C.-W. Ou and S. Ranka, "Parallel incremental graph partitioning," *IEEE transactions on Parallel and Distributed Systems*, vol. 8, no. 8, pp. 884–896, 1997.

[26] S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Gamer: Gpu-accelerated maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 583–593, 2022.

[27] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Fastgr: Global routing on cpu–gpu with heterogeneous task graph scheduler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2317–2330, 2022.

[28] Z. Guo, T.-W. Huang, and Y. Lin, "GPU-accelerated static timing analysis," in *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, 2020.

[29] G. Guo, T.-W. Huang, and M. D. F. Wong, "Fast STA Graph Partitioning Framework for Multi-GPU Acceleration," in *IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, 2023.

[30] G. Guo, T.-W. Huang, Y. Lin, Z. Guo, S. Yellapragada, and M. Wong, "A GPU-Accelerated Framework for Path-Based Timing Analysis," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[31] G. Guo, T.-W. Huang, Y. Lin, and M. Wong, "GPU-accelerated Critical Path Generation with Path Constraints," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021.

[32] ——, "GPU-accelerated Path-based Timing Analysis," in *IEEE/ACM Design Automation Conference (DAC)*, 2021.

[33] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. Young, and M. Wong, "Gcs-timer: Gpu-accelerated current source model based static timing analysis," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3655983

[34] T. Liu, L. Chen, X. Li, M. Yuan, and E. F. Young, "Finemap: A fine-grained gpu-parallel lut mapping engine," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 392–397.

[35] E. F. Young, "Gpu acceleration in physical synthesis," in *Proceedings of the 2023 International Symposium on Physical Design*, ser. ISPD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 167. [Online]. Available: https://doi.org/10.1145/3569052.3578912

[36] J. Jiang, L. Zou, W. Zhao, Z. He, T. Chen, and B. Yu, "Pdrc: Package design rule checking via gpu-accelerated geometric intersection algorithms for non-manhattan geometry," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657367

[37] P. Liao, Y. Zhao, D. Guo, Y. Lin, and B. Yu, "Analytical die-to-die 3-d placement with bistratal wirelength model and gpu acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 6, pp. 1624–1637, 2024.

[38] Z. He, Y. Zuo, J. Jiang, H. Zheng, Y. Ma, and B. Yu, "Opendrc: An efficient open-source design rule checking engine with hierarchical gpu acceleration," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[39] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A gpu-enabled level-set method for mask optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 594–605, 2023.

[40] Z. He, Y. Ma, and B. Yu, "X-check: Gpu-accelerated design rule checking via parallel sweepline algorithms," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549383

[41] J. Hu, G. Schaeffer, and V. Garg, "TAU 2015 contest on incremental timing analysis," in *IEEE/ACM ICCAD*, 2015, pp. 882–889.

[42] D. Guide, "Cuda c++ programming guide," *NVIDIA, July*, 2020.

[43] T.-W. Huang and M. Wong, "OpenTimer: A High-Performance Timing Analysis Tool," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015.

[44] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, "10th dimacs implementation challenge workshop," 2012.

PacQ: A SIMT Microarchitecture for Efficient Dataflow in Hyper-asymmetric GEMMs

Ruokai Yin
Yale University
New Haven, USA
ruokai.yin@yale.edu

Yuhang Li
Yale University
New Haven, USA
yuhang.li@yale.edu

Priyadarshini Panda
Yale University
New Haven, USA
priya.panda@yale.edu

***Abstract*—Weight-only quantization has been widely explored in large language models (LLMs) to reduce memory storage and data loading overhead. During deployment on single-instruction-multiple-threads (SIMT) architectures, weights are stored in low-precision integer (INT) format, while activations remain in full-precision floating-point (FP) format to preserve inference accuracy. Although memory footprint and data loading requirements for weight matrices are reduced, computation performance gains remain limited due to the need to convert weights back to FP format through unpacking and dequantization before GEMM operations. In this work, we investigate methods to accelerate GEMM operations involving packed low-precision INT weights and high-precision FP activations, defining this as the hyper-asymmetric GEMM problem. Our approach co-optimizes tile-level packing and dataflow strategies for INT weight matrices. We further design a specialized FP-INT multiplier unit tailored to our packing and dataflow strategies, enabling parallel processing of multiple INT weights. Finally, we integrate the packing, dataflow, and multiplier unit into PacQ, a SIMT microarchitecture designed to efficiently accelerate hyper-asymmetric GEMMs. We show that PacQ can achieve up to $1.99\times$ speedup and 81.4% reduction in EDP compared to weight-only quantized LLM workloads running on conventional SIMT baselines.**

I. Introduction

Large language models (LLMs) have demonstrated exceptional performance across a wide range of complex natural language processing tasks [19], [23]. However, their substantial parameter counts introduce significant deployment challenges, including increased memory footprint and computational costs. Quantization has emerged as a popular solution to improve the deployment efficiency of LLMs, particularly on edge devices.

Conventional deep neural network (DNN) quantization reduces both computation and memory traffic by compressing weights and activations into low-precision integers [5], [8]. However, quantizing activations in LLMs is particularly challenging due to their dynamic range and the occurrence of salient values [10], [21]. As a result, many recent studies have focused on weight-only quantization, where weights are compressed to very low-precision integers (e.g., $\leq$INT4), while activations are preserved in high-precision floating-point formats, typically FP16 [2], [9], [10], [20], [21].

This work was supported in part by CoCoSys, a JUMP2.0 center sponsored by DARPA and SRC, the National Science Foundation (CAREER Award, Grant #2312366, Grant #2318152), the DARPA Young Faculty Award and the DoE MMICC center SEA-CROGS (Award #DE-SC0023198).

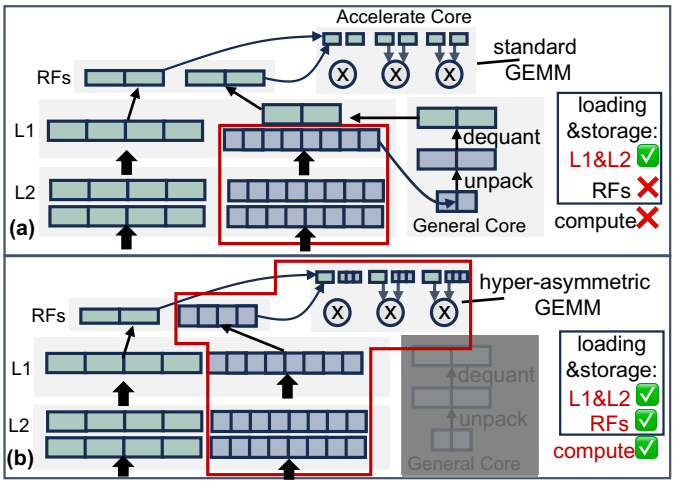


Fig. 1. (a) Standard inference flow on SIMT architecture for weight-only quantized LLMs. Green denotes high-precision floating-point activations, and blue denotes low-precision integer weights. (b) Our proposed flow leverages the hyper-asymmetric GEMM to achieve the benefits of reduced loading, storage, and compute costs from low-precision integer weights throughout the entire GEMM computation stack.

While weight quantization significantly reduces memory traffic for LLMs (e.g., Llama2-70B [19] requires 131.6 GB with FP16 weights but only 35.8 GB with INT4 weight-only quantization), it does not lower computational costs. This limitation stems from the current deployment workflow on single-instruction multiple-thread (SIMT) hardware, widely used in commercial ML accelerators such as NVIDIA GPUs. As shown in Figure 1(a), the inference deployment flow of weight-only quantized LLMs on SIMT hardware faces three major challenges: (1) ***Inefficient memory hierarchy utilization:*** Although low-precision integer (INT) weights are stored and fetched from off-chip DRAM in packed format, they are unpacked and dequantized into high-precision floating-point (FP) format upon loading into the L1 cache, losing memory benefits at and above the L1 cache. (2) ***Overhead of unpacking and dequantization:*** Unpacking and dequantizing weights introduce significant latency and computational overhead [10]. (3) ***Lack of computational savings:*** GEMM computations are performed in FP after dequantization, forfeiting the computational efficiency of low-precision integer weights.

Despite these limitations, existing weight-only quantization techniques often achieve inference speedup on SIMT

architectures in memory-bound scenarios, such as single-batch text generation. However, real-world LLM serving systems predominantly adopt multi-batch processing to achieve better hardware utilization [22]. Multi-batch processing, in contrast, is typically compute-bound, with its performance constrained by the three limitations outlined earlier.

To address the limitations in deploying weight-only quantized models on SIMT architectures, we propose to retain INT weights in packed format throughout the entire GEMM computation stack. We define this problem as **hyper-asymmetric GEMM**, where the operand precisions are highly unbalanced (asymmetric), as illustrated in Figure 1(b). In this work, we explore the design space for accelerating hyper-asymmetric GEMM on SIMT-based hardware. Our key contributions are:
(1) We pinpoint the critical role of INT weight packing direction, a factor often overlooked in prior work, for optimizing hyper-asymmetric GEMM. Specifically, we analyze the inefficiencies of packing weights along the input-feature (k) dimension, a common approach in existing quantized LLM deployment frameworks [1], [11]. To address this, we propose an alternative packing strategy and dataflow optimized for hyper-asymmetric GEMM. Our strategy has 54.3% reduction of register file accesses compared to baseline approaches.
(2) We propose a parallel FP-INT multiplier optimized for our packing and dataflow strategy. By identifying constant patterns in the FP representation of INT weight values, our design efficiently processes four FP16×INT4 or eight FP16×INT2 multiplications in parallel within a single cycle, reusing ∼73% of hardware resources from standard FP16 multipliers. This design achieves up to 6.8× throughput/watt improvement compared to conventional designs.
(3) We encapsulate the proposed packing, dataflow, and parallel multiplier design into PacQ, a SIMT microarchitecture tailored for hyper-asymmetric GEMM. Compared to NVIDIA GPUs as a baseline, PacQ achieves up to 81.4% EDP reduction for selected LLM inference workloads.

II. Background

SIMT Architecture. The single-instruction multiple-thread (SIMT) architectures are widely used in commercial ML accelerators, such as NVIDIA GPUs, to achieve high computational, memory, and control parallelism via sets of consecutively indexed threads (warps). These architectures tile large ML workloads (e.g., GEMMs) into smaller blocks mapped to individual threads, improving bandwidth efficiency and data reuse [18]. NVIDIA's Volta GPUs introduced the first streaming multiprocessors (SMs) with dedicated ML acceleration capabilities, including tensor cores (TCs), specialized for GEMM acceleration [14], [16]. Multiple SMs are interconnected via an on-chip network to a last-level cache backed up by HBM. Each SM contains sub-cores sharing an L1 cache, with sub-cores further divided into general-purpose cores and tensor cores. General-purpose cores handle operations like unpacking and dequantization, while tensor cores are optimized for GEMM operations. Operand data is stored in register files for efficient reuse before being processed by tensor cores.

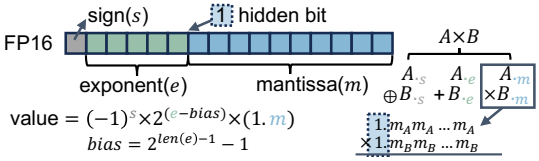


Fig. 2. Illustration of the IEEE 754 standard FP16 format.

Without loss of generality, our study examines the dataflow and microarchitecture of Volta's tensor cores as the baseline.
Floating Point Number. In Figure 2, we show the standard IEEE 754 format for FP16. In this work, we focus on the normalized format (i.e., the hidden bit for the mantissa is always 1). During the floating point multiplication, the hidden bit needs to be considered for the integer multiplication between the mantissas, as shown on the right side of Figure 2.
Related Work. Previous studies [4], [17] on accelerating mixed-precision GEMM for quantized DNNs have primarily targeted workloads with low operand precision imbalance (difference $\leq 4\times$). In contrast, our work focuses on hyper-asymmetric GEMM, characterized by a high operand precision imbalance (difference$\geq 4\times$). Recent work, FIGNA [6], introduced an FP-INT unit to eliminate unpacking and dequantization overhead during the deployment of weight-only quantized LLMs. However, their approach emphasizes co-designing the FP-INT multiplier with a block floating-point format. In contrast, our work integrates the FP-INT multiplier design with packing and dataflow strategies for processing packed INT weights. Furthermore, our design improves throughput by enabling parallel multiplication of FP activations with multiple INT weights, a capability not explored in FIGNA.

Other methods for accelerating mixed-precision GEMM in weight-only quantized LLMs employ lookup table (LUT) techniques, either by designing new LUT-friendly hardware [12] or focusing on single-batch text generation on GPUs [15]. In contrast, our work enhances general SIMT-based architectures to support multi-batch inference, a scenario more common in real-world LLM deployments. Additionally, approaches targeting efficient dequantization kernels [10], [24] address complementary challenges and are orthogonal to our work.

III. Packing flow for Hyper-asymmetric GEMM

Basic GEMM Dataflow. For illustration, we describe the Volta GPU's execution of the FP16 tensor core MMA instruction *mma.sync.m16n16k16* for *W16A16*, where both weights and activations are in FP16 precision, as shown in Figure 3. In this example, each matrix has a size of 16×16. As shown in Figure 3(a), matrices A, B, and C are collaboratively fetched by a warp consisting of 32 threads. The workload is then evenly distributed among four *octets* [16] (groups of 8 threads), as illustrated in Figure 3(b). As depicted by Figure 3(c), each *octet* fetches tiles using a weight-stationary dataflow, represented by the left three-nested-for-loop, and computations within each tile follow an output-stationary dataflow, represented by the right three-nested-for-loop. Finally, Figure 3(d) illustrates how octet-level workloads are mapped onto the hardware (i.e., tensor cores in Volta GPUs). A 4×4 tile of

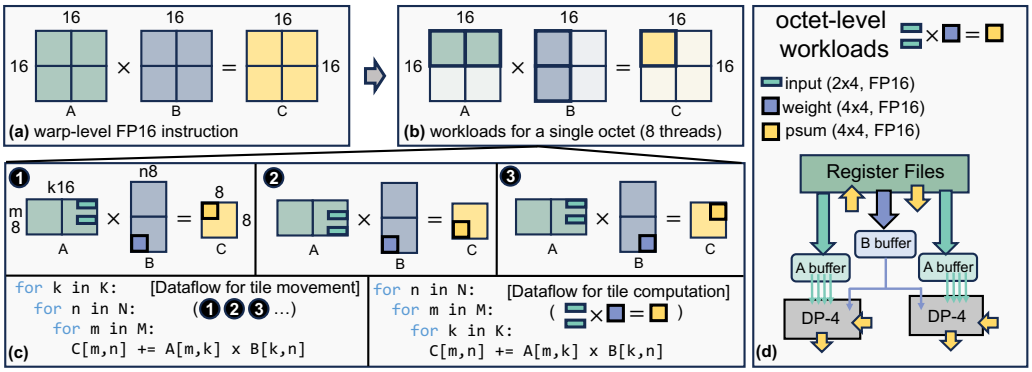


Fig. 3. (a) & (b): The tile mapping of the warp-level (a set of 32 threads) *mma.sync.m16n16k16* instruction to different octets (a subset of 8 threads). Here, A represents the input matrix with a shape of $[m, k]$, B represents the weight matrix with a shape of $[k, n]$, and C represents the output (partial-sum) matrix with a shape of $[m, n]$. (c) An example of a step-by-step breakdown of the octet-level data movement and computation flow. (d) The mapping of octet-level workloads onto the baseline tensor processing units. DP-4 stands for four-element dot-product units that performs 4×4 inner-product.

B is fetched from the register files into a buffer shared by all eight threads within the *octet*. Meanwhile, 2×4 tiles of A are loaded into separate buffers, each shared by four threads. Two four-element dot-product units (DP-4) are responsible to compute the 4×4 tile of C.

Hyper-asymmetric GEMM Packing and Dataflow. We define hyper-asymmetric GEMM as a GEMM operation where the operands A and B have a precision difference of at least $4\times$ (e.g., *W4A16* or *W2A16*). While largely overlooked in prior dequantization-based GEMM methods, the dimension along which the INT weights are packed becomes a critical factor in hyper-asymmetric GEMM. Formally, we define the packing of weight matrix B using the format $P(B_x)_y$, where x represents the number of elements of B packed along the y dimension. For instance, in *W4A16*, if the weights (B) are packed along the k dimension into the INT16 datatype, the packing can be formally described as $P(B_4)_k$. Once fetched to the L1 cache, the INT16 is unpacked to 4 INT4 weights and dequantized to FP16 for being executed in the standard FP16 GEMM.

Although there is limited research on the packing direction of INT weight matrices, nearly all prior weight-only LLM quantization frameworks [1], [3], [11] choose to pack weights along the input feature dimension (k-dim). While the packing dimension is unimportant in dequantization-based GEMM, as packed INT weights are unpacked at the L1 cache regardless of the packing direction, this is not the case for hyper-asymmetric GEMM. In hyper-asymmetric GEMM, INT operands are fetched in their packed form, creating hyper-asymmetry in the amount of data staged in the registers and tensor cores. An improper packing dimension combined with this hyper-asymmetry leads to significant performance degradation.

Specifically, when weights are packed along the k-dim, each packed INT weight fetch requires multiple activation fetch instructions because the k-dim must align between the two operands during GEMM computation. Figure 4(a) provides an example: for processing INT weights packed in the $P(B_4)_k$ format, four distinct fetch instructions for A must be issued to align the operands. Furthermore, packing INT weights (B) along the k-dim results in poor data reuse of activations (A). As illustrated in Figure 4(b), new data from A along the k-dim must be continuously fetched, leading to eviction of buffers inside the tensor cores and preventing data reuse of A. This causes excessive data access from large register files, which is highly power-intensive. When the operand precision asymmetry is significant, this issue can even escalate beyond the register file level to the L1 cache.

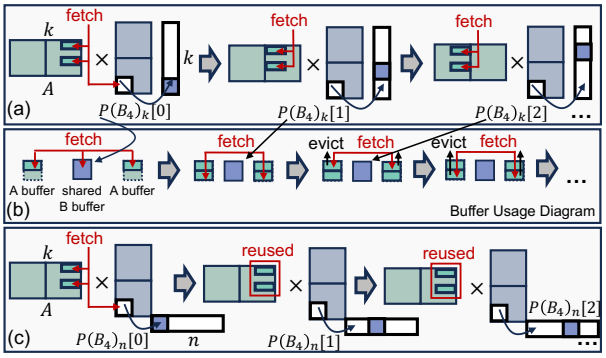


Fig. 4. (a) Illustration of the extra fetch instructions and (b) the poor data reuse of A when packing weights along the k dimension. (c) Illustration of the improved data reuse and reduction in fetch instructions achieved by packing along the n dimension. The tile size is identical to Figure. 3.

We argue that INT weights should instead be packed along the output feature (n) dimension. This approach avoids additional internal loops of fetch instructions compared to processing INT weights packed along the k-dim. As shown in Figure 4(c), once the activations (A) and packed weights ($P(B_4)_n$) are fetched following the standard flow, no extra fetch instructions are required. Furthermore, our proposed packing strategy ensures no eviction of A during the processing of packed B, enabling full reuse of activations across the packed weights. Moreover, instead of using the weight stationary dataflow for tile movement as in Figure 3(c), we use output stationary for both tile movement and tile computation. This packing and dataflow combination strikes a good balance between the data reuse and the partial sum traffic.

IV. Proposed Parallel FP-INT Multiplier

As discussed previously, our proposed packing and data flow reuses activations across all packed INT weights within

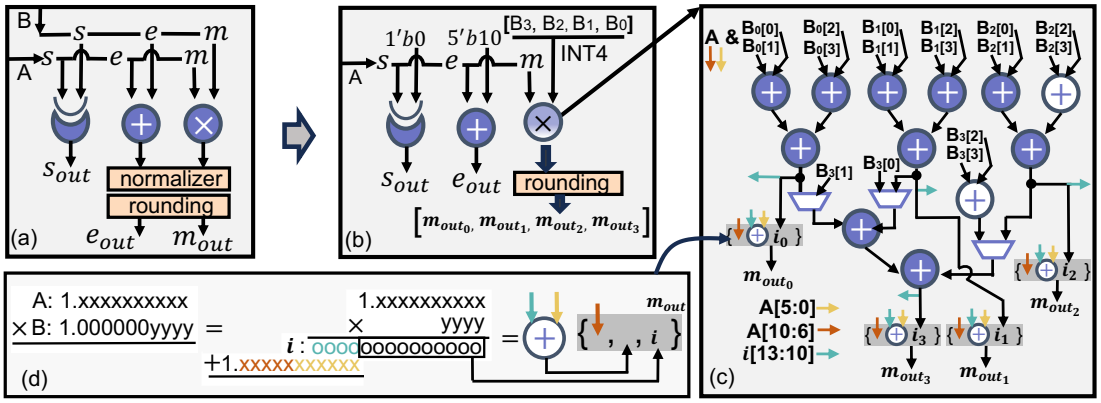


Fig. 5. (a) Standard FP multiplier design. s stands for sign, e represents exponent, and m denotes the mantissa. (b) Overview of our proposed parallel FP-INT-16 multiplier. In the provided example, our design can generate four output mantissas in one cycle. (c) Detailed diagram of the modifications made to the original 11-bit integer multiplier. The elements in purple are part of the original multiplier, while the elements in white represent additional units. All bits of B's mantissas perform a logical AND operation with shifted A's mantissas before being fed into the adders. (d) Explanation of how the final output mantissa (shaded gray) is assembled from different value sources. $\{\}$ denotes the append operation.

a single data fetch instruction. In the standard approach, these data are dequantized and processed sequentially by the pipelined FP multiplier. However, we identify an opportunity to perform the multiplication between a single FP activation and multiple INT weights in parallel. To exploit this, we propose a specialized parallel FP-INT multiplier.

Observation from FP representation of INT values. When representing an INT $x \in [1024, 2048)$ in FP16 format, certain patterns emerge during the conversion process. First, the FP16 representation of x consistently has an exponent value of 11001 (corresponding to $2^{25-15} = 1024$ in decimal). Second, the mantissa of x is always in the form of $10'b0|(x - 1024)$. Based on these patterns, we make further observations on representing low-precision INT weights in FP16 format. Without loss of generality, we use INT4 as an example, which can be easily extended to INT2 [7].

Assume $B \in [-8, 7)$ is a signed INT4 weight scalar. To simplify arithmetic operations, we first transform B to an unsigned representation by adding 8, resulting in $B+8 \in [0, 15)$. Since B is in INT4 format, we ensure that $B + 8 + 1024$ $(B + 1032)$ lies within the range $[1024, 2048)$. Leveraging the observed patterns, we conclude: ❶ The FP16 representation of $B + 1032$ always has an exponent value of 11001. ❷ The mantissa of $B+1032$ is consistently in the form $000000yyyy$, where $yyyy$ is the 4-bit integer representation of $B + 8$.

Proposed Design. Leveraging ❶ and ❷, we design a parallel FP-INT multiplier that maximally reuses the hardware resources of the original FP multiplier, as shown in Figure 5(a). Our design enables the multiplication of one FP16 input with four INT4 values or eight INT2 values in parallel, producing all outputs in a single cycle. For illustration, we focus on INT4.

Since B is transformed into an unsigned representation, the signs of all four outputs depend only on the sign of A. Thus, we compute the common output sign s_{out} by XOR-ing A's sign bit with 0. Using ❶, we derive the shared exponent e_{out} for all outputs by adding A's 5-bit exponent to 11001, as shown in Figure 5(b). Figure 5(c) depicts the details of our

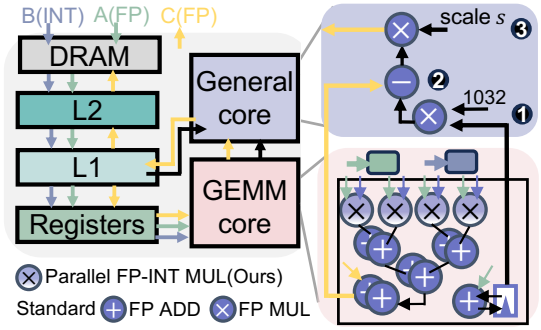


Fig. 6. Overview of the PacQ architecture. B represents the INT weight vectors, A represents the FP activation vectors, and C represents the outputs or partial sums in FP16. The steps illustrated in the general core reflect only the required instructions in sequence. PacQ does not require any hardware modifications to the general core.

proposed parallel FP-INT multiplier. The original FP16 integer multiplier requires 10 parallel adders (16-bit) to compute one 11-bit×11-bit multiplications (colored purple). Utilizing ❷, we break the original four 11-bit×11-bit multiplications into four 11-bit×4-bit multiplication. By introducing two additional 16-bit parallel adders (colored white), our design simultaneously performs all four 11-bit×4-bit multiplications.

Finally, as shown in Figure 5(d), we ensure the hidden bit of B's mantissa is accounted for by performing a 6-bit addition between the top-4 MSBs of the intermediate results i (colored turquoise, from 11-bit×4-bit) and the 6 LSBs of A (colored yellow). We then concatenate the top-5 MSBs of A (colored brown) with the addition results and i to obtain the four final mantissa products m_{out}. These results are passed to the rounding units and truncated to 10 bits (excluding the hidden bit). Since both A and B have normalized mantissa and B's mantissa values are constrained to a 4-bit unsigned integer, normalization to the outputs is unnecessary.

Overall Architecture (PacQ). We encapsulate our proposed packing and dataflow strategies along with the parallel FP-INT multiplier units into PacQ: a SIMT microarchitecture for accelerating hyper-asymmetric GEMMs. The overall architecture

is depicted in Figure 6. Our proposed microarchitecture retains most components from the Volta GPU baseline, requiring hardware modifications only in the GEMM acceleration core (i.e., tensor core). We introduce the following three key changes to the original tensor core:

First, we replace all original FP16 multipliers with our proposed parallel FP-INT multipliers. **Second**, we duplicate the original adder trees in the DP-4 by twice, enabling the accumulation of the inner product of 16 values in 2 cycles for INT4 weights (or 32 values in 4 cycles for INT2 weights). **Third**, we equip the GEMM core with small accumulators to store processed A values. Recall that before feeding the packed INT weights into the parallel FP-INT multipliers, we transform all weight values B into $B + 1032$. To retrieve the original weight values, we subtract 1032 later in the computation. By leveraging the small accumulators, we fuse this subtraction directly into the inner product calculation:

$$\sum_{0}^{k}(A_k(B_k - 1032)) = \sum_{0}^{k} A_k B_k - 1032 \times \sum_{0}^{k} A_k, \quad (1)$$

where k denotes the dimension of the inner product. The accumulators are responsible for generating the results of $\sum_0^k A_k$. As shown in the general core of Figure 6, the accumulated results are multiplied by 1032 in ❶, then subtracted from the inner product results ($\sum_0^k A_k B_k$) in ❷. Finally, the group quantization scales s are applied in ❸ to scale the outputs back to their expected range.

TABLE I
CONFIGURATION OF THE PACQ AND BASELINES.

INT11 MUL (baseline)	10 INT16 adders
Parallel INT11 MUL	12 INT16 adders, 4 INT6 adders
FP16 MUL (baseline)	1 INT11 MUL, 1 INT5 adder 1 normalization unit, 1 rounding unit
Parallel FP-INT-16 MUL	1 parallel INT11 MUL, 1 INT5 adder 1 normalization unit, 4 rounding units
FP-16 DP-4 (baseline)	4 FP16 MUL, 4 FP16 adders
Parallel FP-INT-16 DP-4	4 parallel FP-INT-16 MUL, 8 FP16 adders
Tensor Core	4 Parallel FP-INT-16 DP-4 baseline: 4 FP16 DP-4 2×3072-bits buffer, 256KB register files [14]
Streaming Multiprocessor	8 tensor cores, 96 KB shared L1 cache

V. EXPERIMENTAL RESULTS

Experimental Setup. For the packing and dataflow evaluation, we developed a custom simulator in Python to monitor memory access patterns and record execution cycles for PacQ and the baselines. Key components of PacQ and our baselines were implemented in RTL and synthesized using Synopsys Design Compiler at 400MHz with 32nm technology. We utilized CACTI7.0 [13] to model on-chip SRAM and register files for getting memory statistics. Configuration details for PacQ and the baseline architectures are summarized in Table I. Unless otherwise specified, in the rest of the section, INT2 and INT4 refer to the precision of weights, while activations and partial sums are always represented in FP16 format. For hardware unit comparisons, 'baseline' always refers to the standard baseline designs as denoted in Table. I. $P(B_{4(8)})_k$ denotes the baseline of hyper-asymmetric GEMM flow that packs 4 INT4 (8 INT2) weights in one INT16 along k-dim.

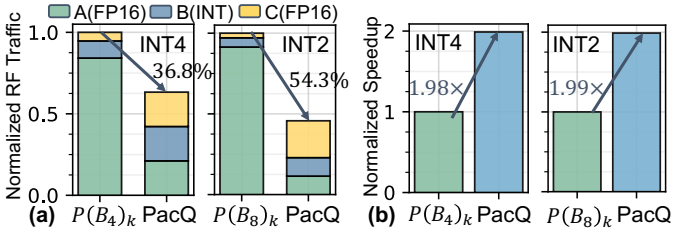


Fig. 7. (a) Comparison of the normalized number of register file accesses. (b) Comparison of normalized speedup between PacQ and hyper-asymmetric GEMM with INT weights packed along k. The workload is *m16n16k16*. $P(B_{4(8)})_k$ represents packing 4 INT4 weights (or 8 INT2 weights) into one INT16 data along the k dimension.

TABLE II
PERPLEXITY OF RTN-BASED PTQ LLAMA2 MODELS ON WIKITEXT-2 AND C4. W4A16-G128 REFERS TO 4-BIT WEIGHT-ONLY QUANTIZATION WITH A 128-GROUP SIZE ALONG k-DIM. G[32,4] INDICATES A 128-GROUP SIZE DISTRIBUTED AS 32 GROUPS ALONG k AND 4 GROUPS ALONG n.

Model Llama2-7B	W16A16 fp16 baseline	W4A16 g128	W4A16 g[32,4]	W4A16 g256	W4A16 g[64,4]
wikitext-2	5.47	5.73	5.72	5.75	5.77
C4	7.26	7.58	7.59	7.64	7.66

Proposed Packing and Data Flow. In Figure. 7(a), we present the number of register file accesses for PacQ compared to the hyper-asymmetric GEMM with INT4 (INT2) weights packed along k. The results demonstrate that with improved data reuse, our approach achieves up to a 54.3% reduction in register file accesses. Furthermore, in Figure. 7(b), we show the normalized speedup of PacQ compared to the hyper-asymmetric GEMM packing along k. Thanks to our parallel processing strategy, we achieve an average speedup of $1.99\times$.

We want to confirm here that PacQ ***does not require any quantization algorithm modifications*** to achieve hardware efficiency, as there is no approximation in our design. However, we observe that a small adjustment to the existing PTQ algorithm can further enhance the efficiency of PacQ. Specifically, by spanning the quantization group across both dimensions ($[n, k]$) instead of solely on the input feature dimension (k), we can reduce the number of fetches of the quantization scale s for PacQ in the general core, as illustrated in Figure 6-❸. We adapt the standard round-to-nearest (RTN) based PTQ algorithm to define quantization groups across both n and k dimensions within the existing framework [11]. The perplexity results for Llama2-7B [19] are shown in Table II. The results demonstrate that with these quantization group-dimension modifications, we achieve iso-perplexity with standard RTN 4-bit weight-only quantized models.

Parallel FP-INT Multiplier. We first evaluate the throughput / Watt to measure the overall performance of our proposed parallel FP-INT multiplier. Figure 8 compares the normalized performance of our design against the original FP16 multiplier and the FP16 DP-4 unit. The throughput of the original FP16 multiplier is 1, whereas it is 4 (8) for our parallel design with INT4 (INT2) weights. The original FP16 DP-4 unit requires

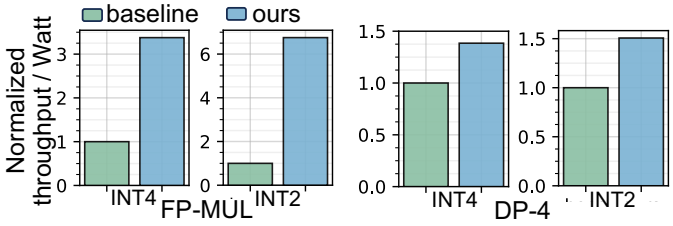


Fig. 8. Normalized performance comparison between the baseline FP16 designs with our parallel FP-INT-16 designs of running INT4 and INT2 weights. For DP-4, we consider the workload of *m2n4k4*.

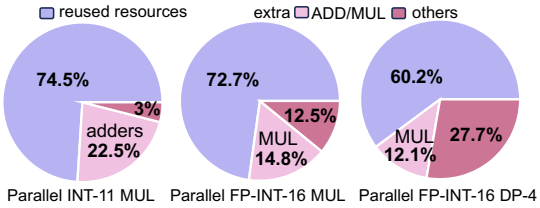


Fig. 9. Power breakdown of our proposed units. MUL stands for multiplier.

11 cycles to generate 8 FP16 outputs, while our parallel design takes 19 (35) cycles to generate 32 (64) FP16 outputs for INT4 (INT2) weights. Compared to the original FP16 multiplier, our design achieves 3.38× better performance for INT4 weights and 6.75× better performance for INT2. We further show the power breakdown of our proposed units in Figure 9. Our design objective is to maximize the reuse of hardware resources from the standard design. As shown by the purple-colored portions in the figure, we successfully reuse nearly 75% of the original INT-11 multiplier resources, which serve as the fundamental building blocks in our parallel FP-INT multiplier. For the DP-4 unit, we achieve approximately 60% hardware resource reuse. The observed decrease in the resource reuse ratio stems from the duplication of additional units; for instance, we needed to double the FP16 adder trees in the DP-4 design. Despite this, our design maintains an average hardware resource reuse ratio of 69%, which significantly minimizes resource duplication and enhances overall efficiency.

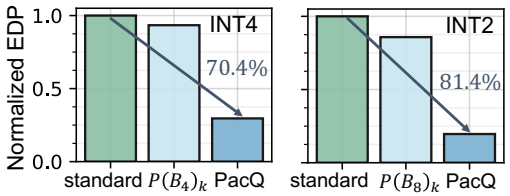


Fig. 10. Normalized EDP of PacQ vs. baselines. 'Standard' refers to the standard dequantization-based W16A16 GEMM. $P(B_{4(8)})_k$ denotes the hyper-asymmetric GEMM with INT4 (INT2) weights packed along k.

Other Results. In Figure. 10, we present the overall energy-delay-product (EDP) comparison of PacQ against Volta-like SIMT baselines. Two baseline cases are evaluated: the standard dequantization-based GEMM flow and the hyper-asymmetric GEMM flow with weights packed along the k dimension. Results demonstrate that PacQ achieves up to an 81.4% reduction in EDP for the workload $m16n4096k4096$, which represents a FFN layer in Llama2-7B with 16 batches.

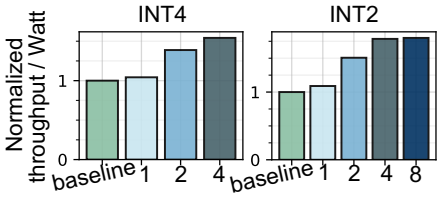


Fig. 11. Ablation study on the adder tree duplication. The number indicates the level of duplication of the adder trees (with a total of 4 FP16 adders in the baseline). For example, a duplication level of 4 would indicate 16 FP16 adders in our parallel FP-INT-16 DP-4 design.

In Figure. 11, we investigate the impact of duplicating FP16 adder trees in our parallel FP-INT DP-4 unit on overall performance (throughput / Watt) for the *m16n16k16* workload. We observe that while increasing the level of duplication continues to improve performance, a duplication factor of 2 provides the most significant gain. It achieves a 1.33× (1.38×) improvement over a duplication factor of 1 for INT4 (INT2). In comparison, a duplication factor of 4 results in only a 1.11× (1.18×) improvement over duplication factor 2 for INT4 (INT2).

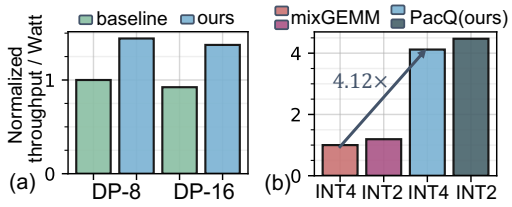


Fig. 12. (a) Comparison of different sizes of the DP unit. (b) Comparison of PacQ with Mix-GEMM [17]. Both (a) and (b) use the metric of throughput per watt. The workloads for both comparisons are *m16n16k16*.

In Figure. 12(a), we further study the effect of the size of the DP unit (DP-8 and DP-16). The results show that PacQ's performance improvements are orthogonal to the size of the DP units. In Figure. 12(b), we compare the performance of PacQ with prior work, Mix-GEMM [17], which also aims to improve the throughput of mixed-precision GEMM workloads through binary segmentation. With FP16 activation, PacQ achieves a 4.12× performance improvement for INT4 weights and a 3.75× improvement for INT2. This performance advantage is attributed to the fact that the binary segmentation technique performs poorly for hyper-asymmetric GEMM.

VI. Conclusion

In this work, we present PacQ, a novel SIMT microarchitecture designed to efficiently handle hyper-asymmetric GEMMs with mixed-precision weights (INT4 and INT2) and activations (FP16). By introducing a new packing and dataflow strategy, coupled with our parallel FP-INT multiplier, PacQ achieves significant improvements in hardware efficiency and throughput. Through extensive experimental evaluation, we demonstrate that PacQ delivers up to 54.3% reduction in register file accesses, a 1.99× speedup in throughput, and up to 81.4% reduction in energy-delay product (EDP) compared to conventional SIMT baselines. We believe our work can provide the community with insights into the design space exploration for hyper-asymmetric GEMM acceleration.

REFERENCES

[1] A. contributors, "Autogptq," https://github.com/AutoGPTQ, 2024.

[2] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[3] ——, "Optq: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2022.

[4] D. Gope, J. Beu, and M. Mattina, "High throughput matrix-matrix multiplication between asymmetric bit-width operands," *arXiv preprint arXiv:2008.00638*, 2020.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[6] J. Jang *et al.*, "Figna: Integer unit-based accelerator design for fp-int gemm preserving numerical accuracy," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 760–773.

[7] Y. J. Kim, R. Henry, R. Fahim, and H. H. Awadalla, "Who says elephants can't run: Bringing large scale moe models into cloud scale production," *arXiv preprint arXiv:2211.10017*, 2022.

[8] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," *arXiv preprint arXiv:2102.05426*, 2021.

[9] Y. Li and P. Panda, "Tesseraq: Ultra low-bit llm post-training quantization with block reconstruction," *arXiv preprint arXiv:2410.19103*, 2024.

[10] J. Lin *et al.*, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.

[11] llmc contributors, "llmc: Towards accurate and efficient llm compression," https://github.com/ModelTC/llmc, 2024.

[12] Z. Mo *et al.*, "Lut tensor core: Lookup table enables efficient low-bit llm inference acceleration," *arXiv preprint arXiv:2408.06003*, 2024.

[13] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, 2009.

[14] NVIDIA Corporation, "NVIDIA TESLA V100 GPU ARCHITECTURE," http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, June 2017.

[15] G. Park, B. Park, M. Kim, S. Lee, J. Kim, B. Kwon, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models," *arXiv preprint arXiv:2206.09557*, 2022.

[16] M. A. Raihan *et al.*, "Modeling deep learning accelerator enabled gpus," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 79–92.

[17] E. Reggiani *et al.*, "Mix-gemm: An efficient hw-sw architecture for mixed-precision quantized deep neural networks inference on edge devices," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1085–1098.

[18] G. Tan *et al.*, "Fast implementation of dgemm on fermi gpu," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–11.

[19] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[20] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "Bitnet: Scaling 1-bit transformers for large language models," *arXiv preprint arXiv:2310.11453*, 2023.

[21] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[22] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[23] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[24] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.

TAXI: Traveling Salesman Problem Accelerator with X-bar-based Ising Macros Powered by SOT-MRAMs and Hierarchical Clustering

Sangmin Yoo[1,2,4], Amod Holla[2], Sourav Sanyal[2], Dong Eun Kim[2],
Francesca Iacopi[1], Dwaipayan Biswas[3], James Myers[3], Kaushik Roy[2]
[1]*imec USA* & [2]*Purdue University*, West Lafayette, Indiana, USA, [3]*imec*, Leuven, Belgium,
[4]*Oregon State University*, Corvallis, Oregon, USA

***Abstract*—Ising solvers with hierarchical clustering have shown promise for large-scale Traveling Salesman Problems (TSPs), in terms of latency and energy. However, most of these methods still face unacceptable quality degradation as the problem size increases beyond a certain extent. Additionally, their hardware-agnostic adoptions limit their ability to fully exploit available hardware resources. In this work, we introduce TAXI – an in-memory computing-based TSP accelerator with crossbar(Xbar)-based Ising macros. Each macro independently solves a TSP sub-problem, obtained by hierarchical clustering, without the need for any off-macro data movement, leading to massive parallelism. Within the macro, Spin-Orbit-Torque (SOT) devices serve as compact energy-efficient random number generators enabling rapid "natural annealing". By leveraging hardware-algorithm co-design, TAXI offers improvements in solution quality, speed, and energy-efficiency on TSPs up to 85,900 cities (the largest TSPLIB instance). TAXI produces solutions that are only 22% and 20% longer than the Concorde solver's exact solution on 33,810 and 85,900 city TSPs, respectively. TAXI outperforms a current state-of-the-art clustering-based Ising solver, being 8× faster on average across 20 benchmark problems from TSPLib.**

***Index Terms*—Ising Machine, Accelerator, In-Memory Computing, Combinatorial Optimization Problem, SOT-MRAM.**

I. Introduction

Combinatorial optimization problems (CoPs), like the traveling salesman problem (TSP), are ubiquitous with applications in logistics, electronic design automation, genetics, and operations research [1], [2]. For TSPs, one of the most extensively studied CoPs, traditional methods [3] use heuristics like branch-and-bound and backtracking to find optimal solutions. However, such methods slow down exponentially as the problem size increases, due to exponential search space explosion. Additionally, traditional von Neumann architecture causes memory bottlenecks, rendering the TSP solvers inefficient for large instances due to repeated read-write operations during search-space exploration. This operating mismatch between the traditional TSP solver's execution pattern and von Neumann architecture's dataflow further slows down the optimization process, with the increase in problem size.

Ising models, inspired by statistical physics, offer a promising alternative for solving small-scale TSPs by encoding combinatorial state-spaces onto 2D arrays of binary spin values, that naturally settle into minimal energy states. Small

This work was funded through the joint MOU between the Indiana Economic Development Corporation, Purdue University, imec with in-kind support from the Applied Research Institute. Yoo participated in this work as a postdoctoral researcher at imec & Purdue University

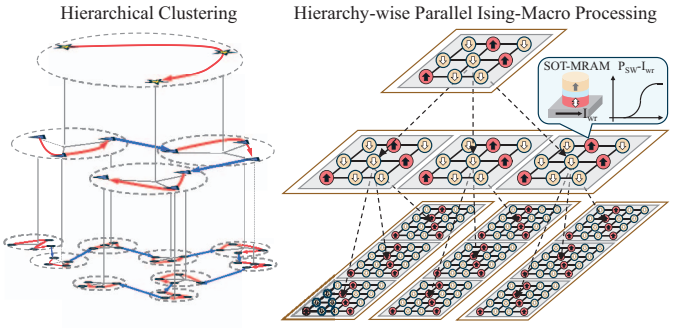


Fig. 1. Hierarchical clustering and Ising macros for large-scale traveling salesman problems. The shortest route within each cluster (red in the left) is optimized by an Ising macro (each grey box in the right). The final solution is derived by merging inter-cluster (blue in the left) and intra-cluster routes.

TSP instances can be efficiently mapped onto Ising crossbar (Xbar) arrays, potentially as hardware macros, enabling high-speed optimization. However, as the problem size grows, solution quality degrades due to the expanding search space and quadratic rise of spin interactions leading to a growing complexity of interconnects required to realize larger Ising Xbars. To address this issue, hierarchical clustering methods have been introduced [4]–[7], where large TSPs are decomposed into sub-problems that can be solved using smaller Ising models with acceptable quality. This work builds upon these clustering-based approaches, scaling further to tackle even larger problems with improved solution quality and further improvements in latency.

Typically, Ising solvers rely on stochastic processes to find near-optimal solutions. In conventional Complementary metal-oxide-semiconductor (CMOS)-based systems, random number generators (RNGs) are used to perform the stochastic switching of spins. However, these RNGs consume significant area and are too slow to meet the demands of rapid annealing, causing CMOS-based Ising solvers to be bulky with large area overheads and sluggish response-times [8], [9]. To overcome this, Spin-Orbit-Torque Magnetic Random Access Memory (SOT-MRAM)-based RNGs are considered in this work for Ising annealing, due to their energy-efficient and compact RNG implementation capability. In addition, in-memory computing (IMC) enables computations within memory arrays, reducing data movement and energy consumption. Xbar-based Ising primitives efficiently map TSP sub-problems onto hardware arrays [6], [7], making them suitable for realizing IMC macros. However, existing IMC-based Ising implementations still rely on off-macro memory accesses to store spin states.

To that effect, we propose TAXI – a TSP accelerator that uses Xbar-based Ising primitives as IMC macros with minimal off-macro data movement. TAXI applies hierarchical clustering to decompose large TSPs into smaller sub-problems, which are then solved in parallel using dedicated Ising hardware units. This architecture ensures that each sub-problem is solved directly in memory. This results in a hardware-algorithm co-design framework, where the algorithm (hierarchical clustering) and the architecture (spatially arranged Ising cores) are tightly integrated (See Fig 1). The custom hardware is designed by considering the algorithm's hierarchical execution pattern, ensuring that the Ising computations align with the algorithm's flow, enabling TAXI to maintain high solution quality as well as high speed even when problem size increases. Our contributions are as follows:

- A custom multiplication-and-accumulation (MAC)-based TSP solver using SOT-MRAMs and its stochastic switching to accelerate Ising annealing, overcoming the limitations of traditional CMOS-based RNGs (Section III).
- A novel hierarchical clustering with fixed inter-cluster routes that reduces TSP scale, with independently invoked Ising sub-solvers to enhance parallelization and sub-solution quality, enabling near-optimal solutions for large-scale TSPs (Section IV).
- A crossbar (Xbar)-based parallel architecture that supports in-macro Ising computing through hardware-algorithm co-design, mapping sub-problems onto Xbar arrays, and aligning the custom hardware with the algorithm to improve scalability (in terms of latency and energy) for large-scale TSPs (Section V).

Our experimental results (Section VI) demonstrate that TAXI is 8× faster on average across 20 benchmark problems (with up to 85,900 cities) from TSPLib [10] compared to a state-of-the-art hierarchical Ising solver [5]. TAXI generates 3% shorter traveling route on 33,810 cities over method [7] and 31% shorter route on 85,900 cities over method [5] – two recent clustering-based Ising works which demonstrate the best solutions on the respective problems, so far.

II. BACKGROUND AND RELATED WORKS

A. *Ising Solver*

In Ising model, Hamiltonian (energy) of the total system (H_{total}) and each spin (element) (H_i) are defined as below:

$$H_{total} = -\sum_i \sum_j J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i \tag{1}$$

$$H_i = \sum_j J_{ij}\sigma_j + h_i \tag{2}$$

where J_{ij} represents interaction between spins, σ_i and σ_j, while h_i denotes external field applied to spins. Equation 1 can be reformulated by equation 2 as below:

$$H_{total} = -\sum_i \left(\sum_j J_{ij}\sigma_j + h_i \right) \sigma_i \tag{3}$$

According to the rule that spin state, +1 or -1, follows the polarity of H_i, greater than 0 or not, respectively, ΔH_{total} is *always negative*. It means every spin update minimizes H_{total}, approaching an optimal solution in an unsupervised manner.

Inspired by the nature, Ising solvers have been proposed to naturally find the optimal solution to problems that can be properly encoded in Ising format. For Ising acceleration, MAC operation based on a memory Xbar has been deployed [7], [11] for $\sum_j J_{ij}\sigma_j$ in equation 2; Spin states (σ_j) are applied to rows and multiplied with J_{ij}. Then, H_i is derived.

However, the always-descending energy update is insufficient to get the optimal solution when problems are complicated. The complication makes the energy search space of Ising problem non-monotonic as depicted in (Fig 2), causing the Ising solver to get stuck in local minimas. To climb up the energy hill and reach global minima which is optimal solution, Ising solver needs the power to violate the always-descending nature. For the violation, stochastic switching of spin disregarding equation 3 is necessary.

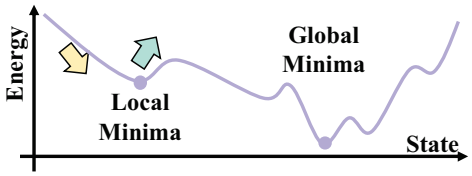


Fig. 2. Non-monotonic energy search space. Energy minimization (yellow) and Stochastic update (green) jointly find the global minima by enabling descending the energy landscape and escaping from local minimas, respectively.

B. *Source of Stochasticity*

CMOS-based random number generator (RNG) [7], [12], [13] and emerging electronic devices, such as resistive random access memory (RRAM) [11], silicon-oxide-nitride-silicon (SONS) FinFET [6], and spin-transfer torque MRAM [14]–[18] have been deployed as the source of stochasticity.

RRAM and SONS FinFET offer stochasticity through their inherent thermal and temporal noise when used for a Xbar array for MAC operation [6], [11]. Despite their better chip density, the intrinsic noise becomes uncontrollable when the Xbar grows [11]. Magnetic tunnel junction (MTJ)-based RNGs based on low barrier nanomagnets and telegraphic switching [15]–[18] and CMOS-based RNGs [8], [9] can be placed in the peripheral circuitry, which are not scaled up by Xbar size. However, current MTJ-based implementations require very low barrier nanomagnets ($\Delta \sim 0kT$) for truly high-speed RNG (>1Gb/s) and high-frequency sense circuitry to detect rapid switching of the device, while CMOS-based RNGs are slower (<2400Mb/s) [9] and take much more area ($> 375\mu m^2$) [8].

C. *Ising Solvers for Large-scale TSP*

Large-scale TSPs are hard for Ising solvers due to the quadratically growing number of connections as TSP size increases. Hierarchical Vertex Clustering (HVC) [4] proposed hierarchical clustering to reduce the number of connections. Despite the reduction, it still sparsely maps all the clustered problems to a single matrix, which leaves the connection problem unsolved when the problem size is beyond a certain

level. Neuro-Ising [5] proposed a hybrid clustering-and-graph-neural network manner but shows performance degradation as the problem size increases. In-memory annealer (IMA) [6] and its follow-up, digital compute-in-memory annealer (CIMA) [7], based on the hierarchical clustering achieves acceleration by simultaneously optimizing non-adjacent clusters. However, they store the spin states outside macros, causing slow-down.

III. SOT-MRAM-BASED TSP SOLVER

A. *MAC-based Energy Minimization for TSP*

TAXI leverages the Xbar-based MAC operation to minimize H_{total}. To achieve the objective of TSP, finding the shortest route to visit all cities only once, information on the distance between cities and the visiting order initialized by input order are used. The distance is reformulated as follows:

$$W_D(A,B) = \frac{D_{min}}{D_{A-B}} \times B_{Precision} \tag{4}$$

where $W_D(A,B)$ denotes a conductance value to map on a cross-point of Xbar, while D_{A-B} and D_{min} represent the distance between cities A and B and the minimum distance among all paths, respectively, and $B_{Precision}$ is the bit-precision that can be offered by the hardware.

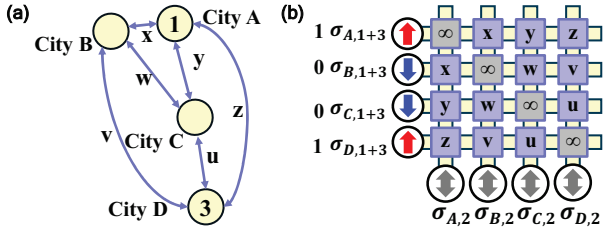


Fig. 3. (a) TSP with 4 cities. $u-x$ represent distances and numbers in a circle denote visiting orders. (b) Distance matrix mapped on a crossbar. Distances reformulated to W_D are programmed to crosspoints in resistance.

Fig 3 displays how the W_D is mapped on the Xbar and used to choose a city to visit at order *i* which results in the shortest route. For the optimization, cities visited at the previous (*i*-1) and the next (*i*+1) orders are considered because the total traveling distance is decided by both. The visiting information ($\sigma_{A,i}$) to a specific city (*A*) at a specific order (*i*) is represented by a binary spin, 1 or 0, following quadratic unconstrained binary optimization (QUBO) and Ising equivalence [20]. $\sigma_{A,1}$ and $\sigma_{D,3}$ are 1 because they are visited at order 1 and 3, respectively. With the fact that the distance information is static, we superpose the vector representing visiting information of order *i*-1 ($\sigma_{A-D,1}$) and *i*+1 ($\sigma_{A-D,3}$), as shown in Fig 3b. It reduces the crossbar array sizes, leading to better area-efficiency and less involvement of non-ideality.

In this setup, the current flowing through each column ($\sigma_{A-D,2}$) following Ohm's law and Kirchhoff's current law represents relative distance when the city is chosen for the order. It can be expressed as:

$$D_{x,i} = \sum_{k \neq x} W_D(x,k)(\sigma_{k,i-1} + \sigma_{k,i+1}) \tag{5}$$

According to equations 4 and 5, the current flowing through the 3rd column becomes the largest due to shorter distance (lower resistance), making $\sigma_{C,2}$ selected as up-spin.

B. *Stochasticity-Involved Decision Making*

In TAXI, stochasticity is involved in the choice of the largest current via a stochastically produced binary vector as big as the problem size. Only cities corresponding to 1s in the vector are considered for the choice. It offers the power to violate the always-descending nature, mentioned in Section II-A. The visiting order is updated by the newly chosen city.

C. *Ising Macro as a TSP Solver*

A Xbar array with customized peripheral circuitry is leveraged as an Ising macro to solve a TSP in the manner of the aforementioned algorithms. Fig 4 shows the system-level layout of the Xbar-based Ising macro along with the associated peripherals inset in Fig 4c,d. The unit cell of the Xbar array is a 3T-1M SOT-MRAM unit, with a transmission gate to control the read current, and 1 transistor to control the write current. The transmission gate was chosen owing to the need for accurate bi-directional reads.

The Xbar is divided into $B+1$ partitions, where B is the chosen bit-precision for the W_D. The first B partitions contain the W_D, while the last partition stores the spin information ($\sigma_{k,i}$) as the solution of the given TSP. To minimize effect from non-ideality, such as wire resistance, higher significant bit is stored closer to the left end. The rows and columns of the spin storage (SS) represent cities and visiting orders. If a specific city (A) is chosen at a specific order (1), a SOT-MRAM that is located in the first row and the first column in the SS is programmed in the low resistance state (LRS), while the others on the same column are in the high resistance state (HRS). This implies that the size of the Xbar required for a TSP problem size of N is $N \times N \cdot (B+1)$.

1) ***Superpose****:* The superposition of vectors representing visiting information, discussed in Section III-A and Fig 3, is executed by MAC operation using the SS, as illustrated in Fig 4a. In the case of Fig 3, the first and the third columns are activated to recall visiting information at order 1 and 3. Then, the Xbar natively derives the superposed vector in current at the end of the rows, while a following current comparator [21] translates it in binary and a D-latch stores the binary vector. This flow is depicted by the red arrow in Fig 4.

2) ***Calculate Distance****:* The stored vector in the D-Latch is fed back to the rows, generating currents by executing equation 5, based on the W_D stored in the first B partitions of the Xbar. Only columns in the first B partitions are activated, with columns in the SS deactivated. The currents from the Xbar flow into current mirrors, scaled by 2^{b-1} depending on the bit-significance (b) of the partition, as shown in Fig. 4b.

3) ***Stochastic Binary Vector*** *:* The stochastic binary vector mentioned in Section III-B is generated by SOT-MRAM's stochastic switching-based circuit consisting of N identical units shown in Fig 4c. SOT-MRAM has two ferromagnetic layers (one with a fixed magnetic moment direction, while another is free to rotate) separated by a thin insulator. Its resistance is determined by how many electrons can tunnel through this MTJ, which depends on whether the magnetic moment of the

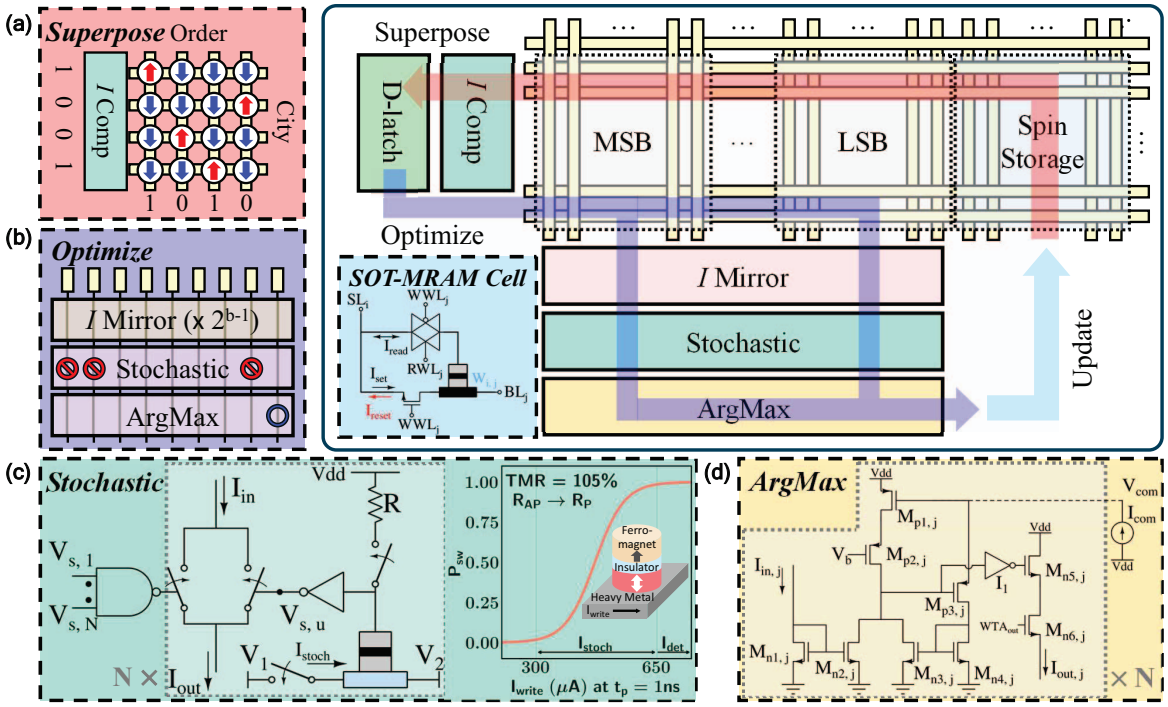


Fig. 4. Floor plan of an Ising macro. Arrows represent data flow across the macro. Insets (a-d) illustrate components in the macro. (a) Superposition of vectors to optimization. Red and blue arrows represent up- and down-spin stored in spin storage, respectively. (b) Optimization by crossbar array as the distance matrix and following peripheral circuits. (c) SOT-MRAMs in the stochastic circuit pass or prevent current from the crossbar following its stochastic switching. Inset displays the probability of switching R_{AP} to R_P of the chosen SOT device [19]. I_{stoch} and I_{det} denote the current ranges for stochastic and deterministic operations, respectively. (d) ArgMax circuit picks a city to visit by choosing the largest current.

two ferromagnetic layers is aligned in parallel (R_P) or anti-parallel (R_{AP}). This alignment can change due to the Spin Hall effect induced by current flowing through a heavy metal layer under the ferromagnet [22]. The stochastic switching that TAXI leverages is based on this dynamic. Depending on the amount of current, the probability of switching (P_{sw}) between R_P and R_{AP} can be controlled, as shown in Fig 4c [19]. By controlling the current that derives a specific P_{sw} (300μA — 650μA), TAXI can control the expected number of 1s in the stochastic binary vector. SOT-MRAMs in the Xbar are operated only in the deterministic regime (>650μA).

The stochastic circuit (Fig. 4c) controls the flow of the preceding current mirror outputs into the next stage upon the stochastic switching. For a given unit, if the SOT device switches, the voltage across the divider circuit becomes lower than units whose SOT device has not switched. The output of the inverter is therefore high, resulting in the unit allowing the input current to pass through. The circuit allows the current only through the columns ($\mathcal{S}$) where the SOT device has stochastically switched for a given iteration to pass through. In case $\mathcal{S} = \varnothing$, currents through all columns are allowed to pass because the NAND gate output is high for every unit.

*4) **Choosing the Largest Current**:* The largest current is selected by the ArgMax circuit shown in Fig 4d. It processes a vector of input currents and outputs a vector where only the index 'i' corresponding to the largest input current remains nonzero. The winner current is set to the minimum required to switch a SOT-MRAM device in the deterministic regime. The ArgMax circuit is based on Lazzaro's winner-take-all circuit [23], enhanced with a cascaded transistor ($M_{p2,j}$) in the input branch to increase output resistance [24], and a current mirror ($M_{n2,j}$, $M_{n3,j}$) to boost feedback, significantly improving resolution and speed [25].

*5) **Update Spin Storage**:* The one-hot encoded current vector derived by the ArgMax circuit is directly used to update the spin storage, as illustrated by the cyan arrow in Fig 4. Before the update operation, all the devices on the column representing the order which is being optimized are reset to HRS which represents 0. Then, the output current vector of the ArgMax circuit is applied to write the column.

*6) **Annealing**:* The processes described in sections III-C1 - III-C5 are repeated for the pre-designated number of iterations from the first to the last order by activating different columns in the SS depending on the visiting order to optimize. After the whole iteration, the spin information stored in the SS is retrieved as the final solution of the given TSP. To optimize the solution, stochasticity should decrease during iterations, allowing the Ising system to remain at the global minimum as the solution improves. Otherwise, the system may escape the global minimum and end up with a worse solution. The solution is optimized the best when the stochasticity decreases slowly. Optimization is however faster in the early stages making the non-linear decrease of stochasticity appealing.

For the non-linearity, TAXI utilizes the native sigmoidal switching probability vs I_{write} characteristic, as shown in Fig 4c. The non-linearity can enable a relatively rapid decrease in stochasticity at the earlier stage, leading to shorter overall latency, while keeping the advantage of slow decrease at

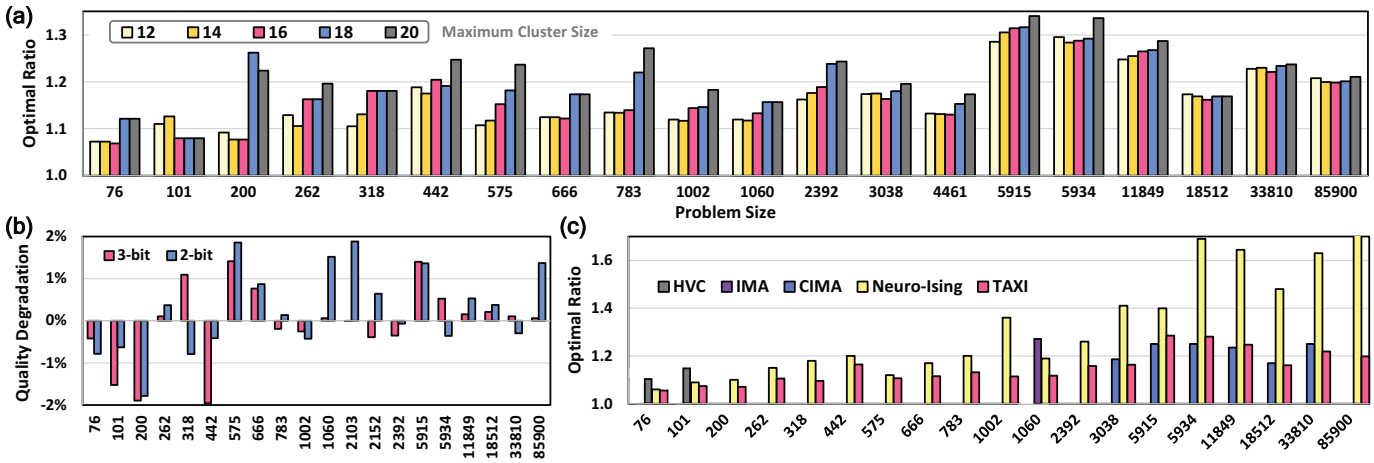


Fig. 5. (a) Optimal ratio depending on the maximum cluster size in 4-bit precision. (b) Solution quality degradation when the bit precision changes from 4-bit to lower-bit options. The positive number represents degradation. (c) Comparison of solution optimality. Data of TAXI with 4-bit precision and 12 cluster size are presented. Data of other Ising solvers are adapted from [5]–[7].

the later stage. To leverage the native sigmoidal decrease, the I_{write} is initialized to 420μA (corresponding switching probability, 20%) and linearly decreased by 50nA after every iteration. Once the I_{write} becomes 353μA (corresponding probability, 1%), the Ising solver stops, then, the solution stored in the SS is taken as the final output of the given TSP.

IV. HIERARCHICAL CLUSTERING

*1) **Locally Solve Global Problems**:* We adopt a hierarchical structure similar to HVC [4] for clustering, executed in a bottom-up fashion. After clustering all cities at level i, the cluster centroids form level $i+1$. This process repeats until the number of centroids at a level is less than the maximum TSP size confidently solvable by an Ising macro. The number of clusters at each level is determined by the maximum size.

*2) **Parallel Processing**:* After the hierarchical structure is defined, the TSP is solved in a top-down manner. Once the topmost TSP (level L) is solved, the solution, which is the visiting order of clusters at level $L-1$, is determined. Unlike HVC, which sparsely maps all clusters to a single Xbar array to optimize routes both within and between clusters, TAXI fixes the first and last cities of sub-problems first to minimize the inter-cluster route. This fixation ensures that sub-problem solutions do not compromise the inter-cluster distance. The first and last cities of each cluster are determined by identifying the closest city pairs between neighboring clusters, enabling parallel solving of as many clusters in a level as there are Ising macros in a chip. The entire process concludes when all sub-problems in the lowest level are solved.

*3) **Agglomerative Clustering**:* TAXI leverages Agglomerative clustering to achieve the hierarchical clustering for its robustness to outliers, instead of K-means clusters adopted by others [4]–[7]. It is a type of hierarchical clustering that merges clusters based on an inter-cluster similarity metric until the desired number of clusters is obtained [26]. With the Ward linkage, the squared distance or variance within a cluster is minimized [27]. Even though K-means clustering also aims to minimize the intra-cluster variance, they usually lead to nearly spherical or regular clusters [28], while agglomerative clustering can produce compact irregular clusters.

V. X-BAR-BASED PARALLEL ARCHITECTURE

Hierarchical clustered Ising macros are mapped to the Xbar architecture to enable in-macro Ising computing through hardware-algorithm co-design, mapping sub-problems onto Xbar arrays, and aligning the custom hardware with the algorithm to improve scalability for large-scale TSPs. We instrumented the PUMA architecture [29] to perform the mapping and evaluated the latency and energy consumption from data movement between off-chip memory and the proposed Ising macros executing parallel Ising computations. PUMA is a cycle-accurate simulator based on an IMC spatial architecture, structured with a hierarchy of chip, tile, core, and MVMUs. The simulator's compiler generates instructions using PUMA ISA, and the simulator executes them to assess latency and energy. We replaced PUMA's ReRAM-based MVMUs with our Ising macro and scaled all elements from PUMA's 32nm to a 65nm node. See Section VI-C for more details.

VI. EVALUATION

A. *Results on Multiple TSPs*

The performance of TAXI with varied maximum cluster sizes is simulated in C++, following the Ising layer presented in [5]. For realistic simulation, ON/OFF resistance of SOT-MRAM and transistors and wire resistance are considered [19]. Fig 5a displays the optimal ratio which is derived by dividing a solution of Ising solver with exact solution [30]. A smaller cluster (sub-problem) size leads to a better solution quality in most cases as Ising solvers usually show better results on smaller problems. With the problem size of 12, we tested different bit precisions for the W_D.

Fig 5b presents solution quality degradation when the bit precision changes from 4-bit to 3- or 2-bit. It shows that the performance is maintained in the 2% range. We attribute this fluctuation to variation of both W_D precision and non-ideality arising from Xbar size change. From the energy-efficiency

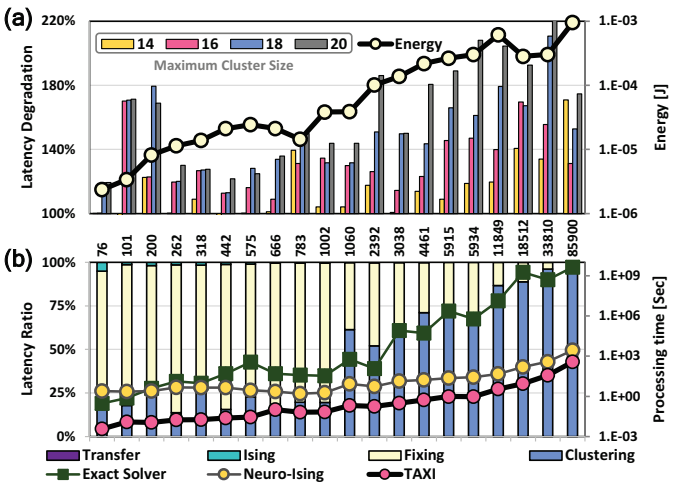


Fig. 6. (a) Latency (bar) and energy (line) arising from Ising macros and data transfer within the architecture depending on the maximum cluster size in 4-bit precision. Energy based on a problem size of 12 and 2-bit precision is representatively displayed. (b) Total Latency including clustering, fixing, Ising processing, and data transfer illustrated in lines (right y-axis). Bars represent the ratio of each component contributing to the total latency (left y-axis).

TABLE I
CIRCUIT SIMULATION RESULTS FOR COMPLETION OF 1 ITERATION INCLUDING SUPERPOSITION, OPTIMIZATION, AND SPIN-STORAGE UPDATE.

		2 bit	3 bit	4 bit
Array Size		12×36	12×48	12×60
Power [mW]		4.202	5.033	5.11
Latency [ns]	**Superposition**	3	3	3
	Optimization	4	4	4
	Storage Update	2	2	2
Energy [pJ]		37.82	45.3	45.98

perspective, the lower bit-precision is beneficial. Depending on the priority between the power budget and the solution quality, TAXI can be reconfigured. The comparison will be discussed in the following section VI-C. TAXI with a clustering size of 12 and 4-bit precision is benchmarked with existing Ising solvers for TSPs in Fig 5c. TAXI outperforms the others [5]–[7] in most cases including three of the largest TSPs, 18512, 33810, and 85900.

B. Circuit Simulation

An entire Ising macro shown in Fig 4 is simulated using the `Cadence® Spectre®` simulator in the TSMC 65nm technology node. Verilog-A behavioral modeling was performed to fit the characteristics of the considered SOT device in section III-C3. We choose a TSP problem size of 12. We run the simulation for 1 complete iteration including the superposition, optimization, and spin-storage updates. The pre-layout circuit simulation results are presented in Table I. The higher-bit precision causes a larger area and higher energy consumption.

C. Architecture Simulation

The total latency excluding clustering with fixing simulated by PUMA simulator is presented in Fig 6a. It presents the ratio of the latency with a specific maximum cluster size to that with a maximum cluster size of 12. The higher value means the slower operation. In most cases, the larger cluster size causes longer latency. The clustering time does not vary noticeably depending on the problem size. The latency arising from fixing the intercluster routes theoretically increases as the problem size increases, because the possible number of pairs between clusters increases quadratically while the number of clusters linearly decreases. Based on the results presented in Fig 5a and Fig 6a, the smaller problem size is preferred if the chip can contain as many Ising macros as the number of clusters in a level. It enables mapping and solving all clusters in parallel. Otherwise, the larger size could be a better option to maximize the parallel operation. The total energy consumption excluding clustering and fixing is presented in line in Fig 6a. Lower energy consumption for transferring, mapping, and processing (presented in Table I) smaller W_D makes the lower bit-precision more efficient. Table II shows the energy consumption, excluding data transfer and mapping, to fairly compare TAXI and previous works, demonstrating that ours is the most power-efficient.

TABLE II
ENERGY COMPARISON WITH STATE-OF-THE-ART

	[4]	[6]	[7]	**This Work**
Technology	CPU	14nm FinFET	16/14nm CMOS	65nm CMOS + SOT-MRAM
Problem Size	101	1060	33K, 86K	1060, 33K, 86K
Energy (J)	1.1	20.08μ	$\sim 20\mu$, $\sim 45\mu$	1.81μ, 2.67μ, $3.07\mu^{\dagger}$

† Excludes mapping energy. Including mapping, energies are $38.7\mu J$, $302\mu J$, and $952\mu J$ for TSPs with sizes 1060, 33K and 86K respectively.

Fig 6b shows the overall latency of TAXI with a maximum cluster size of 12 along with that of Neuro-Ising [5] and an exact solver [3]. We exclude CIMA [7] from this comparison as they do not report the total latency including clustering. The bar plots display the ratio of each component contributing to the total latency. As problem size increases, the speed advantage over the exact solver gets bigger and the total latency is dominated by clustering and fixing. The biggest problem, pla85900, is solved by TAXI in 375.4s and consumes 9.51×10^{-4}J, while it takes a projected time of 136 years (1.14×10^{7} longer) and 3.82×10^{11}J (4.01×10^{14} more) for the exact solver on a single core CPU [31].

VII. CONCLUSION

We discussed existing hierarchical clustering-based Ising solvers suffer performance degradation with increasing problem size, due to search-space explosion and inherent mismatch between hierarchical execution pattern and hardware architecture data flow. To that effect, we proposed TAXI – a hardware-algorithm co-designed TSP accelerator with Xbar-based Ising macros – which outperforms existing Ising solvers on very large-scale TSPs. We report $8\times$ latency improvement across 20 benchmarks over a clustering-based Ising solver [5] from TSPLib [10], up to the largest TSP with 85,900 cities. TAXI is endowed with SOT-MRAM-based in-memory Ising primitives and a hierarchical clustering algorithm. The co-design enables TAXI to solve sub-problems of large-scale TSPs on a cluster-by-cluster basis without off-macro memory access, resulting in high inter-cluster parallelism with graceful acceleration, while maintaining solution quality within an acceptable $1.2\times$ that of exact solver [3] on a TSP with 85,900 cities.

REFERENCES

[1] MH BJORNDAL, A CAPRARA, PI COWLING, F DELLACROCE, H LOURENCO, F MALUCELLI, AJ ORMAN, D PISINGER, C REGO, and JJ SALAZAR. Some thoughts on combinatorial optimization. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH*, 83(2):253–270, JUN 8 1995.

[2] Jann Michael Weinand, Kenneth Sörensen, Pablo San Segundo, Max Kleinebrahm, and Russell McKenna. Research trends in combinatorial optimization. *International Transactions in Operational Research*, 29(2):667–705, 2022.

[3] David Applegate and William Cook. Concorde tsp solver, 2006.

[4] Akira Dan, Riu Shimizu, Takeshi Nishikawa, Song Bian, and Takashi Sato. Clustering approach for solving traveling salesman problems via ising model based solver. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[5] Sourav Sanyal and Kaushik Roy. Neuro-ising: Accelerating large-scale traveling salesman problems via graph neural network guided localized ising solvers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(12):5408–5420, 2022.

[6] Anni Lu, Jae Hur, Yuan-Chun Luo, Hai Li, Dmitri E. Nikonov, Ian A. Young, Yang-Kyu Choi, and Shimeng Yu. Scalable in-memory clustered annealer with temporal noise of charge trap transistor for large scale travelling salesman problems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):422–435, 2023.

[7] Anni Lu, Junmo Lee, Yuan-Chun Luo, Hai Li, Ian Young, and Shimeng Yu. Digital cim with noisy sram bit: A compact clustered annealer for large-scale combinatorial optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery.

[8] Kaiyuan Yang, David Fick, Michael B. Henry, Yoonmyung Lee, David Blaauw, and Dennis Sylvester. 16.3 a 23mb/s 23pj/b fully synthesized true-random-number generator in 28nm and 65nm cmos. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 280–281, 2014.

[9] Sanu K. Mathew, Suresh Srinivasan, Mark A. Anders, Himanshu Kaul, Steven K. Hsu, Farhana Sheikh, Amit Agarwal, Sudhir Satpathy, and Ram K. Krishnamurthy. 2.4 gbps, 7 mw all-digital pvt-variation tolerant true random number generator for 45 nm cmos high-performance microprocessors. *IEEE Journal of Solid-State Circuits*, 47(11):2807–2821, 2012.

[10] Gerhard Reinelt. TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[11] Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh, Xia Sheng, Rui Liu, Can Li, Zhan Liu, Martin Foltin, Shimeng Yu, Qiangfei Xia, J Joshua Yang, Raymond Beausoleil, Wei D Lu, and John Paul Strachan. Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks. *Nature Electronics*, 3:409–418, 2020.

[12] Masanao Yamaoka, Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. A 20k-spin ising chip to solve combinatorial optimization problems with cmos annealing. *IEEE Journal of Solid-State Circuits*, 51(1):303–309, 2016.

[13] Chieh-Pu Lo, Sirish Oruganti, Yipeng Wang, Mengtian Yang, Shanshan Xie, and Jaydeep P. Kulkarni. Data movement-aware, ping-pong ising machine supporting full connectivity and variable bitwidths. *2024 IEEE European Solid-State Electronics Research Conference (ESSERC)*, pages 721–724, 2024.

[14] Saima Sharmin, Yong Shim, and Kaushik Roy. Magnetoelectric oxide based stochastic spin device towards solving combinatorial optimization problems. *Scientific Reports*, 7, 12 2017.

[15] Akio Fukushima, Takayuki Seki, Kay Yakushiji, Hitoshi Kubota, Hiroshi Imamura, Shinji Yuasa, and Koji Ando. Spin dice: A scalable truly random number generator based on spintronics. *Applied Physics Express*, 7(8):083001, jul 2014.

[16] Won Ho Choi, Yang Lv, Jongyeon Kim, Abhishek Deshpande, Gyuseong Kang, Jian-Ping Wang, and Chris H. Kim. A magnetic tunnel junction based true random number generator with conditional perturb and real-time output probability tracking. In *2014 IEEE International Electron Devices Meeting*, pages 12.5.1–12.5.4, 2014.

[17] Hochul Lee, Farbod Ebrahimi, Pedram Khalili Amiri, and Kang L. Wang. Design of high-throughput and low-power true random number generator utilizing perpendicularly magnetized voltage-controlled magnetic tunnel junction. *AIP Advances*, 7(5):055934, 03 2017.

[18] Kerem Yunus Camsari, Sayeef Salahuddin, and Supriyo Datta. Implementing p-bits with embedded mtj. *IEEE Electron Device Letters*, 38(12):1767–1770, 2017.

[19] K. Cai, G. Talmelli, K. Fan, S. Van Beek, V. Kateel, M. Gupta, M.G. Monteiro, M. Ben Chroud, G. Jayakumar, A. Trovato, S. Rao, G.S. Kar, and S. Couet. First demonstration of field-free perpendicular sot-mram for ultrafast and high-density embedded memories. In *2022 International Electron Devices Meeting (IEDM)*, pages 36.2.1–36.2.4, 2022.

[20] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem, 2008.

[21] H. Träff. Novel approach to high speed cmos current comparators. *Electronics Letters*, 28:310–311, 1992.

[22] Luqiao Liu, O. J. Lee, T. J. Gudmundsen, D. C. Ralph, and R. A. Buhrman. Current-induced switching of perpendicularly magnetized magnetic layers using spin torque from the spin hall effect. *Phys. Rev. Lett.*, 109:096602, Aug 2012.

[23] J. Lazzaro, S. Ryckebusch, M.A. Mahowald, and C. A. Mead. Winner-take-all networks of o(n) complexity. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

[24] B. Sekerkiran and U. Cilingiroglu. Improving the resolution of lazzaro winner-take-all circuit. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 2, pages 1005–1008 vol.2, 1997.

[25] J.A. Starzyk and X. Fang. Cmos current mode winner-take-all circuit with both excitatory and inhibitory feedback. *Electronics Letters*, 29:908–910, 1993.

[26] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[27] Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[28] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

[29] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 715–731, New York, NY, USA, 2019. Association for Computing Machinery.

[30] Concorde benchmarks (from 1999). https://www.math.uwaterloo.ca/tsp/concorde/benchmarks/bench99.html.

[31] Concorde benchmarks on pla85900. https://www.math.uwaterloo.ca/tsp/pla85900/compute/cpu.htm.

An Enhanced Data Packing Method for General Matrix Multiplication in Brakerski/Fan-Vercauteren Scheme

Xiangchen Meng, Yan Tan, Zijun Jiang and Yangdi Lyu*
Microelectronics Thrust, The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, Guangdong, China
Emails: {xmeng027, ytan910, zjiang438}@connect.hkust-gz.edu.cn, yangdilyu@hkust-gz.edu.cn

***Abstract*—General Matrix-Matrix Multiplication (GEMM) stands as the most ubiquitous operation in machine learning applications. However, performing GEMM within Fully Homomorphic Encryption (FHE) is inefficient due to high computational demands and significant data migration constrained by limited bandwidth. Additionally, the inherent limitations of FHE schemes restrict the widespread application of machine learning, as standard activation functions are incompatible. This incompatibility necessitates alternative nonlinear functions, which lead to notable accuracy reductions. To address these challenges, we introduce a polynomial encoding methodology for GEMM under the Brakerski/Fan-Vercauteren (BFV) scheme and extend the method to inference with packing inputs and weights for different sizes. Furthermore, we design specialized hardware to accelerate the inference process through optimized scheduling between the hardware and the host system. In experiments, we implemented our hardware on an FPGA U250 platform. Compared to existing solutions, our method achieves superior performance, achieving the highest $4.22\times$ and $3.99\times$ speedups on MNIST and CIFAR-10.**

***Index Terms*—Fully Homomorphic Encryption, FPGA, Matrix Multiplication**

I. Introduction

The intensive tasks based on cloud platforms offer unparalleled scalability and computational prowess, rendering them ideal for training complex models and processing vast datasets. Yet in sensitive domains like healthcare, finance, and personalized services, data privacy concerns significantly impede the adoption of cloud-based solutions. To address these challenges, various privacy-preserving methodologies have been proposed for machine learning tasks, including differential privacy [1], secure multi-party computation [2], federated learning [3], and trusted execution environments [4]. Among these techniques, FHE has garnered significant attention from both academia and industry due to its unique capability to perform computations directly on encrypted data without the need for decryption. This property makes FHE particularly suitable for privacy-preserving machine learning applications, as it allows for secure model inference on sensitive data.

However, current implementations of CNN models under FHE face significant bottlenecks that hinder practical adoption:

Accuracy Degradation: Inference based on FHE limited by fundamental constraints, which couldn't use non-linear activation functions, such as *ReLU*, *Sigmod* et al. To maintain homomorphic encryption compatibility, existing solutions employ the *square* as alternative activation functions that introduce a degradation in model accuracy [5].

Homomorphic Latency: Homomorphic computation involves numerous complex polynomial calculations, resulting in operations that run 4 to 5 orders of magnitude slower than unencrypted counterparts. This substantial performance gap severely limits the feasibility of real-time machine learning applications [6], [7].

*Corresponding Author: yangdilyu@hkust-gz.edu.cn

Data Proliferation: The encryption process transforms data into high-degree polynomials (ranging from 2^{10} to 2^{16}) with each coefficient reaching hundreds of bits [8], [9]. This transformation causes substantial data expansion, leading to significant storage and bandwidth overhead.

To address these challenges, extensive research efforts have been devoted to mitigating the computational bottlenecks of FHE in machine learning applications. Initial approaches focused on CPU-based implementations with limited parallelism capabilities in handling the complexity of FHE operations. Seminal work by CryptoNets [10] established the foundational framework for privacy-preserving inference using homomorphic encryption, but the substantial computational overhead, requiring approximately 205 seconds for inference completion. Subsequently, Lola [5] introduced an innovative approach that categorized data types based on layer-wise intermediate results, implementing specialized operations for each category. This optimization yielded significant performance improvements, reducing inference latency to 2.2 seconds on the MNIST dataset relative to CryptoNets. However, the speed is still not fast enough to meet the requirements of real-time applications. Meanwhile, GPU-based implementations have been explored at the same time. Although optimized for tensor operations, which could achieve higher parallel, it is not optimized for the modular arithmetic intrinsic to homomorphic encryption. For instance, TensorFHE [11], despite leveraging GPU acceleration for homomorphic operations in convolutional neural networks, achieves a large improvement over CPU. But homomorphic multiplication operation (*HMULT*) still exhibits a substantial latency of 851 milliseconds.

To overcome the inherent hardware limitations, the research community has increasingly explored FPGA-based architectures for accelerating homomorphic encryption computations [12]–[14]. FPGAs offer distinct advantages through their reconfigurable hardware architecture, facilitating customized optimizations for modular arithmetic operations while maintaining high parallelism and computational efficiency. Cheetah [15], which introduced specialized optimizations for FPGA-based inference. Through weight duplication, Cheetah minimized rotation operations in dot-product computations, thereby enhancing computational efficiency and reducing noise accumulation during processing. However, this approach encounters scalability challenges with larger networks due to increased memory bandwidth requirements and data transfer overhead. FxHENN [16] advanced the field by implementing detailed hardware optimizations for inference. It utilized design space exploration layer by layer to maximize memory usage and computational efficiency. However, FxHENN processes each pixel individually during convolution operations, which increases memory consumption and computational complexity because it fails to leverage data reuse and parallel processing opportunities, resulting in higher system overhead.

Therefore, we propose an enhanced data-packing method, to speed up matrix multiplication and relieve bandwidth pressure. Our key contributions are as follows:

1) We introduce an enhanced data-packaging methodology for general matrix multiplication under FHE, reducing both memory footprint and computational complexity.
2) We extend the method to convolutional layers and fully connected layers and dynamically plan the packing method based on the model parameters.
3) We design hardware based on FPGA and plan the pipeline to set up scheduling between the host and FPGA
4) The experiment on Xilinx FPGA platforms for two CNN models demonstrates the significant outperforms compared with existing solutions, achieving $4.22\times$ and $3.99\times$ speed up with higher accuracy.

II. Background

A. Fully Homomorphic Encryption

FHE is a cryptographic technique that allows computations to be performed directly on encrypted data without requiring decryption. One widely used FHE scheme is the BFV scheme [17], [18]. In the BFV scheme, a plaintext message m is represented as a polynomial in the ring $R_t = \mathbb{Z}_t[x]/(x^n+1)$, where n denotes the polynomial degree and t is a plaintext modulus. This polynomial m can be expressed as:

$$m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{n-1} x^{n-1}$$

where $m_i \in \mathbb{Z}_t$ are the coefficients of the plaintext polynomial. The commonly used data encoding method of BFV is batch encoding, which allows multiple plaintext messages to be encoded into a single polynomial, thereby enabling parallel homomorphic computations on these elements.

The encryption process involves several key steps. First, the plaintext polynomial m is scaled by a large integer Δ to ensure noise tolerance. Then, two random polynomials a and e are sampled from a discrete Gaussian distribution. This ciphertext $ct = (c_0, c_1$ is generated as :

$$c_0 = -a \cdot s + e_1 + \Delta \cdot m \pmod q$$

$$c_1 = -a$$

where s is the secret key polynomial, a and e are randomly chosen polynomials from the ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$. The parameter q is a large modulus that determines the noise capacity of the scheme. The BFV scheme supports two primary homomorphic operations enabling computations on encrypted data. These operations are defined as follows:

Homomorphic Addition: Given two ciphertexts ct_1 and ct_2 corresponding to plaintexts m_1 and m_2, homomorphic addition is defined as:

$$Dec(ct_1 + ct_2) = m_1 + m_2$$

Homomorphic Multiplication: Similarly, for homomorphic multiplication of ciphertexts ct_1 and ct_2, the decrypted product yields:

$$Dec(ct_1 \cdot ct_2) = m_1 \cdot m_2$$

III. Matrix Multiplication based on polynomial packing

A. Motivation

GEMM constitutes a pivotal operation in a myriad of machine learning applications, contributing to approximately 70% of the total computational workload [19]. In the realm of FHE, data is encapsulated within polynomial structures to facilitate secure computation. Traditional batching techniques for matrix multiplication necessitate numerous rotational and multiplicative operations, leading to significant latency and the exacerbation of noise accumulation. Moreover, to uphold ciphertext security, polynomials must sustain a high degree, thereby imposing substantial spatial overhead.

To mitigate these challenges, we introduce an optimized polynomial-based data-packing methodology tailored for GEMM. This strategy is engineered to maximize the utilization of polynomial space, thereby reducing computational latency and minimizing noise accumulation while maintaining robust security guarantees.

B. Inner Product of Two Vectors

In homomorphic encryption, matrix multiplication efficiency can be significantly enhanced through strategic encoding of matrix entries as polynomial coefficients. Consider two one-dimensional matrices with k elements (where $2k-2 < n$):

$$A = [a_0, a_1, \cdots, a_{k-1}], B = [b_0, b_1, \cdots, b_{k-1}]^T$$

The result C of their matrix multiplication A · B can be computed as:

$$C = A \cdot B = \sum_{i=0}^{k-1} a_i b_i$$

To implement this polynomial packing method, we encode each element a_i and b_i as coefficients of polynomials. For matrix A, the elements are packed into the polynomial:

$$P_A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

For matrix B, the elements are arranged in reverse order within the polynomial:

$$P_B(x) = b_{k-1} + b_{k-2} x + b_{k-3} x^2 + \cdots + b_0 x^{k-1}$$

And after polynomial multiplication, the result $P_C(x)$ can be expressed as:

$$\begin{aligned} P_C(x) = {} & a_0 b_{k-1} + (a_0 b_{k-2} + a_1 b_{k-1}) x \\ & + (a_0 b_{k-3} + a_1 b_{k-2} + a_2 b_{k-1}) x^2 \\ & + \cdots \\ & + \left(\sum_{i=0}^{k-1} a_i b_i \right) x^{k-1} \\ & + \cdots \\ & + (a_{k-1} b_0) x^{2k-2} \end{aligned}$$

Therefore, the result of the inner product of two arrays can be easily achieved in the coefficient of x^{k-1}.

C. Inner Product of A Matrix and A Vector

A straightforward way of packing for matrix multiplication is creating one polynomial for each row of matrix A and one polynomial for each row of matrix B. However, it is a waste of space and time to pack and encrypt each row individually.

One observation is that the inner product of two $k-$element arrays would use up $2k-1$ coefficients, where the coefficient in the middle is our result, and all the others contain some garbage that we do not

need. Therefore, when k is much smaller than n, we can use the other coefficients that are not polluted to store other valid results.

When consider the situation that two matrix that $A \in \mathbb{Z}^{r\times k}$ and $B \in \mathbb{Z}^{k\times 1}$.

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,k-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,k-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{r-1,0} & a_{r-1,1} & a_{r-1,2} & \cdots & a_{r-1,k-1} \end{bmatrix}$$

$$B = \begin{bmatrix} b_0 & b_1 & b_2 & \cdots & b_{k-1} \end{bmatrix}^T$$

We propose to pack the matrix in the following way:

$$\begin{aligned} P_A(x) = & \underbrace{a_{0,0} + a_{0,1}x + \cdots + a_{0,k-1}x^{k-1}}_{A_0} \\ & + \underbrace{a_{1,0}x^k + a_{1,1}x^{k+1} + \cdots + a_{1,k}x^{2k-1}}_{A_1} \\ & + \cdots \\ & + \underbrace{a_{r-1,0}x^{(r-1)k} + \cdots + a_{r-1,k-1}x^{rk-1}}_{A_{r-1}} \end{aligned}$$

$$P_B = b_{k-1} + b_{k-2}x + b_{k-3}x^2 + \cdots + b_0x^{k-1}$$

As shown in Section III-B, the inner product of A_0 and B will use the coefficients $[0, 2k-2]$, and the result is in the coefficient of $k-1$. Similarly, the inner product of A_1 and B will use the coefficients $[k, 3k-2]$, and the result is in the coefficient of $2k-1$. To better illustrate where the valid results are in each coefficient, we plot the results in Figure 1. We introduce 'polluted space' in the coefficient space of the resulting polynomial. Each result of matrix multiplication is generated at fixed slots $mk-1, m \in [1, r]$, while the regions between these slots represent polluted space and are irrelevant.

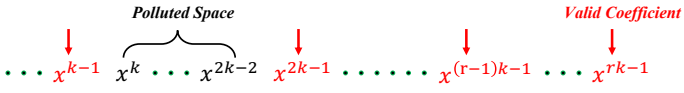


Fig. 1: Distribution of Valid Coefficients and Polluted Spaces in Polynomial Multiplication Results. Red dots indicate the positions of valid coefficients at multiples of k, while the black slots represent polluted spaces that require careful management in the polynomial encoding scheme.

The result of A_0B is valid only when the polluted space of $A_{r-1}B$ does not overlap with the valid result of A_0B, i.e., $rk-1+k-1 \leq n+k-2 \implies rk <= n$. In other words, as long as the total number of elements in A is no larger than the polynomial degree n, the inner product of AB can be finished in one round, with all the results in the coefficients of x^{mk-1}, where $m \in [1, r]$.

D. Inner Product of Two Matrices

For the multiplication of two matrices, we can encode the matrices based on the size of matrix A to maximize the utilization of the coefficient space.

1) When A is small ($rk \leq n$): We could pack multiple columns of B into the same polynomial and finish in one shot. The number of columns that could be packed together is decided by $\lfloor n/(rk) \rfloor$. For example, consider two matrices A and B of sizes 2×3 and 3×2 respectively:

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix}, \quad B = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{bmatrix}$$

And the result $C = A \times B$ is a 2×2 matrix.

$$C = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

Assuming the polynomial degree $n \geq 2\times 3\times 2$, we can encode the matrices in the following way:

$$P_A(x) = (a_{00} + a_{01}x + a_{02}x^2) + (a_{10} + a_{11}x + a_{12}x^2)x^3$$

$$P_B(x) = (b_{20} + b_{10}x + b_{00}x^2) + (b_{21} + b_{11}x + b_{01}x^2)x^6$$

The result of polynomial multiplication is:

$$\begin{aligned} P_C(x) = & P_A(x) \times P_B(x) \\ = & \cdots + c_{00}x^2 + \cdots + c_{10}x^5 + \cdots \\ & + c_{01}x^8 + \cdots + c_{11}x^{11} + \cdots \end{aligned}$$

Compared to packing B with individual columns, this packing scheme can significantly improve the overall performance by reducing the number of packing and encryption.

2) When A is large ($rk > n$): Similar to the tiling technique in general matrix multiplication, we could split the row into multiple smaller tiles that can fit in one polynomial. For example, consider two matrices A and B of sizes 2×6 and 6×2 respectively:

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \end{bmatrix}, \quad B = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \\ b_{30} & b_{31} \\ b_{40} & b_{41} \\ b_{50} & b_{51} \end{bmatrix}$$

and the result $C = A \times B$ is a 2×2 matrix.

$$C = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

For the ease of illustration, we assume the polynomial degree n is 3. We could encode matrix A using four polynomials, and could encode matrix B using four polynomials.

$$P_{A0}(x) = a_{00} + a_{01}x + a_{02}x^2, P_{A1}(x) = a_{03} + a_{04}x + a_{05}x^2$$

$$P_{A2}(x) = a_{10} + a_{11}x + a_{12}x^2, P_{A3}(x) = a_{13} + a_{14}x + a_{15}x^2$$

$$P_{B0}(x) = b_{20} + b_{10}x + b_{00}x^2, P_{B1}(x) = b_{50} + b_{40}x + b_{30}x^2$$

$$P_{B2}(x) = b_{21} + b_{11}x + b_{01}x^2, P_{B3}(x) = b_{51} + b_{41}x + b_{31}x^2$$

The result of polynomial multiplication is:

$$P_{C_{00}}(x) = P_{A0}(x) \times P_{B0}(x) + P_{A1}(x) \times P_{B1}(x)$$

$$P_{C_{01}}(x) = P_{A0}(x) \times P_{B2}(x) + P_{A1}(x) \times P_{B3}(x)$$

$$P_{C_{10}}(x) = P_{A1}(x) \times P_{B0}(x) + P_{A1}(x) \times P_{B1}(x)$$

$$P_{C_{11}}(x) = P_{A1}(x) \times P_{B2}(x) + P_{A1}(x) \times P_{B3}(x)$$

IV. Hardware Acceleration for the BFV Scheme

To evaluate our polynomial encoding method for GEMM, we develop hardware that implements the BFV homomorphic encryption. In our design, we take the Residue Number System (RNS) [20] and decompose the polynomial to q_{Num} distinct residue polynomials, each associated with a prime modulus q_i.

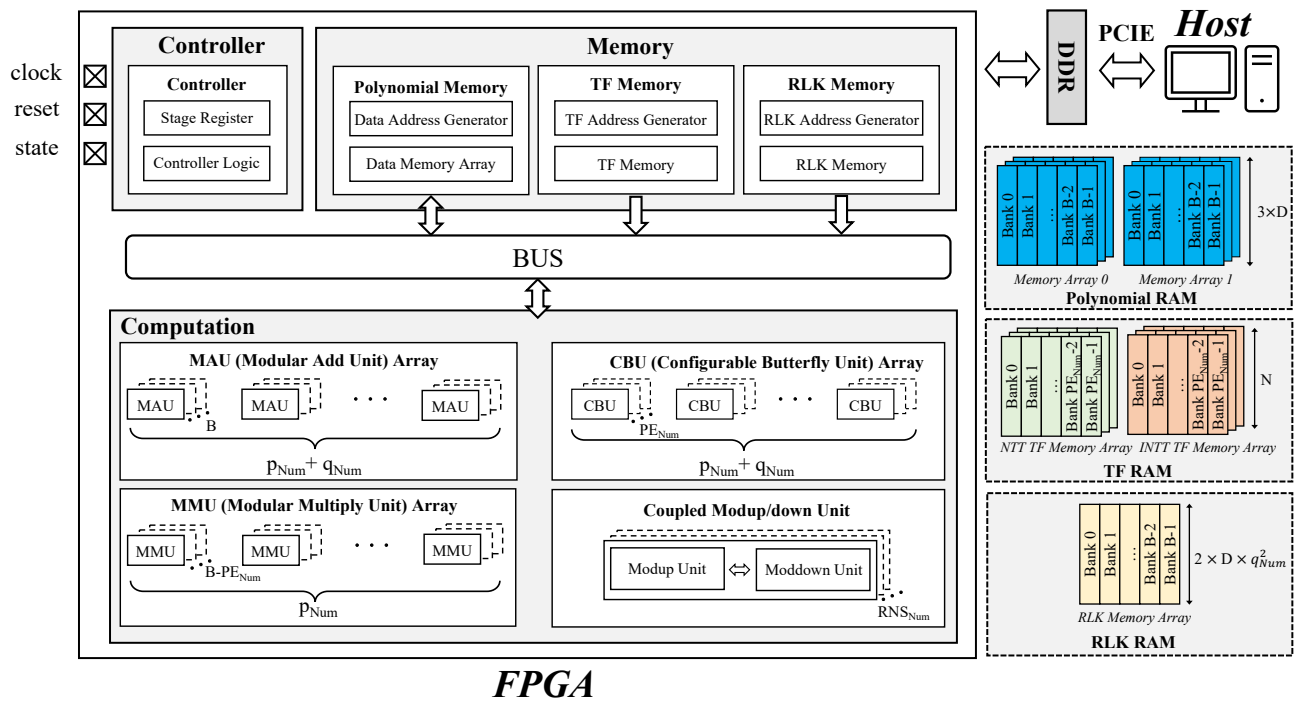


Fig. 2: Overall Architecture of Accelerator

A. Overall Achitecture

The overall architecture, depicted in Figure 2, is organized into three primary components: the memory module, the controller module, and the computation module. Additionally, it features a DDR interface and connection to the host system through a PCIe interface.

The controller module functions as a state machine, generating control signals in response to specific stimuli. This module is essential for scheduling various operations, including Modup, Moddown, NTT, INTT, Relinearization, Keyswitch, and others. One key functionality of the controller module is to generate read/write (R/W) state signals to manage the data flow, involving Data Memory, Twiddle Factor (TF) Memory, and Relinearization Key (RLK) Memory. The control logic allows the configurable butterfly unit to form diverse data paths, enabling the execution of different computational tasks and control data flow among different modules.

The data memory module includes an address generation unit and $2 \times (q_{\text{Num}} + p_{\text{Num}})$ dual-port BRAM arrays, each comprising B memory banks ($B = \sqrt{N}$) with a depth of $3D$ for polynomial degree $N = DB$. The address generator produces R/W addresses based on control signals. The TF memory stores twiddle factors for all moduli q_i and p_j required for NTT and INTT operations, totaling $PE_{\text{Num}} \times (p_{\text{Num}} + q_{\text{Num}}) \times N$ factors. Each Ultra RAM in the TF memory contains PE_{Num} banks with a depth of N and a 72-bit width on the FPGA U250, storing NTT and INTT twiddle factors in the upper and lower 36 bits of each row, respectively. These factors remain constant after initialization and are accessed as directed by control signals. The RLK memory utilizes BRAM to store the RLK, which is essential for efficiently converting three polynomials into two during relinearization. The RLK memory array consists of B memory banks, each with a depth of $2D \times q_{\text{Num}}^2$. Due to hardware resource limitations, the KeySwitch operation stores the RLK in a single memory array instead of multiple arrays.

The computation module comprises the Configurable Butterfly Unit (CBU) module, Coupled ModUp/Down module, Modular Addition Unit (MAU) module, and Modular Multiplication Unit (MMU) module. The CBU module, designed based on the work in [21], consists of $(q_{\text{Num}} + p_{\text{Num}})$ CBU arrays, each containing PE_{Num} butterfly units. These configurable units perform operations such as Number Theoretic Transform (NTT), Inverse NTT (INTT), and modular multiplication, with each CBU array associated with a modulus q_i or p_j. The Coupled ModUp/Down module facilitates basis scaling and extension operations essential for ciphertext-ciphertext multiplication and comprises RNS_{Num} units. To address the computational intensity of the KeySwitch operation, we introduce additional MAU and MMU modules, enhancing performance through increased parallelism and efficiency. The MAU array consists of $(q_{\text{Num}} + p_{\text{Num}})$ arrays, each containing B units. The MMU array, which operates in conjunction with the CBU array and is primarily utilized during KeySwitch operations, includes P_{Num} arrays, each with $(B - PE_{\text{Num}})$ MMU units.

B. Polynomial Packing Inference Schedule

Figure 3 illustrates the inference pipeline leveraging polynomial packing optimization techniques. The proposed framework comprises six distinct computational stages: polynomial packing, homomorphic encryption, secure inner product computation, homomorphic decryption, coefficient extraction, and non-linear activation transformation. In our design, only the input data is encrypted to protect user privacy, while the model weights remain in plaintext form to optimize computational efficiency[1].

[1]This design choice aligns with the typical privacy requirements in machine learning inference, where protecting user input data is paramount while model weights are considered public parameters.

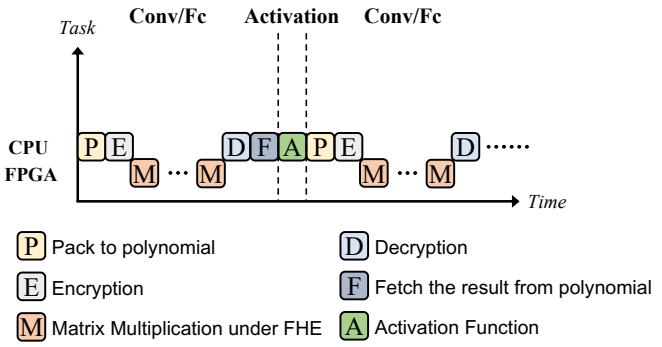


Fig. 3: Inference Schedule. The input data is encrypted to protect user privacy, while model weights remain in plaintext form for computational efficiency.

To achieve an optimal balance between computational efficiency and privacy preservation, we strategically partition the workload across heterogeneous computing platforms - executing the non-linear activation functions on plaintext data within the CPU environment while offloading the privacy-critical inner product operations (between encrypted inputs and plaintext weights) to the FPGA accelerator. This architectural design ensures both the confidentiality of sensitive user data and the computational efficiency of the inference pipeline.

V. Experimental Evaluation

A. Experimental Setup

The proposed accelerator was implemented in Chisel, translated to Verilog, and then synthesized using the Xilinx Vivado toolchain version 2024.1. To evaluate the performance, we deployed the accelerator on the commercial FPGA platform, AMD Xilinx Alveo U250. The software reference implementation was developed using SEAL [22]. For the CPU host, we used a 12th Gen Intel® Core™ i7-12700 processor with 32GB of memory. The AMD Xilinx Alveo U250 features 1,728,000 Lookup Tables (LUTs), 3,456,000 Flip-Flops (FFs), 12,288 Digital Signal Processors (DSPs), 1,280 Ultra RAM, and 2,000 Block RAM. Additionally, the U250 is equipped with 16 GB of DDR memory and supports high-speed data transfer through a PCIe interface.

B. Benchmarks and Security Parameters

Benchmarks: For experimental evaluation, we adopted two representative CNN architectures from [5], which were originally designed for MNIST and CIFAR-10 classification tasks. To accommodate the integer-only arithmetic constraints of the BFV scheme, we performed uniform 8-bit quantization on both model weights and input features. We enhanced the network architecture by replacing the original $Square$ activation with $ReLU$ and incorporating $BatchNorm$ layers after convolution operations, which improved the model's inference accuracy.

Security Parameters: The choice of BFV parameters significantly impacts ciphertext and plaintext data sizes, as well as computational complexity. To enable efficient inference without exceeding the plaintext modulus, we set the polynomial degree N to 2^{12}, the modulus q_i to 32 bits, and the total modulus Q to 224 bits.

C. Resource Utilization and MatMul Performance

Resource Utilization: In our hardware implementation, we configured the number of processing elements PE_{Num} to 8 and the number of RNS bases RNS_{Num} to 4. After synthesizing and implementing our design on the Xilinx Alveo U250 FPGA platform, the resource utilization analysis shows that our design consumes **98.93**% of available Look-Up Tables (LUTs), **8.29**% of Flip-Flops (FFs), **28.8**% of Block RAM (BRAM), **4.375**% of Ultra RAM (URAM), and **26.75**% of Digital Signal Processors (DSPs).

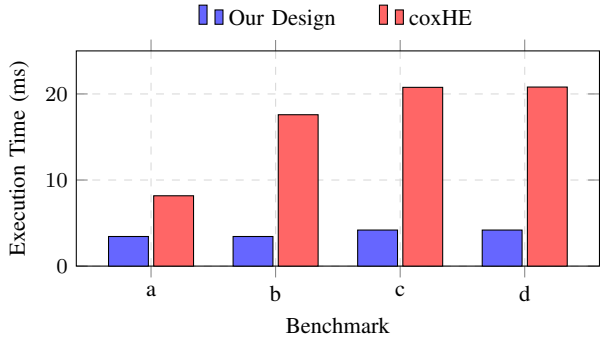


Fig. 4: Performance comparison between our design and the coxHE framework. For coxHE, benchmark **a** uses a polynomial degree of $N = 2^{12}$ with a 33-bit modulus (q_i). Benchmarks **b**–**d** employ $N = 2^{13}$ with 25-bit, 33-bit, and 43-bit modulus, respectively.

Small Matrix Multiplication: Matrix multiplication is the most time-intensive component of the inference process. Figure 4 presents a performance comparison between our design and the coxHE [23] framework, which is implemented under the CKKS scheme. Our evaluation focuses on the following matrix multiplication tasks: **a**: MatMul c-p: $(1 \times 2048) \cdot (2024 \times 1)$; **b**: MatMul c-p: $(1 \times 10) \cdot (10 \times 1)$; **c**: MatMul c-c: $(1 \times 2048) \cdot (2048 \times 1)$; **d**: MatMul c-c: $(1 \times 10) \cdot (10 \times 1)$.

In benchmarks **a** and **b**, our design achieves execution times of 3.439 ms, resulting in speedups of approximately $2.376\times$ and $5.114\times$ over coxHE's 8.17 ms and 17.58 ms, respectively. Similarly, in benchmarks **c** and **d**, our design reduces execution times to 4.188 ms, offering speedups of about $4.958\times$ and $4.967\times$compared to coxHE's 20.766 ms and 20.800 ms, respectively. The superior performance of our design is attributed to the ability to pack all data into a single polynomial of degree 4096, enabling a one-time computation process. Additionally, the execution time differences between benchmarks **a** and **b** compared to benchmarks **c** and **d** arise from the varying number of clock cycles required for c-c and c-p operations, respectively.

Large Matrix Multiplication: For large-scale matrices, we implement a decomposition strategy that partitions the matrix into smaller polynomial segments. Each segment is encoded into a polynomial of length 4096, followed by the necessary multiplications. Subsequently, the results are extracted and aggregated to obtain the final product. Specifically, for matrix multiplication of size 8192×8192, our hardware design completes the operation in 392,285.64 seconds under ciphertext operations. In stark contrast, our software implementation utilizing the SEAL library—employing a polynomial degree of 8192 and a coefficient modulus size of 218 bits—consumes approximately 7,413,087,362 seconds. This results in a speedup of approximately 1.89×10^4, thereby demonstrating that our polynomial encoding method achieves substantial performance improvements for large-scale matrix multiplication.

D. Inference Performance on Benchmark Datasets

Accuracy: By optimizing the network architecture to incorporate more complex activation functions, we achieved superior accuracy compared to previous works. Specifically, on the MNIST dataset, our model attained an accuracy of **99.11%**, outperforming FxHENN [16] by 0.21% and Lola [24] by 0.16%. On the CIFAR-10 dataset, our model reached an accuracy of **78.99%**, surpassing both FxHENN and Lola by 4.89%.

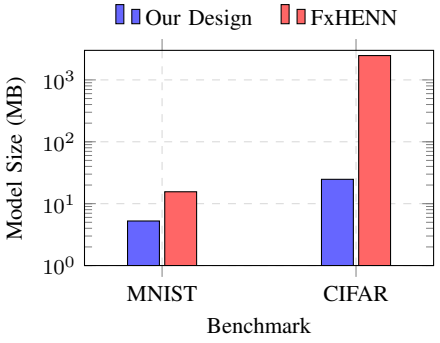


Fig. 5: Model Size of MNIST and CIFAR

Model Size: The model size is determined by the number of polynomials and the modulus size q. Our polynomial encoding method compresses the model by packing a significant number of weights into a single polynomial, thereby reducing the overall size. Figure 5 illustrates the model sizes for different benchmarks. For the MNIST dataset, our polynomial encoding occupies 15 MB compared to FxHENN's 15.57 MB, resulting in a 3.67% reduction in model size. Similarly, for the CIFAR dataset, our design requires 169.5 MB, whereas FxHENN demands 2,471.25 MB, reducing the model size by 93.14%. The model size is primarily influenced by the number of input and output channels, as the weights are packaged in a channel-first manner during the encoding process.

Performance: We compare the inference speed of our accelerator with prior works based on the BFV scheme, including CryptoNets [25], Lola [24], and Falcon [26], as well as with FxHENN [16] based on the CKKS scheme, using the MNIST and CIFAR datasets. Table I presents a detailed performance comparison.

TABLE I: Inference Performance Comparison

Implementation	Platform	MNIST (s)	CIFAR (s)
[25]†	Intel Xeon E5-1620L	205	-
[24]†	Azure B8ms (8 vCPUs)	2.2	730
[26]†	Azure B8ms (8 vCPUs)	1.2	107
[16]†	ALINX ACU15EG	0.19	54.1
Ours†	AMD Xilinx U250	**0.045**	**13.53**
Ours$	AMD Xilinx U250	**0.077**	**27.3**

† Encrypted inputs and plaintext weights.
$ Encrypted inputs and encrypted weights.

As shown in Table I, our method significantly outperforms previous implementations by optimizing convolutional and fully connected layers. These optimizations reduce the inference latency to 0.045ms per image for encrypted inputs with plaintext weights and 0.077ms per image for fully encrypted inputs and weights. Compared to FxHENN, our accelerator achieves speedups of 4.22× under encrypted inputs and plaintext weights and 2.47× when both inputs and weights are fully encrypted. For inference on the CIFAR-10 dataset, the experiment processes one image in just 13.53ms with plaintext weights and 27.3ms with encrypted weights, which are 3.99× and 1.98× faster than FxHENN.

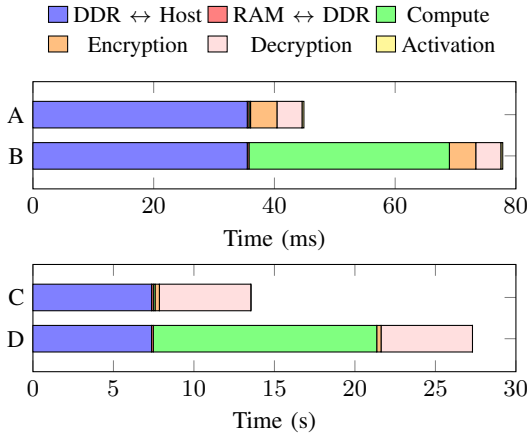


Fig. 6: Time Breakdown of Different Benchmarks. **A**: MNIST c-p; **B**: MNIST c-c; **C**: CIFAR c-p; **D**: CIFAR c-c

Time Breakdown: The time breakdown for the different benchmarks is presented in Figure 6. Benchmarks **A** and **C** involve encrypted inputs with plaintext weights, while benchmarks **B** and **D** involve both encrypted inputs and encrypted weights.

In scenarios with encrypted inputs and plaintext weights, data transformations between the host and the FPGA constitute the majority of the inference time, accounting for approximately **54%** to **79%**. Decryption and encryption processes contribute around **19%** and **44%** of the total time, depending on the benchmark. Conversely, in fully encrypted configurations, compute operations become the dominant factor, representing roughly **43%** to **51%** of the inference time. Additionally, data transformations still account for a significant portion, between **28%** and **46%**. These results highlight that data transmission remains a major contributor to inference latency across all benchmarks. However, in fully encrypted environments, the computational overhead introduced by encryption operations further elevates the overall latency.

VI. Conclusion

General Matrix-Matrix Multiplication with Fully Homomorphic Encryption presents significant challenges, including slow computation speeds, complex operations, and substantial overhead associated with processing encrypted data. To address these limitations, we introduce an enhanced data-packing method for matrix multiplication and extend this approach to accelerate both convolutional and fully connected layers. Additionally, we develop a novel computational schedule that incorporates more efficient activation functions, thereby improving network accuracy. To evaluate our method, we implemented dedicated hardware for homomorphic encryption using the BFV scheme. Deployed on the AMD Xilinx U250 FPGA platform, our hardware demonstrates exceptional performance in inference tasks on the MNIST and CIFAR-10 datasets. Compared to previous implementations, our hardware achieves significant speedups of up to 4.22× for MNIST and 3.99× for CIFAR-10, while also attaining higher accuracy. These results underscore the effectiveness of our enhancements in mitigating the inherent challenges of GEMM with FHE, leading to more efficient and accurate neural network inference.

REFERENCES

[1] T. Zhu, D. Ye, W. Wang, W. Zhou, and P. S. Yu, "More than privacy: Applying differential privacy in key areas of artificial intelligence," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 6, pp. 2824–2843, 2022.

[2] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, 2021.

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 12:1–12:19, 2019.

[4] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: towards model privacy at the edge using trusted execution environments," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 161–174.

[5] A. Brutzkus and A. Globerson, "Low-degree polynomial approximation of relu with applications to neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 902–910.

[6] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. ACM, 2011, pp. 113–124.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. J. F. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, vol. 10031. Springer, 2016, pp. 3–33.

[8] S. Halevi and V. Shoup, "Algorithms in helib," in *Annual Cryptology Conference*, ser. Lecture Notes in Computer Science, vol. 8616. Springer, 2014, pp. 554–571.

[9] A. Costache and N. P. Smart, "Which ring-based somewhat homomorphic encryption scheme is best?" in *International Conference on Cryptography and Coding*, ser. Lecture Notes in Computer Science, vol. 10076. Springer, 2016, pp. 325–340.

[10] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 201–210. [Online]. Available: https://proceedings.mlr.press/v48/gilad-bachrach16.html

[11] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "Tensorfhe: Achieving practical computation on encrypted data using gpgpu," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 922–934.

[12] X. Hu, Z. Li, Z. Wang, and X. Lu, "Alt: Area-efficient and low-latency fpga design for torus fully homomorphic encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 4, pp. 645–657, 2024.

[13] R. Agrawal, A. Chandrakasan, and A. Joshi, "Heap: A fully homomorphic encryption accelerator with parallelized bootstrapping," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 756–769.

[14] Z. Xu, Y. Yang, R. Kannan, and V. K. Prasanna, "Bandwidth efficient homomorphic encrypted discrete fourier transform acceleration on fpga," in *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2024, pp. 1–12.

[15] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 26–39.

[16] Y. Zhu, X. Wang, L. Ju, and S. Guo, "Fxhenn: Fpga-based acceleration framework for homomorphic encrypted cnn inference," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 896–907.

[17] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Annual cryptology conference*. Springer, 2012, pp. 868–886.

[18] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[19] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[20] S. Halevi, Y. Polyakov, and V. Shoup, "An improved rns variant of the bfv homomorphic encryption scheme," in *Topics in Cryptology – CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2019, p. 83–105. [Online]. Available: https://doi.org/10.1007/978-3-030-12612-4_5

[21] X. Meng, Z. Jiang, and Y. Lyu, "Hf-ntt: Hazard-free dataflow accelerator for number theoretic transform," 2024. [Online]. Available: https://arxiv.org/abs/2410.04805

[22] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[23] M. Han, Y. Zhu, Q. Lou, Z. Zhou, S. Guo, and L. Ju, "coxhe: A software-hardware co-design framework for fpga acceleration of homomorphic computation," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1353–1358.

[24] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 812–821. [Online]. Available: https://proceedings.mlr.press/v97/brutzkus19a.html

[25] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 48. PMLR, 2016, pp. 201–210. [Online]. Available: http://proceedings.mlr.press/v48/gilad-bachrach16.html

[26] T. Xu, M. Li, R. Wang, and R. Huang, "Falcon: Accelerating homomorphically encrypted convolutions for efficient private mobile network inference," 2023. [Online]. Available: https://arxiv.org/abs/2308.13189

DROIDFUZZ: Proprietary Driver Fuzzing for Embedded Android Devices

Jianzhong Liu*, Yuheng Shen*, Yifei Chu*, Qiang Zhang†, Heyuan Shi‡, Wanli Chang†, Yu Jiang*✉
*School of Software, Tsinghua University Hunan University Central South University
Emails: *{liujz21, shenyh20, chuyf24}@mails.tsinghua.edu.cn, jiangyu198964@126.com
†zhangqiang9413@126.com, wanli.chang.rts@gmail.com ‡hey.shi@foxmail.com

***Abstract*—Embedded Android Devices have proliferated in many security-critical embedded scenarios, requiring sufficient testing to root out vulnerabilities. Due to Android's architecture, which uses a Hardware Abstraction Layer (HAL) for vendor-specific driver implementations, traditional kernel testing techniques cannot detect such bugs within the actual driver logic, which are commonly proprietary and vendor-specific. In this paper, we propose DroidFuzz, an embedded Android system fuzzer that targets such vendor-specific driver implementations to find such bugs. Through leveraging pre-testing HAL driver probing, kernel-user relational payload generation, and cross-boundary execution state feedback, we effectively test the proprietary drivers in both the kernel and the HAL layer. We implemented DroidFuzz and evaluated its effectiveness on 7 embedded Android devices, and found 12 security-critical previously unknown bugs, all of which have been confirmed by the respective vendors.**

***Index Terms*—Android testing, fuzz testing, embedded Linux, bug detection**

I. INTRODUCTION

The widespread adoption of Android in mobile devices has led to its increasing presence in embedded systems, where it is used to power a diverse range of applications, from set-top boxes and automated kiosks to automotive systems, medical devices, and industrial control systems. As the complexity of these systems grows, so does the attack surface, with device drivers emerging as a critical component that interacts directly with hardware components and manages their operations. With millions of lines of code in modern Android-based embedded systems, ensuring the security and reliability of device drivers has become a pressing concern.

Android's architecture utilizes a Hardware Abstraction Layer (HAL) to allow high-level applications to interface with hardware devices and peripherals. HAL is situated between the Android framework and the Linux kernel, where it provides a standardized interface for accessing hardware resources, while also allowing hardware vendors to implement proprietary and closed-source code that is specific to their hardware platforms. As a result, much of the device driver's logic resides within the HAL layer, such that a significant portion of the driver codebase remains opaque and inaccessible to external scrutiny, making it challenging to detect and mitigate potential security vulnerabilities or bugs within the HAL layer.

We use the example of CVE-2021-0673 [26] to demonstrate this point. This vulnerability is a heap-buffer-overflow found in the audio component of MediaTek chipset's HAL, which at the time accounted for 43% of Android devices shipped. When exploited, the bug results in privilege escalation, allowing malicious attackers to gain access to the entire system. Due to MediaTek's HAL being closed source and proprietary, applying public scrutiny to ensure the safety of its components is difficult, hindering the bug discovery process, consequently demanding more effective and systematic methods in rooting out such vulnerabilities in Android proprietary drivers.

Software testing is a crucial step in ensuring the reliability, security, and quality of modern software systems. Among various testing techniques, fuzz testing (fuzzing) has emerged as an effective approach for detecting bugs and vulnerabilities in software applications. It mainly involves providing large amounts of randomized input to a target program to trigger bugs within. In the context of operating system kernels, fuzzing typically involves generating payloads that mimic real-world workloads and injecting them into the kernel through various interfaces such as system calls. Syzkaller [39] is a state-of-the-art kernel fuzzer, and has found more than 5000 bugs in the Linux kernel that have subsequently been confirmed and fixed.

However, despite the importance of driver security, existing research and industry application of Android fuzzing has primarily focused on fuzzing and testing applications and system frameworks, leaving a significant gap in the state of the art for effective fuzzing techniques tailored to detecting bugs in Android's device drivers, consisting of code in kernel drivers and HAL drivers. Established methods for kernel fuzzing on Android mainly focuses on testing the Linux kernel itself, whereas Android's HAL layer, which is in the userspace, cannot be directly reached using these methods. Furthermore, effectively covering Android's kernel drivers require meaningful interaction with its corresponding HAL, which is difficult to emulate using simply system call invocations.

This research aims to address this knowledge gap by exploring new methodologies for adapting fuzzing techniques to the unique characteristics of HAL drivers in embedded Android frameworks. To do this, we need to solve the following challenges effectively. 1) First, in addition to system call interfaces, we need to model the interfaces exposed by HAL to the Android runtime and libraries such that we can generate effective invocations and payloads to the HAL's interfaces, which is proprietary and mostly undocumented, barring traditional static analysis or LLM-based methods from achieving such goals. 2) Second, we face the critical problem of effectively generating test cases that are meaningful to both Android's kernel and HAL, which are further complicated as HAL's underlying functionalities interact with the kernel through system calls, thus direct randomized and static generation yields poor results. 3) Finally, as the kernel and HAL run on different permission modes and memory spaces, and provide differing execution state tracking, we cannot directly interpret the execution state of both entities directly to identify interesting inputs, as the semantics of these are vastly different.

To address the aforementioned challenges, we propose DROIDFUZZ, a fuzzer targeting proprietary drivers in embedded Android devices. In order to effectively conduct fuzz testing on both Android's kernel drivers and HAL layer, DROIDFUZZ introduces three approaches that individually address the challenges. First, DROIDFUZZ provides a *pre-testing HAL driver probing* pass, where the HAL is loaded, probed, to obtain its exposed interfaces, argument types, and the weighting factors for each interface. Second, DROIDFUZZ also integrates a *kernel-user relational payload generation* technique, where it allows the fuzzer to produce test cases that synergistically

tests the kernel drivers and HAL components in an embedded Android device. Finally, we design a *cross-boundary execution state feedback* mechanism, where it coalesces the kernel's code coverage and HAL's kernel execution behavior to analyze the state changes in both components, and in turn assist in further input generation to produce more meaningful input payloads.

We implemented DROIDFUZZ on 7 embedded Android devices from well-known hardware vendors, including Xiaomi, Sunmi, Raspberry Pi, etc., and evaluated its driver code coverage and bugs finding capabilities on these devices. We further compared our performance to that of Syzkaller to demonstrate the effectiveness in tapping into embedded Android's proprietary driver framework. Our results show that DROIDFUZZ found 12 new bugs in these firmware, which have 7 from kernel drivers and 5 from the HAL layers, where Syzkaller was only able to find 2, both of which are from the kernel. Furthermore, through evaluating per-driver coverage in the kernel, DROIDFUZZ achieves a 17% increase on average, demonstrating that interactions with HAL can create meaningful workloads that further traverse the states of the kernel drivers.

II. BACKGROUND AND RELATED WORK

A. Android on Embedded Devices

Embedded Android is the use of the Android operating system in embedded devices, such as automated kiosks, set-top boxes, medical devices, and industrial control systems [13]. These devices typically use various peripherals such as displays, cameras, and sensors, which often require many proprietary drivers to run [43].

Android in embedded systems allows developing complex applications, through its high-level APIs and abstractions that other embedded OSs fail to offer [32]. As shown in Figure 1, an embedded Android system consists of the Linux kernel, the Hardware Abstraction Layer (HAL), Android Runtime and Libraries, and the Android Applications [12]. The Linux kernel [27] provides the underlying system services such as scheduling, process isolation, and kernel drivers, whereas Android's framework provides the core APIs for building applications. The HAL is sandwiched between the two, providing a standardized interface for accessing hardware resources and abstracting away low-level details [14].

B. Fuzz Testing

Fuzz testing, a.k.a. fuzzing, is a software testing technique used to detect security vulnerabilities and bugs in applications [21]–[23]. It mainly involves feeding invalid, unexpected, or random input to an target program and monitoring its behavior for crashes, hangs, or other anomalous behavior. Coverage-guided mutation-based fuzzing uses genetic algorithms to evaluate the effectiveness of test cases by whether they trigger new execution states to preserve those for further mutation into new test cases [16], [31]. Generation-based fuzzing uses input format specifications to produce valid and well-formed test cases. Fuzz testing is widely used in various domains, including protocol security [19], program library security [2]–[4], [10], and system vulnerability detection [5], [20].

Syzkaller is a state-of-the-art fuzzer that leverages both coverage-guided mutation-based fuzzing and generation-based fuzzing techniques through the use of *kcov* [35] and *Syzlang* [40]. Many works use Syzkaller as a basis for improving fuzzing techniques. For instance, Moonshine [28] aims to distill high-quality initial test cases for Syzkaller. Horus [17] reduces data transfer overheads between the fuzzer and manager by offloading RPCs to direct memory accesses.

kAFL [33] is another kernel fuzzer that maximizes throughput by leveraging architectural features in Intel processors. HEALER [37] is a kernel fuzzer targeted at system call relation learning to generate more effective test cases.

There are also works that fuzz tests Android's systems and framework [36], [38]. Difuze [6] addresses the problem of kernel driver fuzzing by introducing *interface-aware fuzzing*, which extracts valid commands and associated data structures through static analysis. Atlas [42] uses static analysis to deduce correct calling sequences and parameters of native APIs in closed-source Android native libraries and uses heuristics for optimizing the generated harness. FuzzGen++ [29] generates fuzzing driver programs for OEM Android libraries and applies automatic cutoff for low-quality driver programs. In comparison, DROIDFUZZ addresses the problem of efficient proprietary embedded Android driver fuzzing, which also requires the joint fuzzing of both the Linux kernel and the HAL layer.

Works aimed towards fuzzing embedded systems [1], [15], [24], [25], [41] are appearing. Tardis [34] adapts coverage-guided fuzzing to embedded operating systems running on emulators. EmbSan [18] addresses the issue of generalized porting of sanitizers to embedded platforms. Gustave [7] transfigurates embedded OS fuzzing into application fuzzing through input conversion.

III. MOTIVATION

Unlike traditional Android application or framework fuzzing, testing proprietary drivers requires generating effective payloads that trigger complex interactions between the kernel and HAL components. To demonstrate the difficulties involved, we show how an embedded Android application accesses the hardware in Figure 1 to demonstrate the complex interactions between HAL drivers and kernel drivers. As shown in the figure, Android's HAL resides between the high-level application frameworks and the low-level kernel, where it accesses low-level interfaces through device files and abstract them into high-level APIs. HAL drivers are also stateful and have vendor-specific features, allowing proprietary drivers to be shipped without source code, common practice for Android vendors.

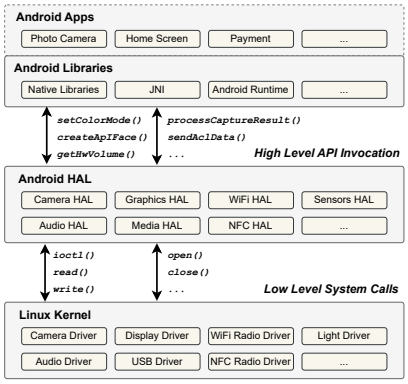


Fig. 1. Architecture of Android Relative to Apps Accessing Hardware

Fuzzing only kernel system calls or the HAL interfaces as a library is not sufficient to trigger complex and meaningful interactions between the HAL and kernel drivers, as profound bugs often require correct states from both components to trigger. Therefore, our goal is to develop a method that can fuzz proprietary drivers, by identifying driver interfaces and interpreting their affinity with other APIs and system calls, i.e. *relations*, generate meaningful sequences of API invocations and system calls that direct fuzzing towards exploring states in the proprietary drivers, and interpret execution feedback from both the kernel and the HAL layer as coverage, ultimately providing an effective way to detect bugs in embedded Android's

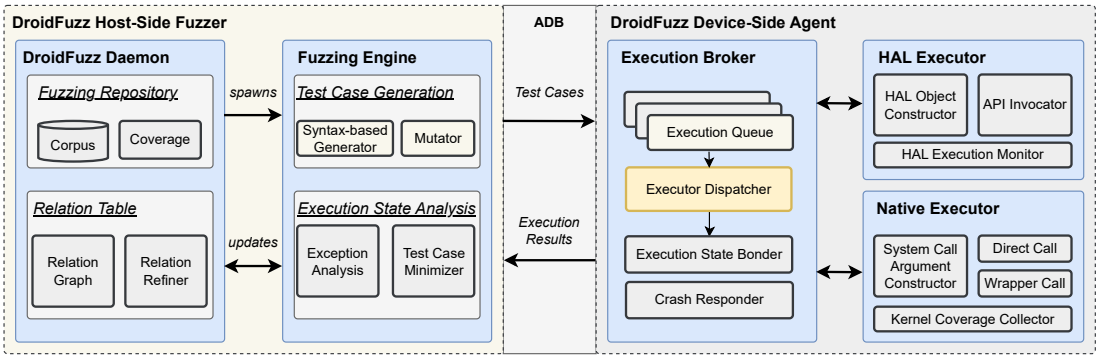


Fig. 2. Overview of DROIDFUZZ's Architecture

drivers. Designing such methods poses several challenges that need to be adequately addressed:

First, we need to determine the list of interfaces to test and the argument syntax and semantics of each individual interface. While previous work such as Difuze and Syzkaller have provided methods and syntax for system calls, the interfaces on the HAL layer are largely undocumented and lack available source code, making code-level analysis difficult.

Second, the challenge lies in generating meaningful sequences of HAL API invocations and system calls that effectively test both sides synergistically. The kernel and HAL are tightly coupled, with the HAL providing a standardized interface for accessing hardware resources, while the kernel provides the underlying operating system services. To detect profound bugs, we need to generate sequences of API invocations and system calls that trigger these interactions and exercise both components simultaneously.

The last challenge is interpreting the execution feedback as coverage also poses significant challenges due to its cross-boundary characteristics. The HAL layer is closed-source, making it difficult to obtain precise information about the internal workings of this component. Furthermore, the kernel's behavior is deeply intertwined with the HAL's behavior, making it challenging to determine whether a particular sequence of API invocations and system calls has effectively exercised both components.

IV. DROIDFUZZ DESIGN

DROIDFUZZ is an operating system fuzzer that proposes solutions to the aforementioned challenges. It takes inspiration from state-of-the-art kernel fuzzers, including techniques such as system call description-based generation and coverage-guided mutation. We show the overall architecture diagram of DROIDFUZZ in Figure 2. The major components of the fuzzing harness include the host-side Daemon and Fuzzing Engine, and the device-side Execution Broker, HAL Executor, and Native Executor, of which each component will be briefly outlined in Section IV-A.

Our primary contributions are mainly covered in the following three designs: 1) a *pre-testing HAL driver probing* pass to obtain the HAL's exposed interfaces and associated argument types, and assess *weights* of each interface (Section IV-B); 2) a *kernel-user relational payload generation* approach that allows the fuzzer produce test cases that jointly tests the HAL drivers and corresponding kernel drivers (Section IV-C); 3) a *cross-boundary execution state feedback* mechanism that interprets the HAL's execution behavior to merge with the kernel's code coverage for uniform analysis of new states, and further produce more meaningful input payloads (Section IV-D).

A. Fuzzing Harness and Execution Agents

To facilitate proprietary driver fuzzing for embedded Android devices, we compartmentalize each required functionality into separate components, including DROIDFUZZ's Daemon, Fuzzing Engines, Execution Brokers, and Executors for HAL and Native system calls.

The root process of DROIDFUZZ is the Daemon, which mainly coordinates synchronization between fuzzing processes and maintains persistent data, such as the *seed corpus*, overall coverage statistics, and relation table which records relations' weights between interfaces and system calls.

When the daemon finishes initialization, it spawns one instance of a Fuzzing Engine for each device. The Fuzzing Engine produces test cases for execution on the target device, and subsequently analyzes feedback for each execution. Communication to the target device leverages the Android Debug Bridge (ADB) [11]. Test cases generated are sequences of HAL interface and Linux kernel system call invocations in a Domain Specific Langauge (DSL) form. The engine also receives execution state information from the device to test against previous runs for any new behavior.

The aforementioned two components are run on the host machine, while the following components are run on the device-under-test.

The Execution Broker is responsible for reliably communicating test cases to run and execution results with its parent fuzzing engine. It spawns a HAL executor and Native executor to run each HAL API or system call invocation. It maintains an internal execution queue that holds all test cases waiting to be executed, where each element of a sequence is dispatched according to their type. The feedback is then bonded to form a uniform feedback statistic, and is passed back to the fuzzing engine for analysis.

The HAL and Native Executor uses the instantiated DSL and constructs corresponding dependencies of objects as required. It then invokes the relevant API or system call, after which its execution feedback is collected and returned to the broker.

B. Pre-Testing HAL Driver Probing

Direct extraction of HAL interfaces is not easily achieved, as their descriptions from proprietary HALs in release builds of the firmware are not readily available. Instead, we take a poke and probe approach to observe actual invocations of the HAL from Android applications and extract a set of interfaces through observation of how the Android framework communicates with the HAL.

Essentially, we use two components, a Poke application running atop the Android framework, and a probe utility that runs natively, i.e. without Android's abstractions. The probe utility first uses Android

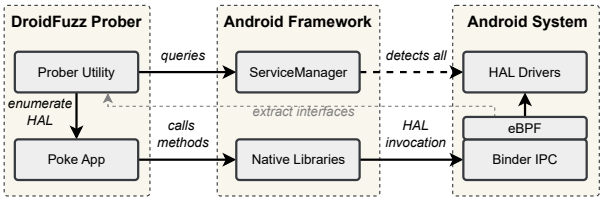


Fig. 3. Process of HAL Interface Probing

utilities (e.g. *lshal* and ServiceManager) to obtain a list of running services and HALs on the device. Then, for each running HAL service, it passes the relevant information to the Poke application, which then requests the service's interface through Android's ServiceManager. Next, the probe utility inserts eBPF code into the kernel that monitors for accesses to Binder IPC from the Poke app. The Poke application then conducts a short trial of all exposed interfaces by marshaling the parameters and invoking the corresponding methods reflected by ServiceManager, which is called into the native libraries, where it is translated into Binder IPC calls, during which the interaction is recorded from the eBPF hooks. The prober utility extracts the actual IPC data between the HAL and the Poke App, and filters out relevant interfaces and arguments. This process is depicted in Figure 3.

We also rank a HAL interface by calculating normalized occurrence through observing the number of instances that a certain interface is triggered through invoking high-level Android APIs. This gives the fuzzer more information on which interfaces to invoke during testing.

C. Kernel-User Relational Payload Generation

Generating effective test cases that traverse deeply into the execution states of drivers in both the kernel and HAL requires the fuzzer to understand the *relations* between HAL interfaces and kernel system calls. To do this, we construct a *relation graph* $\mathrm{G_{rel}} = (\mathrm{V}, \mathrm{E})$, where the set of vertices $\mathrm{V} = \{(v, w) | v \in S \vee v \in H, w \in (0, 1)\}$, of which S is the set of individual system calls and H is the set of individual HAL interfaces, and edges in E being directed and weighted. Each vertex also carries a fixed weight w, which represents the ranking of each individual system call or HAL interface, and corresponds to the probability at which the system call or interface is chosen during generation as the *base invocation*. The direction of edges in E represents the perceived dependency between the kernel system calls or HAL interfaces, whereas the weights represent the confidence regarding the dependencies between the system calls or interfaces.

During initialization, we create the relation graph with the set of vertices being filled with all system calls and HAL interfaces, and their weights from either system call descriptions or evaluated interface weights. The set of edges is initialized to an empty set, i.e. $E = \emptyset$. As we fuzz Android drivers, we detect new coverage and update the edges and weights between the HAL APIs and kernel system calls accordingly. When a new coverage is detected, we *minimize* the call to the bare bones API and system calls, ensuring that only the most essential invocations that trigger the same execution behavior are exercised. The new relations are then saved into the relation graph with the maximum normalized weights through the following process. For each adjacent pair of calls a and b, where the dependency is $a \rightarrow b$, their weights are calculated using the formula:

$$w_{(a,b)} = 1 - \Sigma_{\forall e=(x,b), x \neq a} w_{(x,b)}/2 \quad (1)$$

whereas weights for other edges with the same endpoint are halved.

Upon generating an input payload, we randomly pick a system call or an interface as the *base invocation*, i.e. the API or system call that acts as the basis for all depending invocations to form a sequence, based on the weights of each vertex. We instantiate the call in the DSL with parameters and objects based on the descriptions, using a combination of syntax-based generation and historical payload mutation. After this, we traverse the relation graph from the current vertex to a dependent vertex with a probability based on the edge weight. We may choose to stop altogether if the random value dictates such. Then, we repeat the process given above. After ending such a search, we iterate over all calls sequentially, find unresolved argument or parameter values, and find *producer calls*, i.e. system calls or API invocations that return the required argument values, instantiate the call accordingly, and insert it into the call sequence as a prefix to the current call. Finally, the generated call sequence, in the DSL representation, is passed to the device for execution.

To encourage more diverse payloads and prevent our fuzzing process from getting stuck in a local optimum, we periodically reduce the weights of all learned relations by multiplying all edge weights with a factor less than 1. This reduction process incentivizes DROIDFUZZ to explore different interaction paths between the HAL APIs and kernel system calls during test case generation, leading to more comprehensive coverage of the Android drivers.

D. Cross-Boundary Execution State Feedback

To conduct effective fuzzing of Android drivers in both the kernel and HAL components, we need to gather execution state feedback that spans across these two boundaries. For the Linux kernel, we can directly utilize debugging facilities such as *kcov* by recompiling the kernel with relevant configurations enabled. However, due to the closed-source nature of most vendor-specific HALs, we cannot directly employ tools such as LLVM's SanitizerCoverage [30] to evaluate its code coverage.

Our observation is that, as HAL abstracts low-level details into high-level interfaces, we can monitor the system call invocation behavior of the HAL layer to identify new behavior. However, we cannot directly interpret kernel code coverage, as it only records the system calls and code blocks executed, but disregards the order in which they are executed.

Thus, we use *directional* system call invocation coverage to reflect the execution behavior of the HAL layer's code. This requires that the HAL executor inserts eBPF probes into the kernel at runtime to detect system calls originating from the HAL. When the HAL executor detects system calls originating from the HAL layer, we record the specifics, including its number, critical position arguments (e.g. *request* in *ioctl()*), and the order in which it appears.

To mimic the output of code coverage, we use a lookup table compiled at initialization consisting of all possible system calls, including *specialized* system calls, which divide system calls that take generalized argument (e.g. *ioctl()*) according to their critical arguments and assign them unique IDs. We arrange the coverage into a sequence of system call IDs that correspond to the specific system calls invoked by the HAL, and append it to the kernel code coverage figures obtained from *kcov*. This joint-state feedback represents a comprehensive view of the execution state of both the kernel and HAL components, allowing us to understand how these two components interact and identify potential issues in their interaction. The analysis logic for both types of coverage remain the same, allowing for simplified processing while gaining comprehensive understandings of how the kernel and HAL components interact.

V. Implementation and Evaluation

We implemented DroidFuzz using 25930 lines of Rust and Go for the daemon, fuzzing engine, and broker components, 1694 lines of C for the executor components and prober utility, and 503 lines of Java for the Poke App. Our implementation borrowed system call descriptions and native executor components from Syzkaller to execute test payloads intended for the kernel. To demonstrate our applicability to real-world embedded Android devices, we adapted DroidFuzz to test 7 devices, which encompass off-the-shelf systems and development boards, all from renowned hardware vendors in the mobile and embedded space using processors commonly used in embedded Android devices. The list of the devices are shown in Table I, with their vendors, CPU architecture, the AOSP version and kernel version of their firmware specified.

TABLE I
List of Embedded Android Devices Tested

ID	Device	Vendor	Arch.	AOSP	Kernel
A1	Phone Dev Board	Xiaomi	*aarch64*	15	6.6
A2	Tablet Dev Board	Xiaomi	*aarch64*	15	6.6
B	Pi 5	Raspberry Pi	*aarch64*	15	6.6
C1	Commercial Tablet	Sunmi	*aarch64*	13	5.15
C2	Cashier Kiosk	Sunmi	*aarch64*	13	5.15
D	LubanCat 5	EmbedFire	*aarch64*	13	5.10
E	UP Core Plus	AAEON	*amd64*	13	5.10

To evaluate the effectiveness of our approach, we ran DroidFuzz on these embedded devices to assess its ability to find real-world bugs on the 7 embedded devices, and its effectiveness in covering driver code compared to state-of-the-art fuzzing approaches. We also analyzed the individual contributions of pre-testing HAL driver probing, kernel-user relational payload generation, and cross-boundary execution state feedback by performing ablation testing that removes each component for a controlled experiment.

A. Evaluation Settings

All experiments are conducted on a host server with dual Intel Xeon Silver 4120R CPUs, 192GiB of RAM, and running *amd64* Debian Linux 12.8. We use one device per experiment to conduct testing. The devices used in this experiment are running rooted firmware with their kernels recompiled with *kcov* and KASAN [9] enabled. DroidFuzz is configured to reboot the target devices upon encountering any bugs during testing, including kernel panics, assertions, and HAL errors. The versions of Syzkaller and Difuze used are commits *fb88827* and *3290997*. Each experiment is repeated for 10 times to eliminate statistical errors. We use the Mann-Whitney U Test to assess the existence of statistical significance, where data groups that do not exhibit such significance will be labelled explicitly.

B. Bug Detection

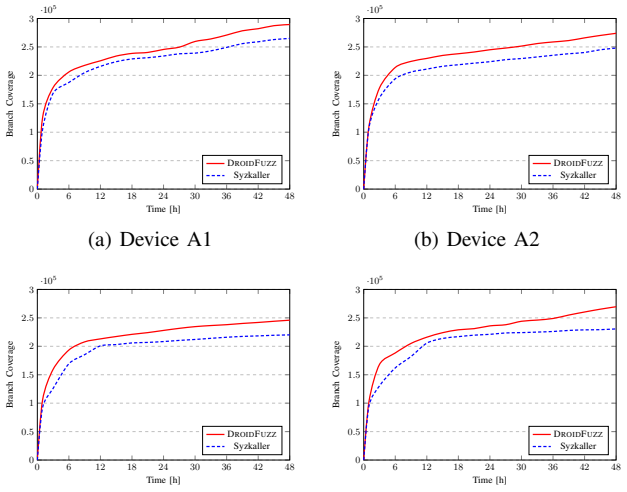
To evaluate DroidFuzz's bug finding abilities, we ran DroidFuzz on each target embedded device for 144 hours. All bugs triggered were initially minimized, deduplicated, and reproduced. Manual effort was also involved in analyzing the bugs to detect duplicates and recover corrupted log messages. Over the course of testing, DroidFuzz found 12 new bugs in the devices' respective firmwares, both in the kernel and HAL layer. All bugs have been confirmed by the respective vendors, where bugs fixes are underway at the time of writing. Some bugs have redacted information in their Bug info column as relevant details are currently embargoed.

The list of bugs are shown in Table II. As shown in the table, the new bugs found by DroidFuzz are found in both the kernel and HAL components, demonstrating that DroidFuzz is capable of effectively finding bugs in vendor-specific HAL drivers itself. Specifically, 3 bugs triggered crashes in the HAL layer, whereas the other 9 bugs were found in the kernel, showing the versatility of DroidFuzz, being capable of jointly fuzzing drivers in both the system kernel and HAL layer. As expected, the bugs triggered in the HAL are all architectural exceptions such as segmentation faults, demonstrating the effectiveness of the test cases generated. Furthermore, we can predict that this bug list is not conclusive, as some bugs triggered in the HAL layer were not perceivable without the use of sanitizers or other bug detection tools.

C. Kernel Coverage

To understand why DroidFuzz was able to find new bugs, we conduct an experiment that uses code coverage of the entire kernel and specific targeted kernel drivers as a proxy to identify the covered states of the drivers in the Android system. We then compare DroidFuzz's statistics to that of Syzkaller and Difuze. Syzkaller is also able to generate test cases for kernel drivers, and thus we can perceive the additional code coverage that jointly fuzzing the HAL layer brings. Difuze is an interface fuzzer that specifically generates *ioctl()* calls to device drivers, thus allowing us to compare

1) Comparison with Syzkaller: We compared DroidFuzz's kernel code coverage on all tested devices with that of Syzkaller's to understand the effect on kernel driver execution state exploration through joint testing the HAL layer. We run each experiment for 48 hours, and take the average coverage at each timestamp.



(a) Device A1 (b) Device A2

(c) Device B (d) Device C1

Fig. 4. Coverage comparison between DroidFuzz, Syzkaller, on select tested devices over 48 hours.

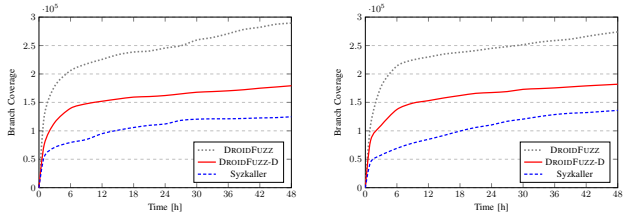
We show the relevant coverage for devices A1, A2, B, and C in Figure 4, whereas the the coverage statistics for devices D, E, and F follow this pattern and are omitted. As shown in the plot, DroidFuzz achieves better coverage than Syzkaller consistently, demonstrating the effectiveness of our approach towards testing Android's kernel and HAL layers together in triggering more execution states, particularly in the kernel, which also shows that system call fuzzers such as Syzkaller still fall short of providing realistic and meaningful payloads to the kernel drivers.

TABLE II
LIST OF ALL NEW BUGS FOUND BY DROIDFUZZ.

№	Device	Bug Info	Bug Type	Component
1	A1: Xiaomi Phone Dev Board	WARNING in rt1711_i2c_probe	Logic Error	Kernel Driver
2	A1: Xiaomi Phone Dev Board	Native crash in Graphics HAL (redacted)	Memory Related Bug	HAL
3	A1: Xiaomi Phone Dev Board	BUG: looking up invalid subclass: NUM	Logic Error	Kernel Subsystem
4	A1: Xiaomi Phone Dev Board	WARNING in tcpc (redacted)	Logic Error	Kernel Driver
5	A2: Xiaomi Tablet Dev Board	Infinite Loop in driver (redacted)	Logic Error	Kernel Driver
6	A2: Xiaomi Tablet Dev Board	Native crash in Media HAL (redacted)	Memory Related Bug	HAL
7	A2: Xiaomi Tablet Dev Board	KASAN: invalid-access in hci_read_supported_codecs	Memory Related Bug	Kernel Driver
8	B: Raspberry Pi 5	WARNING in l2cap_send_disconn_req	Logic Error	Kernel Subsystem
9	C1: Sunmi Commercial Tablet	Native crash in Camera HAL (redacted)	Memory Related Bug	HAL
10	C2: Sunmi Cashier Kiosk	WARNING in rate_control_rate_init	Logic Error	Kernel Driver
11	D: LubanCat 5	KASAN: slab-use-after-free Read in bt_accept_unlink	Memory Related Bug	Kernel Driver
12	E: AAEON UP Core Plus	WARNING in v4l_querycap	Logic Error	Kernel Driver

Thus, we conclude that jointly testing both the kernel and the HAL layer results in more state transitions within the kernel itself, increasing the probability of discovering new bugs.

2) Comparison with Difuze: Difuze utilizes a fuzzer called MangoFuzz (built upon Peach [8]), which feeds *ioctl()* interface invocations based on extracted interfacesto the kernel. As it is based on components designed for legacy kernels targeting select chipsets manufacturers, we only adapted it to test devices A1 and A2, where it succeeded in extracting 285 and 232 driver interfaces from the firmware of devices A1 and A2. To understand the effect and quality of HAL's *ioctl()* calls reflected upon kernel code coverage, we derive another variant DROIDFUZZ-D which limits the executor and HAL to only calling *ioctl()*s, whereas other requests will be blocked.



(a) Device A1 (b) Device A2

Fig. 5. Coverage comparison between DROIDFUZZ, Difuze, and DROIDFUZZ-D for over 48 hours.

The results are shown in Figure 5. As shown in the plot, DROIDFUZZ's coverage far outpaces that of Difuze's, which is expected due to the increase in interfaces tested. A more relevant comparison is DROIDFUZZ-D and Difuze, which shows that DROIDFUZZ-D leads Difuze's coverage by 34%. Our analysis shows that while the invoked *ioctl()* remains the same, sending the requests through fuzzing HAL produces significantly improved results compared to specification-based generation techniques.

Thus, we conclude that jointly testing the kernel and HAL produced additional benefits in that the HAL traps into the kernel with more realistic and meaningful payloads, giving the fuzzer an edge in discovering new states.

D. Ablation Tests

We also wish to understand the effect of each design component on the overall performance of DROIDFUZZ, and thus introduce two variations of DROIDFUZZ, DROIDFUZZ-NoRel, and DROIDFUZZ-NoHCov, which have modifications that disable kernel-user relational payload generation thus solely rely on randomized dependency generation, and removes HAL system call directional coverage from cross-boundary execution state feedback, respectively. We still use kernel coverage as a proxy to convey a statistic on the scope of execution states traversed. We use Syzkaller's statistics as a baseline to better understand the effects. The overall results are shown in Table III.

TABLE III
COVERAGE STATISTICS FOR ABLATION TESTS (48H)

Device	DROIDFUZZ	DF-NoRel	DF-NoHCov	Syzkaller
A1	289402	269281	274021	264920
A2	273930	250192	259402	247958
B	245930	210583	225126	220094
C1	269593	246930	240194	230664
C2	279305	239382	259492	220950
D	250295	237492	232910	220175
E	298593	247294	284629	257320

1) Effect of Kernel-User Relational Payload Generation: We find that the statistics for removing relational generation produces coverage consistently less than DROIDFUZZ. This is due to the generated test cases contain less meaningful and potentially incorrect usage of the system calls and APIs, and thus cannot cover as much code in the kernel. However, it still manages to frequently cover more code than Syzkaller, which indicates that HAL's use of the system call interface can produce more valid semantics.

2) Effect of Cross-Boundary Execution State Feedback: Removing cross-state feedback also results in lower coverage statistics than DROIDFUZZ. The reason is that the fuzzer can only refine the relations based on kernel code coverage alone, and thus produces less-than-optimal test cases. Even so, it still consistently performs better than Syzkaller, as joint fuzzing of both the kernel and HAL can trigger more execution states within these components.

Therefore, we conclude that both relational payload generation and joint state feedback contributes to DROIDFUZZ's overall effectiveness, allowing DROIDFUZZ generate more high-quality test cases by refining relations between system calls and HAL interfaces.

VI. CONCLUSION

DROIDFUZZ leverages pre-testing HAL driver probing, kernel-user relational payload generation, and cross-boundary execution state feedback to jointly test drivers in both Android's HAL layer and kernel. Our evaluations show that DROIDFUZZ finds 12 bugs in 7 embedded Android devices, and covers execution states more effectively than the state of the art.

VII. ACKNOWLEDGMENTS

This research is sponsored in part by the National Key Research and Development Project (No. 2022YFB3104000), and NSFC Program (No. U2441238, 62021002).

References

[1] Richard Barry et al. Freertos, 2003. https://www.freertos.org/.

[2] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2329–2344, New York, NY, USA, 2017. Association for Computing Machinery.

[3] Peng Chen and Hao Chen. Angora: Efficient Fuzzing by Principled Search. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 711–725, 2018.

[4] Peng Chen, Jianzhong Liu, and Hao Chen. Matryoshka: Fuzzing deeply nested branches. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 499–513, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Jaeseung Choi, Kangsu Kim, Daejin Lee, and Sang Kil Cha. NTFUZZ: Enabling type-aware kernel fuzzing on windows with static binary analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 1973–1989, 2021.

[6] Jake Corina, Aravind Machiry, Christopher Salls, Yan Shoshitaishvili, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. Difuze: Interface aware fuzzing for kernel drivers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2123–2138, New York, NY, USA, 2017. Association for Computing Machinery.

[7] Stéphane Duverger and Anaïs Gantet. Gustave: Fuzz it like it's app. *DMU Cyber Week*, 2021.

[8] Michael Eddington. Peach fuzzer. https://peachtech.gitlab.io/peach-fuzzer-community/.

[9] Google. Kernel address sanitizer. https://www.kernel.org/doc/html/latest/dev-tools/kasan.html.

[10] Emre Güler, Philipp Görz, Elia Geretto, Andrea Jemmett, Sebastian Österlund, Herbert Bos, Cristiano Giuffrida, and Thorsten Holz. Cupid: Automatic fuzzer selection for collaborative fuzzing. In *Annual Computer Security Applications Conference*, ACSAC '20, page 360–372, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Google Inc. Android debug bridge (adb). https://developer.android.com/tools/adb.

[12] Google Inc. Architecture overview. https://source.android.com/docs/core/architecture.

[13] Google Inc. Embedded. https://developer.android.com/reference/androidx/room/Embedded.

[14] Google Inc. Hardware abstraction layer (hal) overview. https://source.android.com/docs/core/architecture/hal.

[15] Silicon Lab. Ucos. https://www.silabs.com/developers/micrium.

[16] lcamtuf. American fuzzy lop, 2013. https://lcamtuf.coredump.cx/afl/.

[17] Jianzhong Liu, Yuheng Shen, Yiru Xu, Hao Sun, and Yu Jiang. Horus: Accelerating kernel fuzzing through efficient host-vm memory access procedures. *ACM Trans. Softw. Eng. Methodol.*, 33(1), November 2023.

[18] Jianzhong Liu, Yuheng Shen, Yiru Xu, Hao Sun, Heyuan Shi, and Yu Jiang. Effectively sanitizing embedded operating systems. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery.

[19] Zhengxiong Luo, Junze Yu, Feilong Zuo, Jianzhong Liu, Yu Jiang, Ting Chen, Abhik Roychoudhury, and Jiaguang Sun. Bleem: packet sequence oriented fuzzing for protocol implementations. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA, 2023. USENIX Association.

[20] Dominik Maier and Fabian Toepfer. Bsod: Binary-only scalable fuzzing of device drivers. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '21, page 48–61, New York, NY, USA, 2021. Association for Computing Machinery.

[21] Sanoop Mallissery and Yu-Sung Wu. Demystify the fuzzing methods: A comprehensive survey. *ACM Comput. Surv.*, 56(3), October 2023.

[22] Valentin JM Manes, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J Schwartz, and Maverick Woo. Fuzzing: Art, science, and engineering. *arXiv preprint arXiv:1812.00140*, 2018.

[23] Lucas McDonald, Muhammad Ijaz Ul Haq, and Ashley Barkworth. Survey of software fuzzing techniques.

[24] Anas Nashif. Zephyr is a new generation, scalable, optimized, secure RTOS, 2016. https://github.com/zephyrproject-rtos/zephyr.

[25] Henry Neugass, G Espin, Hidefume Nunoe, Ralph Thomas, and David Wilner. Vxworks: an interactive development environment and real-time kernel for gmicro. In *Eighth TRON Project Symposium*, pages 196–197. IEEE Computer Society, 1991.

[26] National Institute of Standards and Technology. National vulnerability database: Cve-2021-0673 detail. https://nvd.nist.gov/vuln/detail/CVE-2021-0673.

[27] The Linux Kernel Organization. The linux kernel archives. https://kernel.org.

[28] Shankara Pailoor, Andrew Aday, and Suman Jana. MoonShine: Optimizing OS Fuzzer Seed Selection with Trace Distillation. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 729–743, Baltimore, MD, August 2018. USENIX Association.

[29] Shiyan Peng, Yuan Zhang, Jiarun Dai, Yue Gu, Zhuoxiang Shen, Jingcheng Liu, Lin Wang, Yong Chen, Yu Qin, Lei Ai, Xianfeng Lu, and Min Yang. Applying fuzz driver generation to native c/c++ libraries of oem android framework: Obstacles and solutions. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 2035–2040, New York, NY, USA, 2024. Association for Computing Machinery.

[30] LLVM Project. Llvm sanitizercoverage. https://clang.llvm.org/docs/SanitizerCoverage.htmll.

[31] Alexandre Rebert, Sang Kil Cha, Thanassis Avgerinos, Jonathan Foote, David Warren, Gustavo Grieco, and David Brumley. Optimizing Seed Selection for Fuzzing. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 861–875, San Diego, CA, August 2014. USENIX Association.

[32] Teresa Reidt. Several reasons for choosing embedded android. https://emteria.com/learn/embedded-operating-system.

[33] Sergej Schumilo, Cornelius Aschermann, Robert Gawlik, Sebastian Schinzel, and Thorsten Holz. kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 167–182, Vancouver, BC, August 2017. USENIX Association.

[34] Yuheng Shen, Yiru Xu, Hao Sun, Jianzhong Liu, Zichen Xu, Aiguo Cui, Heyuan Shi, and Yu Jiang. Tardis: Coverage-guided embedded operating system fuzzing. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 41(11):4563–4574, nov 2022.

[35] SimonKagstrom. Kcov. https://github.com/SimonKagstrom/kcov.

[36] Ting Su, Yichen Yan, Jue Wang, Jingling Sun, Yiheng Xiong, Geguang Pu, Ke Wang, and Zhendong Su. Fully automated functional fuzzing of android apps for detecting non-crashing logic bugs. *Proc. ACM Program. Lang.*, 5(OOPSLA), October 2021.

[37] Hao Sun, Yuheng Shen, Cong Wang, Jianzhong Liu, Yu Jiang, Ting Chen, and Aiguo Cui. *HEALER: Relation Learning Guided Kernel Fuzzing*, page 344–358. Association for Computing Machinery, New York, NY, USA, 2021.

[38] Jingling Sun, Ting Su, Jiayi Jiang, Jue Wang, Geguang Pu, and Zhendong Su. Property-based fuzzing for finding data manipulation errors in android apps. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 1088–1100, New York, NY, USA, 2023. Association for Computing Machinery.

[39] Dmitry Vyukov and Andrey Konovalov. Syzkaller: an unsupervised coverage-guided kernel fuzzer, 2015. https://github.com/google/syzkaller.

[40] Dmitry Vyukov and Andrey Konovalov. Syzlang: System Call Description Language, 2015. https://github.com/google/syzkaller/blob/master/docs/syscall_descriptions_syntax.md.

[41] Bernard Xiong and Man Jianting. RT-Thread is an open source IoT operating system., 2007. https://github.com/RT-Thread/rt-thread.

[42] Hao Xiong, Qinming Dai, Rui Chang, Mingran Qiu, Renxiang Wang, Wenbo Shen, and Yajin Zhou. Atlas: Automating cross-language fuzzing on android closed-source libraries. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 350–362, New York, NY, USA, 2024. Association for Computing Machinery.

[43] Karim Yaghmour. *Embedded android: Porting, extending, and customizing*. O'Reilly, 2017.

Simulation-based Parallel Sweeping: A New Perspective on Combinational Equivalence Checking

Tianji Liu
CSE Department, CUHK
tjliu@cse.cuhk.edu.hk

Evangeline F.Y. Young
CSE Department, CUHK
fyyoung@cse.cuhk.edu.hk

***Abstract*—Combinational equivalence checking (CEC) is a fundamental task in the realization of digital designs which is unlikely to have universally efficient algorithms due to its co-NP-completeness. Recent researches of CEC have been focusing on SAT sweeping. This paper provides a new perspective other than SAT for tackling CEC, namely exhaustive simulation, and presents a simulation-based CEC engine constructed with fast GPU-parallel algorithms. The proposed engine can solve 4 out of the 9 large cases in the experiments on its own, with up to 88.11× speed-up compared with the checker in ABC. Moreover, a combination of the proposed engine with the ABC checker achieves averaged accelerations of 4.89× and 4.88× over the standalone ABC checker and a commercial checker, respectively.**

I. Introduction

Combinational equivalence checking (CEC) is one of the most important and fundamental problems in the realization flow of modern digital designs, in which the objective is to prove the functional equivalence of two netlists that implement the same combinational circuit differently. Besides logic verification [1], CEC approaches also have a wide range of applications in the areas including logic synthesis [2], [3] and functional ECO [4].

Many tasks in EDA can be formulated as optimization problems that aim to reduce certain costs, e.g., area and delay reduction in logic optimization and technology mapping, and wirelength reduction in placement and routing, which enables the use of approximate and heuristic approaches that run in polynomial time, making them scalable to large designs. In contrast, CEC requires an exact answer of whether two circuits are equivalent, i.e., the problem needs to be tackled directly without any approximation, and it is thus much more challenging than the aforementioned tasks considering the complexity of CEC as well as the increasing scale of modern designs.

Since CEC is co-NP-complete, efficient verification of all kinds of designs using currently developed approaches is essentially infeasible. Binary decision diagram (BDD) [5], [6] had been a common method for CEC in the early years, but was gradually replaced by Boolean satisfiability (SAT) solving [7], [8] due to the excessive memory consumption of BDD which impedes its application to large designs. Today, SAT has become the de facto core technique in CEC because of the noticeable performance improvement in SAT solving over the past two decades [9]–[11]. Nevertheless, it has been argued that SAT-based methods are not as performant as other approaches for certain types of designs, e.g., computer algebra methods for arithmetic circuits [12]. This implies that the demand for novel and fast CEC approaches will never diminish, as long as new and larger sized designs continue to emerge.

In this paper, we present a new perspective on tackling CEC, namely utilizing *exhaustive simulation* for proving functional equivalences. A key advantage of exhaustive simulation over SAT is that it is friendly to parallelization, and we will demonstrate in this paper that exhaustive simulation can be performed efficiently on GPUs, which are powerful massively parallel processors that have shown promising results in logic synthesis applications [13], [14]. When proving difficult cases, however, parallel exhaustive simulation may still be intractable due to its intrinsic exponential complexity. To address this issue, we introduce a local function checking scheme for restricting the computational effort of checking difficult cases, and the effectiveness of checking can still be ensured by delicately designed heuristics. We propose and implement fast GPU-parallel algorithms for exhaustive simulation, local function checking and other auxiliary procedures for CEC, which are combined together as a *simulation-based CEC engine* with high overall performance.

The simulation-based CEC engine is capable of independently proving 4 out of the 9 large circuits in the experiments, with up to 88.11× acceleration compared to the SAT-based equivalence checker in ABC [15]. When combined with the ABC checker for handling the undecided cases, the integrated approach obtains 4.89× and 4.88× speed-up on average compared to the standalone ABC checker and a commercial verification tool, respectively.

This research was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK14210923).

II. Preliminaries

A. Background

A *Boolean network* is a directed acyclic graph (DAG) that models a combinational circuit where nodes stand for logic operations and edges represent wires between the nodes. The *primary inputs* (PIs) and *primary outputs* (POs) of a Boolean network are the sources and the sinks of the DAG. The direct predecessors (resp. direct successors) of a node are called the *fanins* (resp. *fanouts*) of the node. The predecessors (resp. successors) of a node are called the *transitive fanins* (TFIs) (resp. *transitive fanouts*, TFOs) of the node. The *(structural) support* of a node denotes the set of PIs that are TFIs of the node. The function of a node expressed in terms of its support is the *global function* of the node. The *level of a node* is the length of a longest path from any PI to the node, which can be computed recursively as the maximum level of the node's fanins plus one. The *level of the network* is the largest level of its POs.

A *cut* c_n of a node n is a set of nodes such that any path from a PI to n passes at least one node in the set, and n is the *root* of c_n. The *trivial cut* of n is defined as $\{n\}$. The function of a node in terms of its cut nodes is the *local function* of the node. The *logic cone* of a node n (associated with a cut c_n) contains the intersection of the TFIs of n with the TFOs of c_n, and n itself.

An *And-Inverter Graph* (AIG) [16] is a Boolean network in which nodes are two-input AND gates and signals (edges) can be optionally inverted. Because of its compactness and flexibility, AIG has become one of the most common circuit representations in modern logic synthesis and verification tools such as ABC [15]. In this work, we

also focus on the verification of AIGs but the proposed methods can be applied to any kind of circuit representation.

The *truth table* of a k-input Boolean function is a bit string $b_{l-1} \dots b_0$ of length $l = 2^k$, in which a bit $b_i \in \{0, 1\}$ is the function value under the assignment of the k inputs $(a_0, \dots, a_{k-1})$ that satisfies $2^{k-1} \cdot a_{k-1} + \dots + 2^0 \cdot a_0 = i$. Essentially, a truth table consists of the function values under all possible input assignments[1]. We denote the truth tables of projection functions $f_i(x_0, \dots, x_{k-1}) = x_i$ as *projection truth tables*, which serve as the truth tables of inputs when computing the truth table of a node in a network. For instance, when $k = 3$, the projection truth tables of f_0, f_1, f_2 are 10101010, 11001100, 11110000.

The *(internal) satisfiability don't cares* (SDCs) of a node in a Boolean network are the patterns at a cut of the node that will never occur. For example, in the network with nodes $n_1 = x + y$, $n_2 = yz$ and $n_3 = n_1 n_2$, n_3 has SDCs of $\{(n_1 = 0, n_2 = 1)\}$. If a node n has SDCs, the local function values of n under the SDC assignments can be changed without affecting the global function of n. It is believed that SDCs are mainly due to reconvergent paths (different paths with the same start and end) in the TFI structure of the cut [17], [18].

B. Sweeping

In CEC, a *miter* [19] circuit is constructed by sharing the corresponding PI pairs of the two circuits being compared, and combining the corresponding PO pairs using XOR gates which are treated as the POs of the miter. Then, the problem of checking whether the two circuits are equivalent becomes checking whether all the POs of the miter are constant zeros.

A common and effective framework for CEC is *sweeping* [6], [16]. The basic idea of sweeping is to identify equivalent pairs of internal nodes in the miter and merge their logic, so the miter can be gradually reduced, making it easier to prove the POs of the miter. A fast *(partial) simulation* procedure [16] is performed in the beginning of sweeping, which computes the partial truth tables of all the nodes using simulation patterns assigned to the PIs (usually randomly generated), and clusters the nodes with the same partial truth table into an *equivalence class* (EC). This reduces the number of candidate node pairs that need formal checking, since any pair of equivalent nodes must reside in the same class. A usual way to generate candidate pairs is to take the *representative* of a class (defined as the node with the minimum id in the class) and match it with the non-representatives in the class, so a class of N nodes produces $N - 1$ pairs. The equivalence checking of candidate pairs is performed using a formal method, e.g., BDD [6] or SAT [8].

Recent researches mainly focus on improving the SAT sweeping framework. In [3], [20], [21], different methods are proposed for generating high-quality simulation patterns to reduce SAT calls. [21], [22] introduce a customized SAT solver with engineering efforts tailored for SAT sweeping. There have also been works on accelerating SAT sweeping via parallelization. [23] describes a method for concurrently executing SAT checks for multiple candidate node pairs. [24] develops a hybrid CPU-GPU framework in which GPU performs simulation and local matching of candidate pairs, and the pairs undetermined in local matching are further checked by SAT on CPU. The proposed CEC engine drastically differs from these methods in that it relies on exhaustive simulation rather than SAT for proving equivalent nodes.

[1]We also use the term *patterns* for input assignments in this paper.

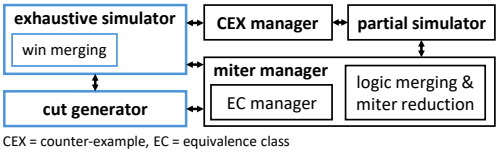


Fig. 1. The structure of the simulation-based CEC engine. The core modules are highlighted in blue. Arrows indicate interactions between modules.

III. Simulation-based CEC Engine

This section introduces the simulation-based CEC engine. An overview of the CEC engine is first provided in Section III-A, followed by two subsections elaborating on the exhaustive simulator and local function checking, which are core components of the engine. Finally, the overall flow of the engine is presented in Section III-D.

A. Overview of Simulation-based CEC Engine

The simulation-based CEC engine consists of five modules, as depicted in Figure 1, in which the exhaustive simulator and cut generator are the core of the engine. The exhaustive simulator serves as the prover in the engine that checks whether a candidate pair of nodes is equivalent by comparing all the possible values in their entire truth tables. An input pattern that disproves the candidate pair, i.e., yielding different function values at the two nodes, will be collected as a *counter-example* (CEX). The exhaustive simulator is designed to be efficient: it is capable of checking a batch of candidate pairs using a highly parallel algorithm, with a window merging procedure for reducing the total simulation effort.

The cut generator plays a crucial role in checking candidate pairs of nodes with large support sizes. Due to exponential complexity, it is infeasible to exhaustively simulate the global functions of such nodes. Instead, the cut generator produces multiple common cuts for each pair of nodes, and the exhaustive simulator evaluates the corresponding local functions of the candidate pairs. By limiting the sizes of these cuts, the simulation effort can be successfully controlled. A detailed introduction to cut generation will be given in Section III-C1.

The remaining modules have their counterparts in the traditional SAT sweeping framework which are conceptually similar to them. The miter manager maintains the miter in AIG data structures and can reduce the miter by merging proved pairs of equivalent nodes. It is also equipped with an EC manager which is responsible for maintaining the equivalence class information and generating candidate pairs to be proved. The partial simulator performs simulation of random patterns for initializing the ECs, as well as CEX patterns for splitting the class of a disproved pair (and potentially other classes) into smaller ones.

All the modules and subroutines in the simulation-based CEC engine are GPU-based except window merging which is partially implemented on CPU, making the engine highly efficient.

B. Exhaustive Simulator

1) Equivalence Checking via Exhaustive Simulation: We start by introducing how to check the equivalence of a pair of nodes via exhaustive simulation. The basic idea, as mentioned in Section III-A, is to compare the entire truth tables of the two nodes. However, the comparison is only meaningful if the input variables (including their ordering) of the two truth tables are exactly the same. Consider the function $xy' + xy'z$ as an example. Its truth tables are 00100010 and 01000100 for the input orders (x, y, z) and (y, x, z) respectively.

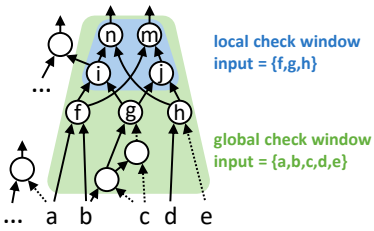


Fig. 2. Illustration of windows in global and local function checking of two nodes n and m. Note, their local functions in terms of $\{f, g, h\}$ are equivalent.

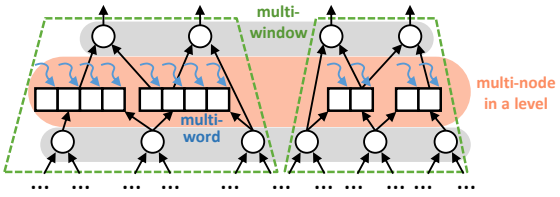


Fig. 3. Illustration of the three dimensions of parallelism.

Also, note that $xy' + xy'z = xy'$, and the truth table of xy' with input order (x, y) is 0010, indicating that the truth table of equivalent functions may not be the same under different variables and orderings.

The proposed exhaustive simulator handles this issue as follows. In the case of simulating the global functions of the two nodes being checked, if their support sets are not identical, the union of their supports are taken as the input nodes of the truth table with the order of increasing node ids. Since it is possible that a node does not functionally depends on one or more of its support nodes (e.g., $xy' + xy'z$ does not functionally depend on z), two nodes with nonidentical supports can be equivalent and still need to be checked. In the case of checking the local functions of a node pair, the inputs are simply the nodes in the corresponding common cut of the pair provided by the cut generator, as introduced in Section III-A.

The truth tables of two nodes n, m being checked are computed by propagating truth tables along the intermediate nodes that drive n and m, starting from the input nodes. We denote the collection of such intermediate nodes as the *simulation window* (or simply *window*) of n and m, and the two nodes are called the *roots* of the window. Figure 2 provides an illustration of windows in the scenario of simulating the global and local functions of two nodes respectively. Formally, a window contains the intersection of the TFIs of the roots with the TFOs of the inputs, as well as the roots.

The computation procedure begins with assigning projection truth tables to the input nodes. Then, the truth table of a node in the window can be derived by processing the truth tables of its fanins according to the node's operation. For instance, in the AIG shown in Figure 2, the truth table of the node $h = de'$ can be computed as $TT(h) = TT(d)$ & $!TT(e)$ where &, ! stand for bitwise AND, NOT respectively. The computations for all the nodes in the window should be scheduled in a topological order to ensure correctness. In our exhaustive simulator, this is performed in a level-wise parallel way, which is detailed in the next subsection.

2) Parallel Simulation: The exhaustive simulator is equipped with three dimensions of parallelism, as illustrated in Figure 3. The most fine-grained parallelism happens in the truth table computation of a single node, in which each word (a 32-bit or 64-bit element) in the bit string of the truth table is computed by a separate thread. Such parallel processing is particularly efficient on GPUs because of coalesced memory accesses [25] which minimizes bandwidth usage.

Algorithm 1 Parallel Exhaustive Simulation

Input: Miter, set of pairs p and windows w, available memory in words M

1: $N \leftarrow \sum_{w_i \in w}(|w_i| + |inputs(w_i)|)$ ▷ number of nodes and inputs
2: $E \leftarrow \max_{e \in \mathbb{N}} 2^e$ s.t. $2^e \cdot N \leq M$ ▷ entry size of simulation table
3: $simt \leftarrow$ simulation table containing $E \cdot N$ words
4: $r_m \leftarrow (\max_{w_i \in w} tt_len(w_i))/E$ ▷ number of rounds
5: **for each** $r = 0, \ldots, r_m - 1$ **do** ▷ simulate range $[rE, (r+1)E)$
6: $w' \leftarrow \{w_i \in w : tt_len(w_i) \geq (r+1)E\}$
7: ▷ w' is the set of windows that need simulation in round r
8: $l_m \leftarrow \max_{w_i \in w'} \max_{n \in w_i} level(n)$ ▷ max level in round r
9: **parallel** write proj. TT segments into $simt$ for inputs to all $w_i \in w'$
10: **for each** $l = 1, \ldots, l_m$ **do**
11: **parallel** simulate all nodes in w' with level l, and update $simt$
12: **parallel for each** $p_i = (n_i, m_i) \in p$ **do**
13: **if** the TT segments of n_i and m_i in $simt$ mismatch **then**
14: Mark p_i as nonequivalent, and optionally collect CEX

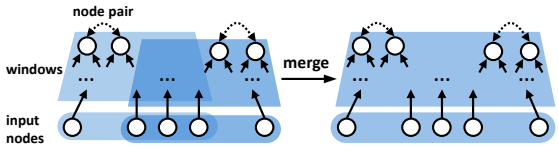


Fig. 4. Illustration of window merging.

The second dimension of parallelism resides in the truth table computation of the nodes in a window, following the level-wise parallel [14] fashion. Specifically, nodes with the same topological level are grouped into a batch and concurrently processed since there is no intra-batch dependency between nodes, and different batches are processed sequentially in the order of increasing topological level. The topological level of a node is similar to the node's level defined in Section II-A, and the difference is that the input nodes to the window are assigned with zero topological levels.

The third dimension of parallelism is to simulate multiple windows concurrently. If the number of inputs to a window is small, the truth tables to be computed are short in length, and the window is likely to be small sized. In this case, the degree of parallelism provided by the other two dimensions is limited, so it is essential to include this dimension to fully exploit the computational power of GPUs.

Algorithm 1 shows the high-level pseudocode of parallel exhaustive simulation. Given a batch of node pairs and their windows, a simulation table is allocated for storing the simulation results of all the nodes in the windows and the patterns at the window inputs (lines 1-3). Since the truth table of a pair can be long and there can be many pairs in the batch, it is impracticable for the simulation table to record the entire truth tables. Instead, each node or input is assigned with an entry of $E = 2^e$ words, where E is decided on-the-fly to be the maximum feasible value such that the simulation table fits in a provided size of memory (line 2). To ensure exhaustive simulation, the main procedure is formulated to be multi-rounded: in round r, the truth table words with indices in the range of $[rE, (r+1)E)$ are simulated with the three dimensions of parallelism (lines 6-11). The simulation results of all the node pairs are compared at the end of each round. If a mismatch is detected at a pair, this pair is marked as nonequivalent and the corresponding input pattern can be collected as a CEX (lines 12-14).

3) Reducing Simulation Effort via Window Merging: In exhaustive simulation, windows may overlap with each other and a node appearing in multiple windows must be simulated separately in each window due to nonidentical input nodes, which potentially affects the efficiency. A possible method for addressing this issue is to merge

multiple windows with high overlaps into one, and simulate all the corresponding pairs using the single merged window, as depicted in Figure 4. In this way, the total number of nodes that need simulation as well as the number of windows can be reduced. On the other hand, the number of inputs may increase after merging a window which leads to longer truth tables, and it should be carefully controlled to ensure that the overall effect of window merging is beneficial.

Our algorithm of window merging is performed in the beginning of exhaustive simulation and it employs a fast heuristic that works well in practice. First, we sort the batch of windows in the lexicographical order of their input nodes. Since the input set to a window is ordered in ids (Section III-B1), the windows with similar input sets tend to be closer to each other after sorting. The resulting windows are then generated by maximally merging consecutive windows in the order while keeping the input sizes of the merged windows under a threshold k_s. As an example, for the five windows with inputs $\{a,b\},\{a,b\},\{a,b,c\},\{a,e\},\{a,f\}$ and $k_s=3$, the first three and the last two are merged together respectively. We remark that better merging results may be obtained using a more dedicated approach (e.g., clustering-based) but it may induce a large overhead. Window merging is only enabled for global function checking, because it is generally not beneficial for local function due to small window sizes.

C. Local Function Checking

1) Cut Enumeration and Selection Criteria for Effective Checking: Local function checking is a key component in the simulation-based CEC engine, which is indispensable for proving the equivalences of node pairs with large support sizes that cannot be exhaustively simulated in terms of global functions. If the local functions of two nodes in terms of a common cut are identical, the two nodes are proved equivalent. As an example, the equivalence of the node pair shown in Figure 2 can be proved with the local functions in terms of the cut $\{f,g,h\}$, which reduces the computation from simulating 2^5 patterns per node in global function checking to 2^3 patterns.

However, two equivalent nodes may also have nonidentical local functions in terms of a common cut, e.g., those in terms of the cut $\{f,h,i,j\}$ in Figure 2. This is due to SDCs at the cut, and the equivalence is not affected as long as all the patterns yielding different local function values are SDCs. In the context of local function checking, the equivalence of two nodes is inconclusive if nonidentical local functions are encountered. To enhance the chance of successful checking, two strategies can be applied: increasing the quantity, and improving the quality of the common cuts used in checking.

To increase the quantity of cuts, we design a cut generator for the CEC engine that is capable of producing multiple common cuts for each pair using a cut enumeration-based [26] framework. For each AIG node n in the miter, the cut generator enumerates a set of candidate cuts $E(n)$ with maximum cut size k_l by

$$E(n) = \{u \cup v : u \in P(n_0) \cup \{\{n_0\}\}, \\ v \in P(n_1) \cup \{\{n_1\}\}, |u \cup v| \leq k_l\}, \quad (1)$$

where n_0, n_1 are the two fanins of n, and $P(\cdot)$ denotes the set of *priority cuts* [27] of the node that contains its best C candidate cuts selected using certain criteria. k_l and C are parameters that control the computational effort of local function checking. The common cuts of a pair can then be computed by Equation (1) with n_0 and n_1 replaced by the pair of nodes and without including their trivial cuts. The main advantages of cut enumeration are that it can efficiently produce multiple cuts for all the nodes (and common cuts for all the pairs) in a topological order traversal, and that it can adopt customized criteria for selecting a set of priority cuts.

TABLE I
CUT SELECTION CRITERIA IN DIFFERENT PASSES

Pass	Main Metric	Tie-breaker Metric 1	Tie-breaker Metric 2
1	fanout	cut size	small level
2	small level	cut size	fanout
3	large level	cut size	fanout

To obtain high-quality cuts, the cut selection criteria is formulated delicately in which the following metrics are involved:

- average number of fanouts of cut nodes: large fanout is used as a cutpoint selection heuristic in [16], so a high value is preferred;
- cut size: a small value is preferred for preventing oversized cuts during cut enumeration, and encouraging the inclusion of more reconvergent structures in the logic cone so that SDCs can be reduced, which enables higher chances of proving equivalences;
- average level of cut nodes: a small value is preferred for including more logic in the cone and reducing SDCs, but a high level may also be beneficial since it leads to smaller cuts, and sometimes a high-level cut is sufficient to prove a pair if one node is generated by the local restructuring of the other (e.g., Figure 2).

In practice, it is difficult to obtain cuts that are good in terms of all the metrics. We tackle this issue by applying three passes of cut generation and checking. In different passes, the metrics are considered with different priorities to increase the diversity of the generated cuts, as specified in Table I. For instance, in the first pass, cuts with larger fanouts are selected; if a tie happens, cuts of smaller size are preferred; if there is still a tie, cuts with smaller levels are used.

A problem of cut enumeration is that it lacks the view of common cuts when generating cuts for single nodes, which subsequently causes many oversized ($> k_l$) common cuts that cannot be used in local function checking. This problem is resolved by generating similar cuts for a pair of nodes in cut enumeration. Specifically, we define a similarity metric between a cut c and a set of priority cuts P as $s(c,P)=\sum_{c'\in P}|c\cap c'|/|c\cup c'|$. When enumerating the cuts of a non-representative node n, the similarity between a cut of n and the priority cut set of the corresponding representative node in the class is computed (since n and the representative form a pair), and cuts with larger similarities are preferred. In case of a tie, the criteria in Table I are used. On the other hand, the representative nodes adopt Table I for cut selection as introduced in the previous paragraph. In this way, the priority cuts of a pair of nodes tend to have high overlaps to each other, thus increasing the number of usable common cuts in checking.

2) Flow of Cut Generation and Checking Pass: Cut generation and checking are performed in level-wise parallel, where the level is specially formulated to consider the dependencies between representatives and non-representatives, since the priority cuts of a representative node should be computed before any other non-representatives in its class. We denote such levels as *enumeration levels*, defined by

$$el(n) = \begin{cases} 0, & n \text{ is PI}, \\ 1+\max\{el(n_0), el(n_1)\}, & n \text{ is not PI and } n \text{ is repr.}, \\ 1+\max\{el(n_0), el(n_1), el(repr(n))\}, & \text{otherwise}, \end{cases} \quad (2)$$

where $repr(n)$ signifies the corresponding representative of a non-representative node n, and n_0, n_1 are the two fanins of n.

Algorithm 2 describes the flow of a cut generation and checking pass. It starts with computing the enumeration levels of all the nodes (line 2) and assigning trivial cuts to the PIs as their priority cuts (lines 4-5). The priority cuts of internal nodes are then computed level by

Algorithm 2 Cut Generation and Checking Pass

Input: Miter $miter$

1: $buf \leftarrow \emptyset$ ▷ allocate common cut buffer
2: Compute enumeration levels $el(\cdot)$ of all nodes by Equation (2)
3: $l_m \leftarrow \max_{n \in miter} el(n)$
4: **parallel for each** PI n of $miter$ **do**
5: $P(n) \leftarrow \{\{n\}\}$
6: **for each** $l = 1, \ldots, l_m$ **do**
7: $enum \leftarrow \{n \in miter : el(n) = l\}$
8: ▷ $enum$ is the set of nodes whose cuts are to be computed at level l
9: **parallel for each** $n \in enum$ **do**
10: **parallel** compute $P(n)$ by cut enum. & selection (Section III-C1)
11: $p \leftarrow \{(repr(n), n) : n \in enum, n$ is a non-representative$\}$
12: **parallel** generate valid common cuts of pairs in p as cmn
13: **if** $|cmn| > capacity(buf) - |buf|$ **then** ▷ insufficient space
14: Local function checking using cuts in buf by Algorithm 1
15: $buf \leftarrow \emptyset$ ▷ clear the buffer
16: $buf \leftarrow buf \cup cmn$ ▷ insert in **parallel**
17: **if** $|buf| > 0$ **then** ▷ check the final batch
18: Local function checking using cuts in buf by Algorithm 1

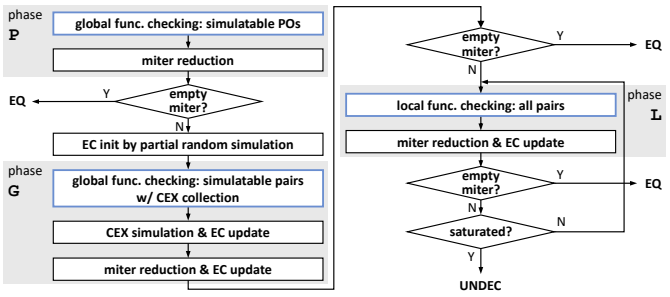


Fig. 5. Flow of simulation-based CEC engine.

level, following the criteria introduced in Section III-C1 (lines 6-10). In particular, cut enumeration and priority cut selection for a single node are performed in fine-grained parallelism (line 10) using the method of [28], where each cut in $E(n)$ is generated using an individual thread, and the selection of the best C cuts that constitutes $P(n)$ are jointly performed by the group of threads in parallel. We skip further details due to the space limit.

Local function checking is interleaved with cut enumeration. At each level, the common cuts of the node pairs whose priority cuts are just computed are generated and inserted into a constant sized buffer (lines 11-12 and 16). When the buffer does not have sufficient space to accept a new batch, an exhaustive simulation is invoked to check all the local functions in terms of the cuts in the buffer and update the equivalence statuses of the proved pairs (lines 13-15). The use of the common cut buffer reduces memory consumption, compared to the method of gathering all the common cuts and performing a single batch of checking in the end.

D. Overall Flow of Simulation-based CEC Engine

Figure 5 depicts the overall flow of the CEC engine which consists of three types of phases: a PO checking phase (`P`), a global function checking phase (`G`), and repeated local function checking phases (`L`).

In the PO checking phase, the CEC engine attempts to prove all or a subset of *simulatable* miter POs in terms of global functions, that is, the primary outputs whose checking can be carried out within a computational budget. The motivation for performing PO checking in the beginning is to maximally reduce the miter so the efforts of checking internal pairs in the removed part of the miter can be saved. A simulatable PO is defined as follows with two parameters k_P, k_p ($k_P > k_p$) that control the budget of checking: if the support sizes of all the POs are no larger than k_P, then all the POs are simulatable; otherwise, a PO is simulatable if its support size is no larger than k_p. The design of two thresholds rather than one is to encourage the CEC engine to prove the miter in a one-shot PO checking if possible.

After initializing equivalence classes by partial random simulation, the global function checking phase verifies simulatable node pairs with support sizes no larger than a threshold k_g, along with collecting CEXs to disprove the nonequivalent pairs and refine the classes.

If the miter cannot be proved by PO and global function checking, the local function checking phases will be conducted repeatedly, where each phase contains three passes of cut generation and checking that attempt to prove all the node pairs as introduced in Section III-C. Since the structure of the miter is modified after reduction at the end of a phase, the cuts generated in the subsequent local checking phase can be different, which provides new chances for verifying the equivalent pairs that failed to be proved in the previous phases. If the miter cannot be further reduced after repeated local function checking, the equivalence of the two circuits is undecided and the reduced miter can be passed to another CEC engine for further proving.

We note that the partial simulator, miter reduction, the EC and CEX manager are all implemented by GPU-parallel algorithms, which ensures the high overall efficiency of the CEC engine.

IV. Experimental Results

The simulation-based CEC engine is implemented in 8,000 lines of CUDA/C++ code on top of the GPU-based logic synthesis tool CULS[2]. We denote it as the "GPU engine" in this section for brevity. The experiments are performed on a Linux machine with Intel Xeon Gold 6326 CPU and NVIDIA RTX A6000 GPU with 48 GB DRAM.

The testcases are selected from the EPFL Combinational Benchmark Suite [29] and the IWLS 2005 Benchmarks [30], covering arithmetic and control designs. To demonstrate the effectiveness of our parallel CEC approach, we enlarge the designs by applying multiple times of the ABC command `double`, which is a common method used in the works on parallel logic synthesis and verification [14], [23], [31]. The two circuits compared by CEC are the original and optimized versions of a benchmark, where the optimized one is generated by executing ABC `resyn2`, an AIG optimization script consisting of several passes of balancing, rewriting and refactoring [32]. Table II shows the statistics of the benchmarks and miters constructed by the pairs of circuits being compared.

In the experiments, we use the following parameter values for the GPU engine: $k_P = 32$, $k_p = k_g = 16$, $k_l = 8$, $C = 8$. The maximum support size of a window after window merging k_s is set as the support size threshold (k_P, k_p, or k_g) of the phase. For instance, in the PO checking phase, $k_s = k_p = 16$ if not all of the POs are simulatable. We adopt the ABC combinational equivalence checker (command `&cec -C 100000`) for further proving a reduced miter if it is undecided (i.e., not empty) after running the GPU engine. A large value is assigned for the maximum number of conflicts during a SAT call (`-C`) since most of the easy-to-prove pairs have already been handled by the GPU engine.

A. Runtime Comparison with ABC and Commercial Checker

We evaluate our approach by comparing it with the ABC checker (command `&cec`) and a commercial verification tool Cadence Conformal LEC (version 19.20). ABC `&cec` is a single-threaded yet high-performance checker based on SAT sweeping. There is no

[2]https://github.com/cuhk-eda/CULS

TABLE II
BENCHMARK STATISTICS AND RUNTIME COMPARISON OF OUR METHOD WITH ABC AND CONFORMAL LEC

Benchmarks	Statistics				ABC `&cec` Runtime (s)	Cfm (16 CPUs) Runtime (s)	Ours (GPU+ABC)				Speed-up	
	#PIs*	#POs*	#Nodes†	Levels†			GPU (s)	Reduced (%)	ABC (s)	Total (s)	vs. ABC	vs. Cfm
hyp_7xd	32768	16384	45881216	24801	7859.26	406002	4616.56	40.2	418.48	5035.04	1.56×	80.64×
log2_10xd	32768	32768	62072832	444	>4 months‡	118392	119633.18	100.0	-	119633.18	88.11×	0.99×
multiplier_10xd	131072	131072	52600832	274	2370.52	3213	159.54	100.0	-	159.54	14.86×	20.14×
sqrt_10xd	131072	65536	44978176	5058	20640.56	30605	52.29	0.7	20623.24	20675.53	1.00×	1.48×
square_10xd	65536	131072	33442816	250	1021.40	2710	144.35	100.0	-	144.35	7.08×	18.77×
voter_10xd	1025024	1024	21862400	70	62610.44	1166	54.20	43.5	35611.63	35665.83	1.76×	0.03×
sin_10xd	24576	25600	10689536	225	2499.28	2081	78.88	100.0	-	78.88	31.68×	26.38×
ac97_ctrl_10xd	2307072	2299904	22685696	12	248.57	1563	97.51	98.9	22.43	119.94	2.07×	13.03×
vga_lcd_5xd	549248	549856	6337536	24	95.82	317	18.51	20.1	81.95	100.46	0.95×	3.16×
Geomean											4.89×	4.88×

* Original/optimized circuit statistics. † AIG miter statistics (excluding contributions of XORs). ‡ Timeout after 122 days. Use 122 days when computing speed-up.
"_$n$xd" in a benchmark name stands for enlarging the benchmark by executing ABC `double` n times.

publicly available parallel checker to our knowledge, so we compare to Conformal which supports parallel checking with up to 16 CPU threads. Although the implementation details of commercial checkers are unknown, it is believed that they perform equivalence checking using a combination of engines [33], and a possible way for multi-threading is to run different engines simultaneously and early stop when an engine finishes.

Table II shows the results of our approach and the runtimes of ABC checker and Conformal. The subcolumn "Reduced (%)" reports the percentages of reduction in miter sizes after executing the GPU engine. It is worth noting that the GPU engine can fully prove (i.e., 100% reduction without the aid of SAT) 4 out of the 9 cases with significant accelerations, including one case (*log2_10xd*) whose runtime is decreased from over 4 months to 1.4 days. Moreover, it solves a large portion of the problem on the cases *hyp_7xd* and *ac97_ctrl_10xd*, as indicated by the fast ABC verification of the reduced miters (subcolumn "ABC (s)") compared to exclusive ABC checking (column "ABC `&cec`"). The GPU engine can only reduce a minor part of the miter on the cases *sqrt_10xd* and *vga_lcd_5xd* but with a short runtime, so the overall solving time is marginally degraded. On average, the combination of GPU engine and ABC checker achieves 4.89× speed-up over the standalone ABC checker.

The combined GPU and ABC checker obtains an averaged 4.88× acceleration and is faster on 7 out of the 9 cases when compared to Conformal with 16 CPU threads. The commercial checker is much more efficient on the case *voter_10xd* and the reason might be that one of the engines in Conformal is well-suited to solving this case.

B. Breakdown Analysis of Simulation-based CEC Engine

Figure 6 shows the runtime percentages of different phases in the GPU engine, which differ across cases. In particular, *log2_10xd* and *sin_10xd* can be directly proved by PO checking (`P`). The rest of the cases (except *sqrt_10xd* that can hardly reduced by the GPU engine) are further analyzed in Figure 7, in which the intermediate miters during the execution of the GPU engine are checked by ABC, and the normalized ABC checking times (over the standalone ABC checking time of the case) are plotted. For instance, the labels PG and PGL refer to the checking times of the miters extracted after the `G` phase and at the end of the GPU flow, respectively. It can be seen that all the three types of phases play important roles in checking some cases, i.e., `P` on *ac97_ctrl_10xd*, `G` on *multiplier_10xd*, *square_10xd*, and `L` on most cases, which demonstrates the effectiveness of the overall design of the GPU CEC engine.

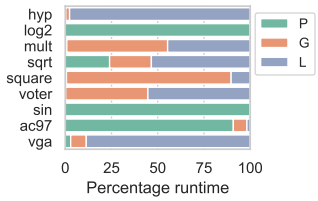


Fig. 6. Runtime breakdown of the simulation-based CEC engine. Benchmark names are abbreviated.

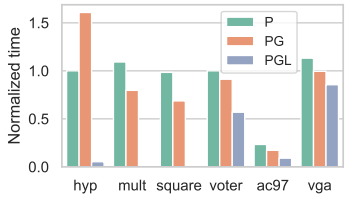


Fig. 7. Time of proving GPU-reduced miter by ABC with different flows, normalized by the time of standalone ABC.

V. DISCUSSION

Our CEC approach is implemented in a minimalist style due to the large development effort required. Here, we discuss a few tweaks that can be applied to further boost checking efficiency.

First, the current integration of the GPU engine and the ABC checker does not implement transferring ECs from GPU to ABC, and the ECs in ABC checking are reinitialized from scratch. With the EC transferring developed, nonequivalent pairs identified by GPU need not be rechecked by ABC, thus reducing the overall runtime.

Second, the GPU flow can be finetuned to be more adaptive, e.g., certain types of passes can be disabled on-the-fly during the repetitive `L` phases if they are found to be ineffective on the case.

Third, some improvements proposed in previous works may also be implemented on GPUs and integrated into our engine, including distance-1 simulation of CEXs [8], reverse simulation [21], and interleaving sweeping with logic rewriting [8], [14].

VI. CONCLUSION

This paper presents a new perspective on checking combinational equivalences using exhaustive simulation and a simulation-based CEC engine implemented on GPUs. The GPU engine is equipped with a fast exhaustive simulator with a high degree of parallelism, a local function checking scheme for proving difficult candidate pairs with restricted computational effort, and other GPU-based managers and algorithms that are crucial to the high overall performance of the engine. Experiments show that the GPU engine can efficiently solve four large cases without the aid of an external checker, and a CEC flow integrating the GPU engine and ABC checker achieves 4.89× and 4.88× speed-up on average over the standalone ABC checker and a commercial verification tool, respectively.

ACKNOWLEDGMENT

The authors acknowledge Yang Sun from CUHK and Jinwei Liu from HKBU for their useful comments and discussions.

REFERENCES

[1] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *Proc. ICCAD*, 2008, pp. 234–241.

[2] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *Proc. ICCAD*, 2005, pp. 519–526.

[3] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A simulation-guided paradigm for logic synthesis and verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2573–2586, 2022.

[4] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," in *Proc. ICCAD*, 2009, pp. 789–796.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, 1986.

[6] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. DAC*, 1997, pp. 263–268.

[7] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," ERL Technical Report, Tech. Rep., 2005.

[8] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proc. ICCAD*, 2006, pp. 836–843.

[9] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proc. DAC*, 2001, pp. 530–535.

[10] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. SAT*, 2003, pp. 502–518.

[11] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," in *Proc. CAV*, 2024, pp. 133–152.

[12] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 12, pp. 2131–2142, 2016.

[13] Y. Sun, T. Liu, M. D. Wong, and E. F. Young, "Massively parallel AIG resubstitution," in *Proc. DAC*, 2024, pp. 1–6.

[14] T. Liu and E. F. Young, "Rethinking AIG resynthesis in parallel," in *Proc. DAC*, 2023, pp. 1–6.

[15] R. K. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. CAV*, 2010.

[16] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, 2002.

[17] A. Mishchenko and R. Brayton, "SAT-based complete don't-care computation for network optimization," in *Proc. DATE*, 2005, pp. 412–417.

[18] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *Proc. ASP-DAC*, 2022, pp. 395–402.

[19] D. Brand, "Verification of large synthesized designs," in *Proc. ICCAD*, 1993, pp. 534–537.

[20] L. Amarú, F. Marranghello, E. Testa, C. Casares, V. Possani, J. Luo, P. Vuillod, A. Mishchenko, and G. De Micheli, "SAT-sweeping enhanced for logic synthesis," in *Proc. DAC*, 2020, pp. 1–6.

[21] H.-T. Zhang, J.-H. R. Jiang, L. Amarú, A. Mishchenko, and R. Brayton, "Deep integration of circuit simulator and SAT solver," in *Proc. DAC*, 2021, pp. 877–882.

[22] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko, "A circuit-based SAT solver for logic synthesis," in *Proc. ICCAD*, 2021, pp. 1–6.

[23] V. N. Possani, A. Mishchenko, R. P. Ribas, and A. I. Reis, "Parallel combinational equivalence checking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 3081–3092, 2019.

[24] D. Chatterjee and V. Bertacco, "EQUIPE: Parallel equivalence checking with GP-GPUs," in *Proc. ICCD*, 2010, pp. 486–493.

[25] NVIDIA, "CUDA C++ programming guide," 2024.

[26] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs," in *Proc. FPGA*, 1998, pp. 35–42.

[27] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *Proc. ICCAD*, 2007, pp. 354–361.

[28] T. Liu, L. Chen, X. Li, M. Yuan, and E. F. Young, "FineMap: A fine-grained GPU-parallel LUT mapping engine," in *Proc. ASP-DAC*, 2024, pp. 392–397.

[29] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proc. IWLS*, 2015.

[30] C. Albrecht, "IWLS 2005 Benchmarks," https://iwls.org/iwls2005/benchmarks.html, 2005.

[31] T. Liu, Y. Sun, L. Chen, X. Li, M. Yuan, and E. F. Young, "A unified parallel framework for LUT mapping and logic optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 1, pp. 214–226, 2025.

[32] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *Proc. DAC*, 2006, pp. 532–535.

[33] A. Koelbl, R. Jacoby, H. Jain, and C. Pixley, "Solver technology for system-level to RTL equivalence checking," in *Proc. DATE*, 2009, pp. 196–201.

DBC: Drift-aware Binary Code for Drift-tolerant Deep Neural Networks

Insu Choi*, Jaeyong Chung*†§, and Joon-Sung Yang*†‡§
*Department of Electrical and Electronic Engineering, †Department of Systems Semiconductor Engineering,
‡BK21 Graduate Program in Intelligent Semiconductor Technology, Yonsei University, Seoul, South Korea
{insuofficial, jyc, js.yang}@yonsei.ac.kr

***Abstract*—Deep neural networks (DNNs) have demonstrated outstanding performance across a wide range of applications. However, their substantial number of weights necessitates scalable and efficient storage solutions. Emerging non-volatile memory technologies, such as phase change memory (PCM) with multi-level cell (MLC) operation, are promising candidates due to their high scalability and non-volatility compared to conventional charge-based storage devices. Despite these advantages, MLC PCM suffers from reliability issues, particularly conductance drift, where the conductance of a PCM cell changes over time. This drift can lead to significant accuracy degradation in DNNs, as their weights are stored in PCM cells. In this paper, we propose *Drift-aware Binary Code* (DBC), a novel binary code designed to improve the tolerance of DNNs to conductance drift. DBC maps smaller decimal values to less error-prone MLC PCM cell levels and ensures that values shift to smaller magnitudes when conductance drift occurs. This approach helps maintain the accuracy of the DNN over an extended period compared to conventional binary code, as dominant DNN weights are stored at levels less prone to errors and DNNs exhibit better tolerance when weight values decrease rather than increase due to drift. Additionally, DBC requires no additional hardware overhead for auxiliary bits and can be combined with other fault-tolerant approaches, such as error correction code (ECC). Experimental results based on the real PCM device developed by IBM Research demonstrate that DBC improves the drift tolerance of DNNs by up to $55.18\times$ compared to conventional binary code.**

***Index Terms*—Deep neural network, emerging memory, phase change memory, conductance drift, fault-tolerance.**

I. Introduction

Deep neural networks (DNNs) have demonstrated exceptional performance across various applications. However, contemporary DNN architectures demand substantial storage capacity, reaching hundreds of megabytes or even gigabytes of weights [1]–[3]. The significant memory requirements pose challenges for deploying DNN models on resource-constrained edge devices, especially when relying on conventional charge-based storage devices like dynamic random-access memory (DRAM) due to scaling limitations.

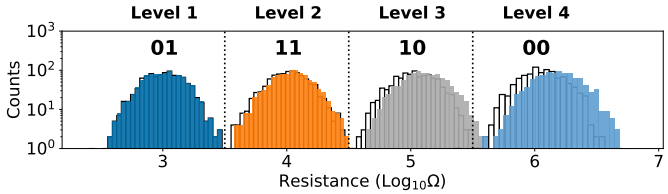
To address the challenges, emerging non-volatile memory technologies, such as phase-change memory (PCM), are being explored as promising alternatives to charge-based storage devices or as new storage-class memory layers within conventional memory hierarchies [4]. PCM devices consist of a phase-change material positioned between two electrodes, capable of exhibiting a wide range of electrical conductivities by transitioning between a highly conductive crystalline phase and a low-conductive amorphous phase, both of which are non-volatile states. Moreover, the ability of PCM to achieve a spectrum of conductance values enables multi-level cell (MLC) operation, which allows multiple bits to be stored in a single PCM cell, resulting in high scalability. However, despite this advantage in scalability, PCM with MLC operation faces reliability challenges due to conductance drift, where the conductance of the phase-change material decreases over time due to structural relaxation. Conductance drift can lead to soft errors, causing the stored data to deviate from their original values. In the context of DNNs, the soft errors in weights can result in a significant drop in accuracy [5].

Previous works [6]–[8] have proposed methods to mitigate the soft error rate caused by conductance drift in MLC PCM devices, effectively enhancing the drift tolerance of DNNs. However, these methods are specifically designed and optimized for the analytical PCM model introduced in [9], which is based on a theoretical representation of MLC PCM and not directly derived from real device measurements. As a result, these conventional methods face limitations when applied to the real PCM device developed by IBM Research [10]–[12], which exhibit distinct characteristics compared to the analytical PCM model, including differences in the error-prone levels of MLC PCM cells.

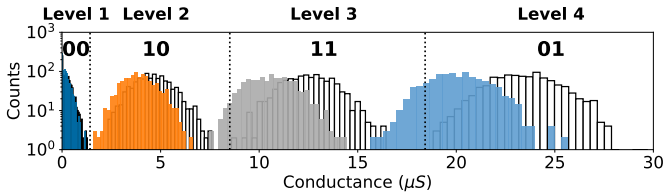
In this paper, we introduce *Drift-aware Binary Code* (DBC), a novel encoding scheme designed to enhance the drift tolerance of DNNs in the real PCM device. DBC maps smaller decimal values to lower MLC PCM cell levels, which are less susceptible to soft errors than higher levels. By applying DBC to DNN weights, accuracy can be preserved for a longer duration compared to conventional binary codes, as the Gaussian distribution of weights [13] ensures that dominant small weights are stored in more reliable cell levels. Moreover, DBC leverages conductance drift to shift decimal values toward smaller values, aligning with the improved tolerance of DNNs under such conditions [14]. DBC also eliminates the need for auxiliary bits, avoiding additional hardware overhead, and can integrate with other fault-tolerant techniques, such as error correction code (ECC), due to its independent operation.

The rest of the paper is organized as follows. Section II provides the background. Section III introduces DBC, followed by evaluations in Section IV. Finally, Section V presents the conclusion.

§Corresponding authors.



(a) Analytical model



(b) Real device

Fig. 1. Example of (a) resistance drift of the analytical model [9] and (b) conductance drift of the real device implemented by IBM Research [10]–[12]. The black-lined bins show the distribution of the PCM cells measured at 27 seconds, while the colored bins depict the distribution at 1024 seconds.

II. Background

A. Related Works

To mitigate the reliability challenges posed by conductance (or resistance) drift in MLC PCM, several studies have proposed drift-tolerant schemes [6]–[8]. These schemes focus on improving resistance drift tolerance of an analytical MLC PCM model described in [9]. Fig. 1a illustrates an example of resistance drift in the analytical PCM model, which comprises four Levels with target resistances of $10^3\,\Omega$, $10^4\,\Omega$, $10^5\,\Omega$ and $10^6\,\Omega$. A set of 1,000 PCM cells is programmed to each target resistances, following a Gaussian distribution. Each resistance range, delineated by dotted lines, corresponds to a unique binary representation encoded using reflected binary code, also known as Gray code, with Levels 1, 2, 3, and 4 represented as 01_2, 11_2, 10_2, and 00_2, respectively. When the resistance of a PCM cell shifts to a different range due to the drift, a soft error occurs because the data represented by the cell no longer matches the original data. For instance, a PCM cell initially set to $10^5\,\Omega$ may experience the drift, exceeding $10^{5.5}\,\Omega$ and resulting in a transition from Level 3 (10_2) to Level 4 (00_2). Similarly, the drift may cause shifts from Level 1 to 2, or from Level 2 to 3. The model demonstrates that the magnitude of the drift increases with higher cell levels. Consequently, mid-range Levels 2 and 3 are more vulnerable to errors, whereas Level 1 is less prone to soft errors due to its smaller drift magnitudes. Conversely, Level 4 is regarded as immune to drift errors, as there is no defined levels beyond Level 4.

In the analytical PCM model, Tri-level cell [6] improves drift tolerance by removing the most error-prone level and using only three levels. Flipcy [7] reduces drift-induced errors by minimizing the amount of data assigned to error-prone levels through simple bit-flip and complement operations. DynaPAT [8] employs a mapping strategy to enhance the drift tolerance of DNNs by aligning the most frequently used cell levels in DNN weights with the least susceptible levels.

TABLE I
Modeling Parameters for Real Device Characterization

Data	$G_T(\mu S)$	Drift		Read Noise	Prog. Noise	
		μ_ν	σ_ν	ν_q	$\sigma_{p.t0}$	$\sigma_{p.t1}$
00	0.25	0.079	0.031	0.030	0.183	0.208
10	4.96	0.060	0.014	0.081	0.653	0.531
11	13.5	0.060	0.007	0.077	0.969	0.687
01	25.0	0.055	0.004	0.046	1.069	1.128

B. PCM Modeling

The conventional methods [6]–[8] have been designed for the analytical model of MLC PCM. However, this model encounters limitations in accurately capturing the behavior of the real PCM device, as demonstrated by IBM Research [10]–[12]. To address this limitation, the model should incorporate various noise sources such as conductance drift, read noise, and programming noise, using direct measurements from the real device. The conductance of a real PCM device as a function of time is represented by following equations. Initially, the target conductance G_T is defined at approximately 20 seconds, as performed in [12]. The conductance measured at time $T_0 = 27.36$ seconds is expressed as follows:

$$\nu = N(\mu_\nu(G_T), \sigma_\nu(G_T)) \tag{1}$$

$$G(T_0) = G_T(T_0/20)^{-\nu} + N(0, \sigma_{p.t0}(G_T)) \tag{2}$$

where ν is a random variable following a normal distribution, and both mean μ_ν and standard deviation σ_ν depend on G_T. The term $\sigma_{p.t0}(G_T)$ represents the standard deviation of the programming noise measured at time T_0 for G_T. Additionally, noise is applied at time $T_1 = 55.39$ seconds to account for the short-term relaxation effects that occur after iterative programming [15], resulting in the following equation:

$$G(T_1) = G(T_0)(T_1/T_0)^{-\nu} + N(0, \sigma_{p.t1}(G_T)) \tag{3}$$

where $\sigma_{p.t1}(G_T)$ denotes the standard deviation of the programming noise at time T_1 for G_T. Finally, incorporating read noise, the conductance at any time t is defined as:

$$G(t) = G(T_1)(t/T_1)^{-\nu} + N(0, \sigma_{nG}(G_T, t)) \tag{4}$$

where $\sigma_{nG}(G_T, t)$ represents the standard deviation of the read noise, which depends on G_T and time t. This function is defined and detailed in the supplementary information of [12].

Fig. 1b illustrates an example of conductance drift based on the model of a real PCM device. The target conductance values for the four levels, along with the time-independent factors μ_ν, σ_ν, $\sigma_{p.t0}$, $\sigma_{p.t1}$ and ν_q (used in $\sigma_{nG}(G_T, t)$ from Equation (4)), are summarized in Table I. The level boundaries are defined at the 2.993σ point of each target conductance. The results demonstrate that conductance consistently drifts to lower levels over time, with transitions occurring from Level 4 (01_2) to Level 3 (11_2), Level 3 to Level 2 (10_2), and Level 2 to Level 1 (00_2). Higher levels exhibit greater magnitudes

of drift, rendering them more error-prone compared to lower levels. Specifically, Level 4 is identified as the most vulnerable to soft errors due to its substantial drift characteristics. This observation contrasts with the predictions of the analytical model, which identifies mid-levels as the most error-prone.

III. Proposed Method

Building upon the observed conductance drift characteristics of the real PCM device, we introduce *Drift-aware Binary Code* (DBC), a novel binary code designed to enhance the drift tolerance of DNNs. DBC assigns smaller decimal values to code words that include lower levels of MLC PCM which is less error-prone than higher levels. This approach improves drift tolerance by reducing the soft error rate, as the dominant small weight values in DNNs, which follow a Gaussian distribution [13], are stored in PCM cells associated with more reliable, less error-prone levels. Furthermore, DBC minimizes accuracy degradation by ensuring that the drift causes weight values to shift only toward smaller magnitudes. This unidirectional shift is beneficial because DNNs tend to preserve accuracy better when erroneous weight values decrease rather than increase [14].

A. DBC Generation Algorithm

Algorithm 1 presents the procedure for generating DBC. The algorithm takes as input the number of MLC PCM cells required to represent a single DBC code word, denoted by C, for $2C$-bit data. For instance, generating DBC for 8-bit data requires four cells ($C = 4$). The output is a 2D array A, where each row $(1 \dots N)$ represents a DBC code word mapped to the levels of MLC PCM cells, and each column $(1 \dots C)$ corresponds to the level values of individual cells. The algorithm sequentially assigns binary codes by prioritizing lower cell levels for smaller values, ensuring a progressive mapping from less error-prone levels to higher levels.

The procedure begins with **Step 1**, generating code words composed solely of Levels 1 and 2 by forming all possible permutations for the C cells using these levels. The resulting permutations are sorted in ascending order and stored in the array A. In **Step 2**, code words that include Level 3 are introduced. The algorithm iterates over the code words generated in the first step and creates new permutations by incorporating Levels 1, 2, and 3. These new code words are appended to A only if they satisfy the condition specified in Line 13 of Algorithm 1. This condition ensures that code words with the same combinations are ordered consistently based on the order of code words generated in the previous step. Finally, **Step 3** adds code words that include Level 4 to A. This is done in a manner similar to Step 2, by building upon the code words from the second step.

Table II provides an example of DBC code words generated for 4-bit data ($C = 2$), requiring two PCM cells, denoted as C_1 and C_2. The first column specifies the step in Algorithm 1 during which each code word is generated. The second column lists the code words, formatted as $\{$Level of $C_1\}$_$\{$Level of $C_2\}$. The third column presents their binary representations, where Levels 1, 2, 3 and 4 are encoded as 00_2, 10_2, 11_2, and 01_2, respectively, as shown in Fig. 1b. The fourth column displays the decimal values assigned to each code word. For instance, the code word L1_L3, where C_1 is in Level 1 and C_2 is in Level 3, corresponds to the binary representation 00_11_2. This code word is mapped to the decimal value 4 and is generated during Step 2 of Algorithm 1.

As illustrated in Table II, DBC maps smaller decimal values to lower levels, which exhibit reduced soft error rates when conductance drift occurs in the real PCM device. This mapping strategy is beneficial because the dominant weights in DNNs, which follow a Gaussian distribution with many small-magnitude values [13], are stored in the least error-prone cell levels. Consequently, the overall error rate due to conductance drift is reduced, as fewer weights are likely to induce soft errors. Furthermore, DBC leverages the unidirectional nature

Algorithm 1: DBC Generation Algorithm

Input : Number of MLC PCM cells used for representing a single DBC code word C
Output: 2D array of DBC code words represented in levels of MLC PCM cells A

```
   /* Step 0: Initialization                                  */
1  N ← 2^(2×C) // Total number of code words
2  Initialize 2D array of A[1...N][1...C]
   /* Step 1: Code words with Levels 1 & 2                    */
3  S_1 ← 2^C
4  P_1[1...S_1][1...C] ← Permutations({1,2}, C)
   /* Generate all possible permutations of
      length C using levels 1 and 2                           */
5  Sort(P_1) in ascending order
6  A[1...S_1] ← P_1[1...S_1]
   /* Step 2: Code words with Level 3                         */
7  S_2 ← 3^C − S_1
8  P_2[1...S_2][1...C] ← Permutations({1,2,3}, C)
9  Sort(P_2) in ascending order
10 i ← S_1 + 1
11 for j ← 1 to S_1 do
12     for k ← 1 to S_2 do
13         if (All positions of 1s in P_1[j] and P_2[k] are the
             same) and (P_2[k] is not in A[1...i−1]) then
14             A[i] ← P_2[k]
15             i ← i + 1
16         end
17     end
18 end
   /* Step 3: Code words with Level 4                         */
19 S_3 ← N − S_1 − S_2
20 P_3[1...S_3][1...C] ← Permutations({1,2,3,4}, C)
21 Sort(P_3) in ascending order
22 for j ← 1 to S_2 do
23     for k ← 1 to S_3 do
24         if (All positions of 1s and 2s in P_2[j] and P_3[k] are
             the same) and (P_3[k] is not in A[1...i−1]) then
25             A[i] ← P_3[k]
26             i ← i + 1
27         end
28     end
29 end
30 return A
```

TABLE II
EXAMPLE OF DBC WITH 4-BIT DATA USING TWO MLC PCM CELLS (C_1 AND C_2)

Algorithm Step	Code Word (Cell Levels)	Binary Representation	Decimal Value
Step 1	L1_L1	00_00_2	0
	L1_L2	00_10_2	1
	L2_L1	10_00_2	2
	L2_L2	10_10_2	3
Step 2	L1_L3	00_11_2	4
	L3_L1	11_00_2	5
	L2_L3	10_11_2	6
	L3_L2	11_10_2	7
	L3_L3	11_11_2	8
Step 3	L1_L4	00_01_2	9
	L4_L1	01_00_2	10
	L2_L4	10_01_2	11
	L4_L2	01_10_2	12
	L3_L4	11_01_2	13
	L4_L3	01_11_2	14
	L4_L4	01_01_2	15

of conductance drift in the real PCM device, which causes conductance values to decrease from higher to lower levels. As a result, when drift occurs, the decimal value represented by PCM cells become smaller than its original value. For example, consider a code word L3_L2, which corresponds to the binary representation 11_10_2 and maps to indicate the decimal value 7. If conductance drift occurs in cell C_1, the code word changes to L2_L2. As L2_L2 (10_10_2) is mapped to 3, the original decimal value of 7 is changed to 3 by the conductance drift, and this makes the changed value smaller than its original value. These changes are unidirectional, always resulting in a decrease in the stored value. By ensuring that conductance drift only decreases weight values, DBC enhances the drift tolerance of DNNs. A decrease in weight values from larger to smaller magnitudes results in a smaller accuracy drop compared to an increase in values [14]. Therefore, DBC improves the resilience of DNNs against conductance drift in the real PCM device, maintaining higher overall accuracy in the presence of drift-induced errors.

While DBC is initially designed for unsigned values, DNN weights are typically represented as signed values. To address this, we introduce a single-cell sign representation (SCSR), which serves as a sign indicator using an additional single PCM cell. In this scheme, Level 1 denotes positive numbers, while Levels 2, 3, and 4 indicate negative numbers, with Level 4 serving as the initial mapping for negatives. The configuration employs $(C+1)$ cells to represent $2C$-bit data. For example, the positive value +7 is represented by the code word L1_L3_L2, while the negative value -7 is initially encoded as L4_L3_L2. After conductance drift, -7 remains accurately represented, even if the sign cell level changes to 3 or 2, resulting in code words L3_L3_L2 or L2_L3_L2, respectively. The SCSR prevents changes from positive to negative values and significantly reduces the probability of an incorrect sign change.

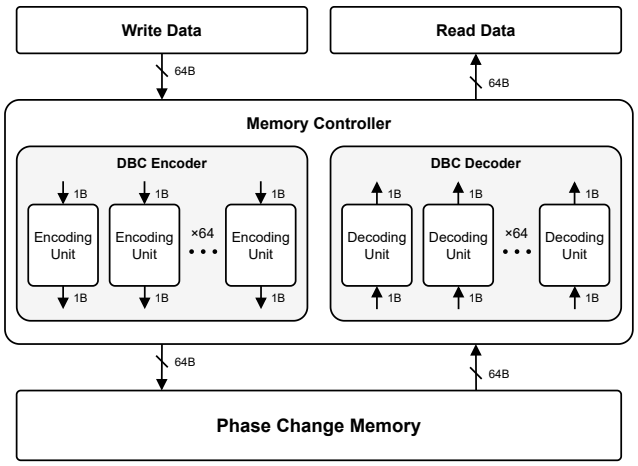


Fig. 2. Overall architecture of DBC encoder and decoder for 64-bytes cache line system.

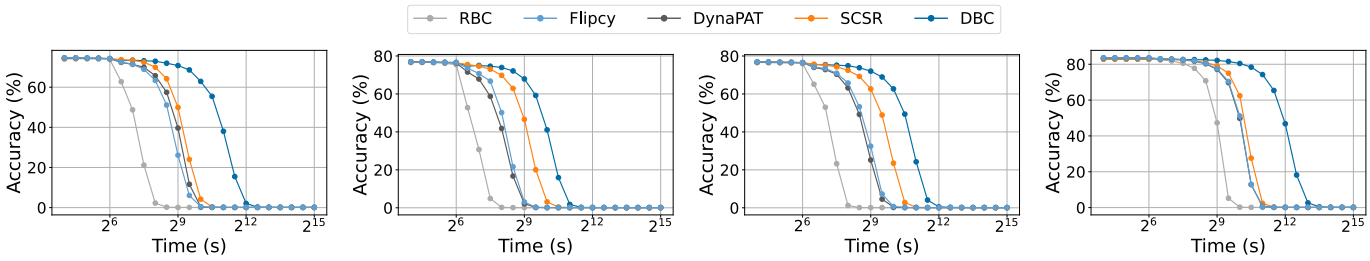
In summary, DBC combines Algorithm 1 for unsigned binary code generation with SCSR to support signed DNN weight values and improve drift tolerance.

B. Hardware Implementation

To facilitate the translation between the original binary code used in conventional systems and the DBC format, a DBC encoder and decoder are required. Fig. 2 illustrates the overall architecture of the DBC encoder and decoder integrated within the memory controller of a system utilizing a 64-byte cache line. The encoder and decoder each comprise 64 encoding units and 64 decoding units, respectively. Each encoding unit is a combinational circuit that converts 1 byte of data from conventional binary code into its DBC representation, while each decoding unit performs the inverse operation. The use of a 1-byte granularity for encoding and decoding units is driven by the widespread adoption of 8-bit quantization in various DNNs [16]–[18], as well as the 7-bit format employed for DBC with SCSR.

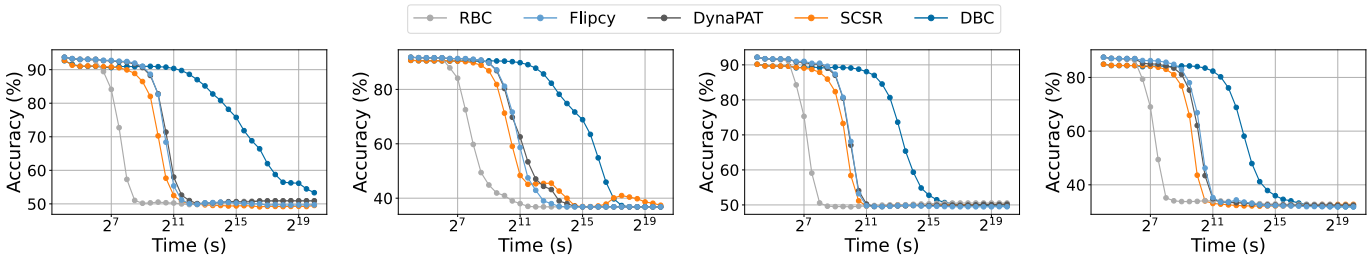
The write operation starts with a write request that includes the data to be stored. This data is encoded into the DBC format on a per-byte basis before being written to the corresponding MLC PCM cells. Similarly, the read operation retrieves data from the PCM, decodes it using the DBC decoder, and outputs 64 bytes of data to the host system.

The DBC scheme eliminates the need for auxiliary bits, avoiding additional hardware overhead apart from the DBC encoder and decoder when applied to PCM systems. Moreover, DBC can seamlessly integrate with other drift-tolerant or fault-tolerant techniques, such as error correction code (ECC), by incorporating extra encoding and decoding hardware at the output of the DBC encoder and the input of the DBC decoder, respectively. This compatibility arises from the complementary operation of DBC with these methods.



(a) VGG-19 (b) Inception-v3 (c) DenseNet-169 (d) ViT-Base

Fig. 3. Accuracy of different vision models evaluated on the ImageNet dataset across various time steps following the occurrence of conductance drift.



(a) SST-2 (b) QQP (c) QNLI (d) MNLI

Fig. 4. Accuracy of RoBERTa assessed on various GLUE benchmark tasks at different time steps following the occurrence of conductance drift.

IV. Evaluation

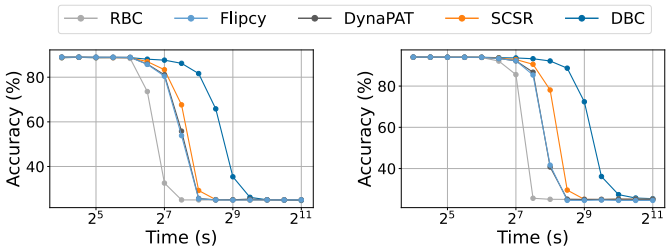
A. Experimental Setup

The proposed DBC is evaluated using various vision and natural language processing (NLP) models. The vision models, such as VGG-19 [19], Inception-v3 [20], DenseNet-169 [21], and ViT-Base [1], are employed with the ImageNet dataset [22]. In NLP tasks, RoBERTa [23] is evaluated across four tasks from the GLUE benchmark [24], including SST-2, QQP, MNLI, and QNLI. Additionally, Large Language Models (LLMs) such as Llama-3.2 1B [3] and Gemma-2 2B [2] are evaluated using the HellaSwag benchmark [25]. All models are quantized to 8-bit and 7-bit integers using the LSQ method [26]. The hardware implementations are developed in Verilog HDL and synthesized using a 28 nm standard cell library with Synopsys Design Compiler. To assess the effectiveness of the proposed DBC, comparisons are made with reflected binary code (RBC) and state-of-the-art methods such as Flipcy [7] and DynaPAT [8]. Additionally, single-cell sign representation (SCSR) combined with RBC is evaluated for ablation studies. While Flipcy and DynaPAT are originally designed using the analytical PCM model [6], they have been adapted to reflect the characteristics of the real PCM device, where levels 3 and 4 are treated as the most error-prone. Drift-induced fault simulations are performed using a custom-built simulator that accurately models the real PCM device [12].

B. Reliability Analysis

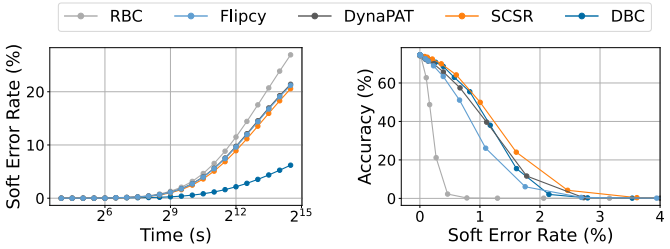
Fig. 3 illustrates the results of drift-induced fault simulations conducted on vision models. The accuracy of each DNN model, measured after conductance drift at each time step, is averaged over 100 iterations. The results reveal that drift-induced soft errors lead to a gradual decline in model accuracy over time. With the baseline scheme, RBC, an average accuracy drop of 5% from the original value occurs after averagely 113 seconds across all models. In comparison, the conventional encoding schemes Flipcy and DynaPAT extend the time to reach this accuracy drop by factors of 1.98× and 1.92×, respectively. SCSR improves drift tolerance by an average of 3.24× over RBC. The proposed DBC outperforms all other methods, achieving an average of 7.00× greater drift tolerance compared to RBC. For NLP tasks with RoBERTa, as shown in Fig. 4, DBC achieves the highest drift tolerance performance, significantly extending the time to a 5% accuracy drop by 48.77× compared to the 88 seconds observed with the baseline RBC. The maximum improvement is observed in QQP, where DBC reaches a retention time 55.18× longer than RBC. In contrast, other methods show shorter retention times, with Flipcy, DynaPAT, and SCSR extending drift tolerance by 6.91×, 6.54×, and 5.07×, respectively, compared to RBC. Moreover, to evaluate recent models more rigorously, we simulate the LLMs, as shown in Fig. 5. Despite the substantial number of weights in LLMs, the results are consistent with those obtained from vision models and NLP tasks. Specifically, for RBC, a 5% accuracy drop is observed at an average of 91 seconds across the two LLMs. In comparison to this baseline, the 5% accuracy drop points for Flipcy, DynaPAT, and SCSR occur at average times that are 1.37×, 1.42×, and 1.72× longer, respectively, while DBC achieves a 3.08× longer time to the 5% accuracy drop point.

We conduct a detailed analysis of the relationship among time, soft error rate, and DNN accuracy using VGG-19. Fig. 6a illustrates the progression of the soft error rate over time. The soft error rate is defined as the proportion of erroneous PCM cells, whose stored values deviate from the original



(a) Llama-3.2 1B (b) Gemma-2 2B

Fig. 5. Accuracy of LLMs evaluated on the HellaSwag benchmark at different time steps following the occurrence of conductance drift.



(a) Time vs. Soft Error Rate (b) Soft Error Rate vs. Accuracy

Fig. 6. Relationship among time, soft error rate, and accuracy in VGG-19.

due to conductance drift, relative to the total number of PCM cells. The results show that the soft error rates for RBC, Flipcy, DynaPAT, and SCSR rise steeply, exceeding 20% by 2^{14} seconds. In contrast, DBC exhibits a much more gradual increase, reaching only 7.16% at the same time, consistently maintaining a lower soft error rate than the other methods across all time steps. Additionally, Fig. 6b depicts the DNN accuracy as a function of the soft error rate. At the 5% accuracy drop point, RBC, Flipcy, DynaPAT, and SCSR exhibit soft error rates of 0.02%, 0.23%, 0.23%, and 0.36%, respectively. In comparison, DBC achieves the highest soft error rate of 0.39%, which is $19.5\times$ higher than that of RBC at the same accuracy drop threshold. These results demonstrate that DBC significantly outperforms other methods by reducing the soft error rate over time and maintaining higher accuracy at equivalent soft error rates.

C. Hardware Analysis

Table. III presents the area, energy, and latency metrics for the encoders and decoders of RBC, Flipcy, DynaPAT, and DBC, based on a cache line size of 64 bytes. The SCSR scheme utilizes the same hardware as RBC but incorporates additional software support. Among the evaluated encoders and decoders, DBC incurs the highest area overhead and energy consumption, except for encoder energy, where Flipcy consumes more. Despite this, DBC eliminates the need for auxiliary bits, avoiding additional hardware overhead beyond the encoder or decoder itself. In contrast, Flipcy relies on read and write operations for auxiliary bits at a granularity of 4 bytes per line from PCM, leading to increased area and energy costs. Similarly, DynaPAT requires extra hardware for auxiliary bit management, involving read and write operations from the host register or PCM, with caching mechanisms

TABLE III
HARDWARE COMPARISON

		RBC	Flipcy	DynaPAT	DBC
Area (mm^2)	**Encoder**	0.001	0.011	0.004	0.012
	Decoder	0.001	0.003	0.004	0.012
Energy (pJ)	**Encoder**	0.13	4.56	1.13	1.98
	Decoder	0.13	0.45	1.13	1.95
Latency (ns)	**Encoder**	0.19	1.84	1.03	0.61
	Decoder	0.19	1.88	1.03	0.66

during the processing of the same layers in DNN models. The absence of auxiliary bit requirements provides DBC with notable system-level advantages. Additionally, DBC achieves superior latency performance, recording the lowest latency among encoders and decoders of Flipcy, DynaPAT and DBC. However, while DBC exhibits larger hardware overheads in terms of area, energy, and latency compared to RBC and SCSR, it significantly enhances drift tolerance. For example, in the QQP task shown in Fig. 4b, DBC achieves a $55.18\times$ improvement in drift tolerance over RBC, whereas SCSR provides a $5.47\times$ improvement over RBC, highlighting a trade-off between hardware overhead and drift tolerance.

V. CONCLUSION

The behavior of the real MLC PCM device, implemented by IBM Research, deviates from the analytical model used in previous studies. Specifically, higher PCM cell levels in the real device is more prone to soft errors due to conductance drift, whereas the conventional model typically identify mid-levels as the most error-prone. To address this, we introduce the *Drift-aware Binary Code* (DBC), a novel binary code designed to enhance the drift tolerance of DNNs operating on the real PCM device. DBC assigns the more reliable lower levels of PCM cells to represent lower decimal values. This approach reduces the drift-induced soft error rate, considering that smaller absolute values dominate the weight distribution in DNNs. Furthermore, DBC minimizes the negative impact on DNN accuracy by ensuring that drift only reduces absolute weight values. Notably, DBC requires no auxiliary bits and operates independently of other drift-tolerant or fault-tolerant mechanisms, such as Error Correction Codes (ECC), allowing it to be seamlessly integrated with existing solutions. Experimental results demonstrate that DBC enhances the drift tolerance of DNNs by up to $55.18\times$ compared to the conventional reflected binary code.

ACKNOWLEDGMENT

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00971), and in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00405495, Plug&Play (P&P) Chiplet Integration research center). The EDA Tools used in this work were supported by IDEC, Daejeon, South Korea.

REFERENCES

[1] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[2] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé *et al.*, "Gemma 2: Improving open language models at a practical size," *arXiv preprint arXiv:2408.00118*, 2024.

[3] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[4] M. Le Gallo and A. Sebastian, "An overview of phase-change memory device physics," *Journal of Physics D: Applied Physics*, vol. 53, no. 21, p. 213002, 2020.

[5] S. Lee, I. Choi, and J.-S. Yang, "Bipolar vector classifier for fault-tolerant deep neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 673–678.

[6] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 440–451.

[7] M. Imran, T. Kwon, J. M. You, and J.-S. Yang, "Flipcy: Efficient pattern redistribution for enhancing mlc pcm reliability and storage density," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[8] T.-H. Nguyen, M. Imran, and J.-S. Yang, "Dynapat: A dynamic pattern-aware encoding technique for robust mlc pcm-based deep neural networks," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[9] W. Xu and T. Zhang, "A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in the presence of significant resistance drift," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 8, pp. 1357–1367, 2010.

[10] G. F. Close, U. Frey, M. Breitwisch, H. Lung, C. Lam, C. Hagleitner, and E. Eleftheriou, "Device, circuit and system-level analysis of noise in multi-bit phase-change memory," in *2010 International Electron Devices Meeting*. IEEE, 2010, pp. 29–5.

[11] G. W. Burr, M. J. Brightsky, A. Sebastian, H.-Y. Cheng, J.-Y. Wu, S. Kim, N. E. Sosa, N. Papandreou, H.-L. Lung, H. Pozidis *et al.*, "Recent progress in phase-change memory technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 146–162, 2016.

[12] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature communications*, vol. 11, no. 1, p. 2473, 2020.

[13] S.-S. Lee and J.-S. Yang, "Value-aware parity insertion ecc for fault-tolerant deep neural network," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 724–729.

[14] I. Choi, J.-Y. Hong, J. Jeon, and J.-S. Yang, "Rq-dnn: Reliable quantization for fault-tolerant deep neural networks," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–2.

[15] R. Degraeve, A. Fantini, G. Gorine, P. Roussel, S. Clima, C. Chen, B. Govoreanu, L. Goux, D. Linten, M. Jurczak *et al.*, "Quantitative model for post-program instabilities in filamentary rram," in *2016 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2016, pp. 6C–1.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[17] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 318–30 332, 2022.

[18] D. Shin, I. Choi, and J.-S. Yang, "Vit-slice: End-to-end vision transformer accelerator with bit-slice algorithm," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[23] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[24] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[25] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[26] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *International Conference on Learning Representations*, 2020.

Decoupling Analog Circuit Representation from Technology for Behavior-Centric Optimization

Jintao Li [1,2,†], Haochang Zhi [3,†], Jiang Xiao [1], Keren Zhu [2,*], Yun Li [1,*]
[1]Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen 518110, China
[2]School of Microelectronics, Fudan University, Shanghai 200433, China
[3]School of Electronic Science and Engineering, Southeast University, Nanjing 214135, China

***Abstract*—Analog IC design is mainly manual and implemented at the device level. A major reason is circuit behavior-extraction. Unlike its digital counterpart, analog IC design is strongly coupled with technology nodes and is difficult to represent by an abstract behavioral model. The lack of accurate and efficient analog modeling has become a bottleneck in analog design automation. This paper proposes a behavior-centric optimization framework for analog circuits that represents circuit behavior using transistor electrical properties instead of sizes, improving model generalization and reducing optimization complexity. To characterize the process, we propose a method for mapping transistor electrical properties to sizes. Moreover, we developed a radial basis functions-based Kolmogorov-Arnold network (RBF-KAN) to accurately approximate circuit nonlinear behavior with limited simulations. Compared to black-box modeling, our approach enables constructing surrogate models via KAN under a set specification with just a few hundred simulations. Experiments on the testing suite showed our framework achieved a 1.76× to 2.64× improvement in large signal figure of merit (FOM) and 1.73× to 2.48× in small signal FOM over state-of-the-art methods, while also enabling 3.5× to 6.2× acceleration in design porting.**

***Index Terms*—Analog circuit sizing, Kolmogorov-Arnold Networks, Knowledge Transfer, Behavioral Model**

I. INTRODUCTION

While digital design benefits from mature CAD tools, analog design still relies on experienced experts. However, as process technology advances and application demands increase, stricter design specifications drive the need for greater automation and intelligence in analog IC sizing [1]–[3].

Currently, most approaches rely on treating the analog sizing problem as a black box, employing machine learning (ML) techniques like reinforcement learning (RL) [4], [5], Bayesian optimization (BO) [6], and evolutionary algorithms (EA) [3], [7] to explore the design space. These methods often overlook the circuit's inherent behavior, instead relying on extensive computations to meet specifications, requiring substantial simulation time and constructing accurate surrogate models.

Considering that circuits are commonly represented as schematics, existing learning algorithms employ graph neural networks (GNNs) [8] to leverage topological information to open the black box. Nonetheless, this graph representation mainly encodes device connectivity, providing limited insight into the circuit's deeper behavior, thus still requiring **thousands of simulations** to build an accurate surrogate model.

Deriving more meaningful design insights requires establishing signal models and response analysis, which link the electrical parameters of devices to circuit performance, enabling guiding designs. In fact, early analog electronic design automation (EDA) methods employed ***symbolic*** approaches [10] to approximate circuit equations rather than relying solely on simulation. For human experts, a commonly used approach is the g_m/I_D methodology [9], which characterizes circuit performance based on transconductance (g_m) to drain current (I_D). As shown in Figure 1, this approach provides behavior-level insights that reduce reliance on specific process design kits (PDKs) data, enabling more efficient optimization than simulation-based methods. While this approach was efficient, it gradually fell out of favor due to three main challenges: (1) Technology scaling introduced non-idealities, making accurate equation-based approximation difficult. (2) Fitting separate equations was required for each specification and circuit topology. (3) Mapping behavior to design parameters requires **presetting the transistor length** (L), making the design process dependent on trial and error and limiting cross-process portability.

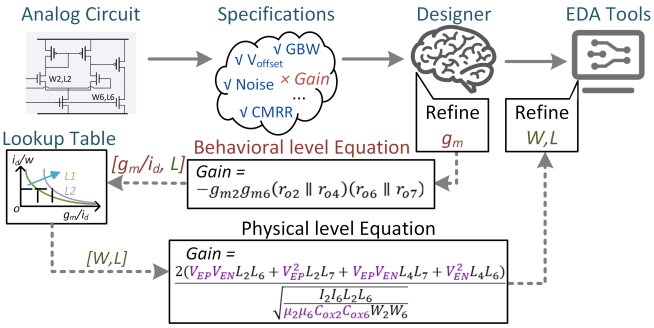


Fig. 1. Designers approximate the behavioral equation for performance metrics based on circuit theory and then utilize the g_m/I_D [9] methodology mapping to specific design parameters.

Revisiting these limitations of the classic equation-based methods, we observed that the key challenges lie in the following: (1) Due to the **strong coupling with the PDKs**, building surrogate models for analog blocks requires capturing their highly nonlinear properties and fitting the corresponding process characteristics. (2) ML is primarily based on ***connectionism***, while circuit representation relies on ***symbolism***, which hinders effective integration. (3) Expanding the objective space with complete specifications leads to ***exponential*** complexity in managing the Pareto front (PF).

We propose a framework for behavioral-centric optimization of analog circuits (BCOA) to address these challenges. Inspired by human expert workflows, we decouple the analog block from the process. Specifically, we approximate circuit behavior with **process-independent electrical properties** and have developed a **direct mapping method** from these properties to transistor sizes, eliminating the need to preset L and facilitating porting. This enables optimization and modeling to directly focus on analog behavior, bridging the gap between symbolic and connectionist approaches.

The contributions of this work are summarized as follows.

1) We decouple the analog block from the process and propose

*Corresponding authors: (Krzhu@fudan.edu.cn, Yun.li@ieee.org).
†Both authors contributed equally to this research.

a behavioral-level optimization framework that reduces the complexity of modeling while enhancing generalization across different technologies.

2) We propose a method to map electrical properties to transistor sizes and develop a radial basis function-based Kolmogorov-Arnold network (RBF-KAN) model to learn circuit nonlinear behavior from limited simulations efficiently.
3) We propose a method for Pareto front estimation that reduces the time complexity from $O(n^m)$ to $O(n^2 \cdot \log n + n \cdot m)$, enabling efficient complete specification optimization.
4) Experiments based on the AnalogGym suite show that the proposed framework builds accurate behavioral-level surrogate models with only a few hundred simulations and effectively generalizes across different processes. Compared to state-of-the-art algorithms, BCOA achieves a 1.73 to 2.64 $\times$ improvement in FOM_L and FOM_S.

II. PRELIMINARIES

A. Surrogate Model for Analog Blocks

Gaussian processes (GP) [6], [11] is among the most popular surrogate modeling methods due to their strong learning ability with limited simulations. However, considering complete specifications, often ranging from 10 to 20, GPs struggle with scalability, causing computational costs to grow exponentially with more objectives or samples [2].

Recent research explores to use deep learning for surrogate modeling due to its scalability for multiple objectives. However, this approach requires large amounts of data for effective training and struggles to model nonlinear transistor behavior accurately. In contrast, KAN [12] uses learnable B-splines as activation functions to decompose complex multivariate functions into simpler univariate components. We observe that this property allows KANs to efficiently capture nonlinear relationships with limited data, providing more accurate and scalable learning.

B. Design Flow with g_m/I_D Methodology

g_m/I_D methodology [9], [13] is a widely-used design methodology that links small-signal models with transistor sizes. In the g_m/I_D design methodology, transistor characteristics for various channel lengths L and widths W are simulated and measured, capturing the relationship between g_m/I_D and the normalized current $I_D/(W/L)$. This data is then compiled into pre-simulated lookup tables (LUTs). Subsequently, the designer selects the appropriate operating region based on the performance requirements. The weak inversion region is typically chosen for low-power applications, where the g_m/I_D is higher. In contrast, the strong inversion region, characterized by a lower g_m/I_D, is preferred for high-speed applications. Once the operating area is defined, the designer utilizes the LUT to determine the appropriate device dimensions.

C. Automatic Analog Sizing

Recent research trends on analog sizing often focus on efficiency. To enhance optimization efficiency, techniques like truncated subspace sampling [14] and domain decomposition [15] have been introduced, targeting specific areas of the design space to reduce computation and accelerate convergence. However, these methods may potentially miss points with better circuit performance. We investigate an amplifier optimization task with seven objectives conducted on the AnalogGym testing suite [16], domain decomposition-based BO reduced the required simulation runs by a factor of 10 compared to EA. However, this improvement in efficiency came at a substantial cost: the figure of merit (FOM) dropped by **10 to 100** $\times$. In BCOA, we use *behavioral representation* to reduce optimization complexity and enhance model generalization, along with efficient PF estimation to guide the many-objective optimization.

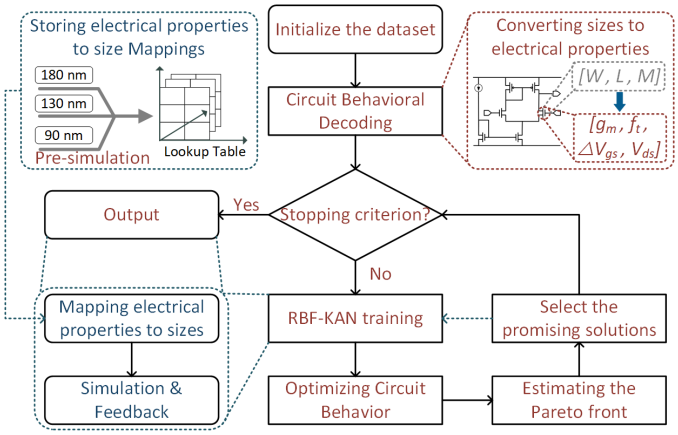


Fig. 2. The overall flow of the proposed BCOA framework.

III. FRAMEWORK

A. Framework Overview

BCOA mainly consists of six steps, as shown in Figure 2. 1) We use pre-simulation to build LUTs that map electrical properties to transistor sizes, as outlined in Section III-B. 2) We apply Latin hypercube sampling [17] to generate a small set of candidate designs as a database and use simulations to obtain transistor electrical properties that represent circuit behavior. 3) If the stopping criteria are met, the best design is selected; otherwise, we proceed with further optimization. 4) We train the RBF-KAN model using the solutions detailed in Section III-C and their corresponding performance from the database as training data. For the new solutions, we use the LUTs from Step 1 to map transistor electrical properties back to sizes and simulate to obtain the corresponding performance. 5) We apply an optimization algorithm to refine circuit behavior using predictions from the RBF-KAN. Notably, the BCOA framework is compatible with various optimization algorithms, and we select MOEA/D-DE [18] as the optimizer for its efficiency and flexibility. 6) The PF is estimated following the method in Section III-D, allowing us to select the most promising solutions for the next iteration.

B. Transistor Size Mapping Method

We develop a transistor size mapping method to decouple circuit behavior from technology, overcoming the g_m/I_D methodology's reliance on presetting L to determine size. Therefore, the key idea of our method is to use large-signal and small-signal parameters to characterize transistor electrical properties and, building on the g_m/I_D methodology, to fix L using these parameters, as shown in Figure 3. The small-signal parameters include g_m, which represents the change in drain current due to a change in gate voltage, and transition frequency f_t, which measures high-frequency performance. These parameters determine bandwidth, slew rate (SR), power consumption, phase, and transient response. The large-signal parameters, including drain-source voltage V_{ds} and overdrive voltage ΔV_{gs} (defined as $V_{gs} - V_{th}$, where V_{gs} is the gate-source voltage and V_{th} is the threshold voltage), define the transistor's operating region and influence swing, gain, and noise performance.

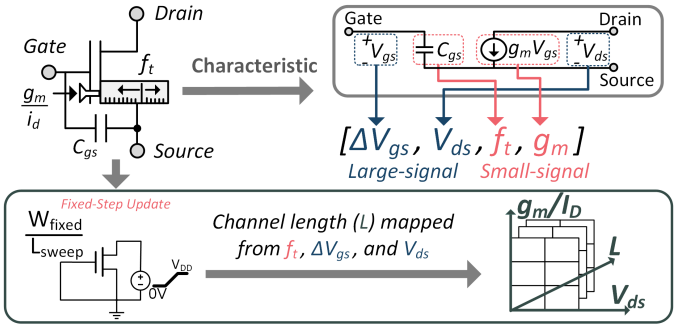


Fig. 3. We use large-signal and small-signal parameters to characterize transistors, providing a more accurate mapping from electrical properties to design parameters than the g_m/I_D methodology.

The mapping between electrical properties and transistor sizes uses the formulas in Table I to calculate L, and the corresponding sizes are then derived from the g_m/I_D LUT. Further details on constructing a LUT with the g_m/I_D method can be found in [13].

TABLE I
CORRELATION BETWEEN ELECTRICAL PROPERTIES AND SIZES

Parameter	Formulas	Relationship
$g_m(A/V)$	$\sqrt{2\mu C_{ox}\frac{W}{L}I_{ds}}$	$\sqrt{W/L}$
$f_t(Hz)$	$\frac{3\mu I_{ds}}{2\pi g_{m0}L^2}$	$1/L^2$
$\Delta V_{gs}(V)$	$\sqrt{\frac{4\pi f_{t0} g_{m0} L^3}{3\mu^2 C_{ox} W}}$	$L\sqrt{\frac{L}{W}}$
$V_{ds}(V)$	Non-analytic	$\sim L/W$

C. RBF-KAN

We adopt RBF-KAN to enable data-efficient and accurate modeling of circuit behavior, enhancing scalability and sensitivity to both global and local features. In KAN, each node x_j^{l+1} in layer $l+1$ is computed as the sum of activation functions $\phi_{l,i,j}$ applied to nodes x_i^l in the previous layer:

$$x_j^{l+1} = \sum_{i=1}^{n_l} \phi_{l,i,j}(x_i^l), \tag{1}$$

where x_i^l represents the i-th node in layer l, and $\phi_{l,i,j}$ denotes the activation function on the edge. The B-splines are adaptable basis functions for activation, enhancing nonlinear fitting capabilities. However, their grid-based structure limits flexibility with non-uniform data and increases computational complexity.

To enhance global smoothness and long-range feature capture with better scalability in high dimensions, we replace B-splines with Gauss-Laplace RBF activations [19], defined as:

$$\phi_{l,i,j}(x_i^l) = W_{ij}^l \cdot \exp\left(-\alpha(x_i^l - g_{ij}^l)^2 - \beta|x_i^l - g_{ij}^l|\right), \tag{2}$$

where α controls the scale of the Gaussian component, capturing smooth variations and low-frequency features, and β determines the decay rate of the Laplace component, enhancing sensitivity to rapid changes and high-frequency signals. W_{ij}^l represents the edge weight, g_{ij}^l denotes the RBF center, and α and β are hyperparameters that control the RBF shape.

For backpropagation, the gradient of the loss function to the weight W_{ij}^l is:

$$\frac{\partial L}{\partial W_{ij}^l} = \delta_j^{l+1} \cdot \phi_{l,i,j}(x_i^l), \tag{3}$$

where $\delta_j^{l+1} = \frac{\partial L}{\partial x_j^{l+1}}$. The gradient for the center g_{ij}^l requires the partial derivative of the activation function concerning g_{ij}^l:

$$\frac{\partial \phi_{l,i,j}}{\partial g_{ij}^l} = W_{ij}^l \cdot \phi_{l,i,j}(x_i^l)\left(2\alpha(x_i^l - g_{ij}^l) + \beta \cdot \mathrm{sgn}(x_i^l - g_{ij}^l)\right), \tag{4}$$

which leads to the gradient as $\frac{\partial L}{\partial g_{ij}^l} = \delta_j^{l+1} \cdot \frac{\partial \phi_{l,i,j}}{\partial g_{ij}^l}$.

To propagate gradients to the previous layer, we compute the gradient concerning the input x_i^l:

$$\frac{\partial L}{\partial x_i^l} = \sum_{j=1}^{n_{l+1}} \delta_j^{l+1} \cdot \frac{\partial \phi_{l,i,j}}{\partial x_i^l}, \tag{5}$$

where the partial derivative of the activation function for x_i^l is:

$$\frac{\partial \phi_{l,i,j}}{\partial x_i^l} = W_{ij}^l \cdot \phi_{l,i,j}(x_i^l)\left(-2\alpha(x_i^l - g_{ij}^l) - \beta \cdot \mathrm{sgn}(x_i^l - g_{ij}^l)\right). \tag{6}$$

D. Pareto Front Estimation

We propose a Monte Carlo-based hypervolume (MC-HV) estimation method to enhance objective scalability and effectively guide optimization. MC-HV approximates the HV by dynamically adjusting reference points closer to the PF and increasing sampling density in complex regions according to curvature. Algorithm 1 illustrates the MC-HV procedures. The ϵ controls the convergence precision of the reference point, typically set to 10^{-3} or 10^{-4}. The Δ, defined as 10% of the objective range of the solution set, ensures that the initial reference point is sufficiently far from the solution set. The $\beta = 0.5$ regulates the rate at which the reference point approaches the PF, while the $\gamma = 1$ adjusts the impact of the distance term in curvature weighting. The $\alpha = 100$ influences the total sample count, and the $\beta = 0.5$ increases sampling density in high-curvature regions. The distance term $d(z, S) = \min_{s \in S} \|z - s\|$ in $p(z)$ concentrates sampling near the solution set to enhance HV estimation accuracy. In high-curvature regions, the sampling count increases adaptively, with $N_i = \alpha \cdot (1 + \kappa(s_i))^\beta$, ensuring more sampling points in complex regions to improve accuracy.

The time complexity of the MC-HV method primarily depends on the solution set size n and the number of objectives m. The hypervolume estimation complexity is mainly due to dominance checks on each sampling point, where each check has a complexity of $O(n \cdot \log n)$, resulting in a complexity of $O(n^2 \cdot \log n)$. The reference point adjustment complexity is $O(n \cdot m)$, with m representing the number of objectives.

E. Design Porting

Behavioral-level optimization enables more efficient and flexible cross-process portability than transistor sizing. Figure 4 illustrates a case study of circuit behavior-based design porting from 180 nm to 130 nm. Since our framework represents the circuit using process-independent transistor electrical properties, the optimized circuit behavior at 180 nm is mapped to specific dimensions at 130 nm through the LUT of the 130 nm process. As process nodes shrink, both V_{th} and supply voltage (V_{dd}) decrease to enhance speed and reduce power, so fine-tuning primarily focuses on the ΔV_{gs} and V_{ds}. For the RBF-KAN, we implement knowledge transfer using hard parameter sharing [20].

TABLE II
SUMMARY OF OPTIMIZATION PERFORMANCE AVERAGED OVER 10 RUNS FOR DIFFERENT TOPOLOGIES

Topology	NMCF				NMCNR				DFCFC1			
Algorithm	rGNNRL	cVTSBO	g_m/I_DRL	BCOA	rGNNRL	cVTSBO	g_m/I_DRL	BCOA	GCNRL	cVTSBO	g_m/I_DRL	BCOA
PM (°)	64.3	59.5	51.2	49.6	67.2	60.3	48.2	52.4	52.7	59.4	55.4	49.6
Gain (dB) ↑	127.6	**137.2**	126.4	119	125.6	123.4	**129.4**	116.4	119.6	**127.1**	125.6	124.1
PSRR (dB) ↓	-73.1	-70.3	-70.5	**-75.4**	-67.5	-58.3	-75.2	**-89.6**	-74.4	-66.2	-72.4	**-76.2**
CMRR (dB) ↓	**-72.9**	-69.4	-68.2	-67.2	-74.2	-46.2	-80.5	**-88.5**	-73.7	-67.5	-77.6	**-79.4**
Ts (μs) ↓	5.4	3.6	3.2	**2.2**	15.2	14.2	9.6	**2.3**	15.6	16.4	10.6	**0.6**
FOM_L (V/μs*pF/mW) ↑	192.1	171.4	256.1	**451.9**	152.3	175.4	219.4	**396.2**	416.2	408.25	506.2	**964.2**
FOM_S (MHz*pF/mW) ↑	436.4	485.9	624.6	**1081.4**	894.2	977.6	952.6	**1409.4**	774.2	767.23	824.5	**1352.7**
Active Area (mm^2) ↓	142.5	139.7	158.2	**121.5**	296.4	245.6	257.6	**232.7**	109.5	84.1	119.6	**71.6**
FOM_{AMP} ↑	19.1	28.5	49.4	**283.5**	4.9	4.2	17.0	**186.2**	31.8	33.2	59.8	**5863.1**
FEs Consumption ↓	12257.2	**122.4**	8554.7	3665.2	10598.2	**116.8**	8752.4	2982.4	9624.5	**120.4**	10265.4	3106.5
Simulation Count ↓	5274.5	**122.4**	3254.3	401.2	4106.4	**116.8**	3626.4	426.5	3962.4	**120.4**	4216.5	376.4
Total Runtime (min) ↓	413.5	**8.8**	224.3	22.6	352.6	**8.9**	294.5	19.4	336.1	**9.6**	354.2	20.2

Algorithm 1 MC-HV Method

Input: Solution set S, termination threshold ϵ, sampling parameter γ, sampling control parameter α, curvature influence parameter β, number of samples N

Initialize r for each objective: $r_{i,\text{initial}} = \max_{s\in S} s_i + \Delta$
while $\max_i (r_i - \max_{s\in S} s_i) > \epsilon$ **do**
  Update each dimension of reference point r: $r_i = \max_{s\in S} (s_i + \beta(r_{i,\text{initial}} - s_i))$
end while
Compute curvatures for each solution $s_i \in S$: $\kappa(s_i) = \frac{\|\mathbf{n}(s_i) - \mathbf{n}(s_j)\|}{\|s_i - s_j\|}$
Initialize sample set: $\mathcal{S} \leftarrow \emptyset$
for $i = 1, \ldots, N$ **do**
  Sample $z \sim p(z) = \frac{1}{Z} \exp\left(-\gamma \cdot d(z, S) \cdot \frac{1}{1+\kappa(z)}\right)$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{z\}$
end for
Estimate hypervolume $HV(S)$ using Monte Carlo integration: $HV(S) \approx \frac{1}{N} \sum_{z\in\mathcal{S}} \frac{\mathbb{I}(z \text{ is dominated by } S)}{p(z)}$
return $HV(S)$

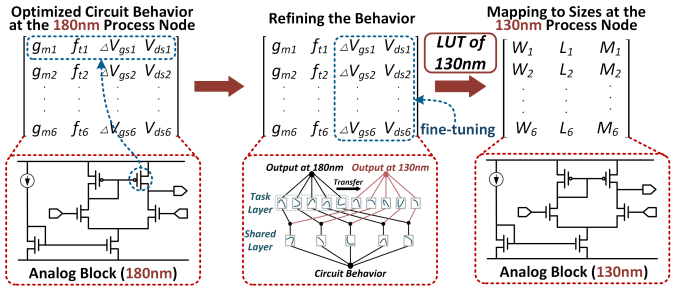


Fig. 4. Our framework enables design porting by mapping optimized behavior to transistor sizes at the new process node via an LUT, with design knowledge transferred through hard parameter sharing in the surrogate model.

IV. EXPERIMENTAL RESULTS

We use the reference metric FOM_{AMP} provided by AnalogGym [16] to evaluate the optimization results by comparing the optimized performance metrics against the baseline metrics, which are also already offered within AnalogGym. The formula is as follows:

$$\text{FOM}_\text{AMP} = \left(\frac{\text{PSRR}}{\text{PSRR}_\text{ref}} \cdot \frac{\text{CMRR}}{\text{CMRR}_\text{ref}} \cdot \frac{\text{Gain}}{\text{Gain}_\text{ref}} \cdot \frac{\text{FOMS}}{\text{FOMS}_\text{ref}} \cdot \frac{\text{FOML}}{\text{FOML}_\text{ref}}\right) \times \left(\frac{Ts}{Ts_\text{ref}} \cdot \frac{\text{Area}}{\text{Area}_\text{ref}}\right)^{-1} \times \left(\frac{v_n}{v_{n_\text{ref}}} \cdot \mathbb{I}(v_n > v_{n_\text{ref}}) \cdot \frac{TC}{TC_\text{ref}} \cdot \mathbb{I}(TC > TC_\text{ref}) \cdot \frac{vos}{vos_\text{ref}} \cdot \mathbb{I}(vos > vos_\text{ref})\right)^{-1}, \tag{7}$$

$$\text{FOM}_S = \frac{\text{GBW}[MHz] \cdot C_{Load}[pF]}{\text{Power}[mW]}, \quad \text{FOM}_L = \frac{\text{SR}[V/\mu s] \cdot C_{Load}[pF]}{\text{Power}[mW]} \tag{8}$$

where v_n is the integrated noise, Ts is the settling time, and vos is the input offset voltage. PSRR represents the power supply rejection ratio, while CMRR denotes the common mode rejection ratio. TC is the temperature coefficient, GBW is the gain-bandwidth product, C_{Load} is the load capacitance. $\mathbb{I}(\cdot)$ is an indicator function that equals 1 if the condition inside the parentheses is true and 0 otherwise.

Our experimental testing circuits include three multi-stage amplifiers (AMPs) from AnalogGym: NMCF, NMCNR, and DFCFC1, **with 11 optimization objectives and 5 constraints**. All experiments are conducted on a Linux workstation with 2 NVIDIA GeForce RTX 4080 GPUs and 4 Intel Xeon Platinum 8260 CPUs.

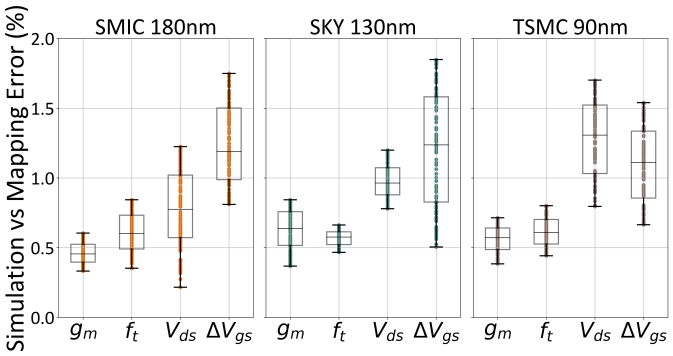


Fig. 5. Error comparison of transistor size mapping method with simulation across three process nodes. Each experiment was repeated ten times with different random seeds.

A. Evaluating Accuracy of Transistor Sizing Mapping

To evaluate the accuracy of the transistor sizing mapping method in Section III-B, we compared the mapping and simulation results across three different processes: SMIC 180 nm, SKY 130 nm [21], and TSMC 90 nm. We used data from the BCOA, selecting 1,000 samples for each process.

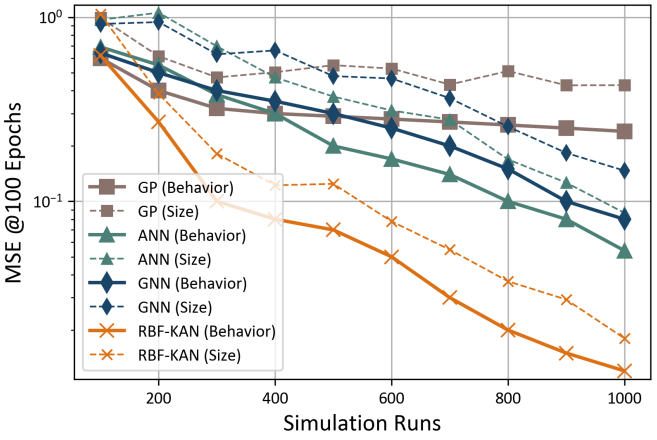


Fig. 6. Learning curves of average MSE over 10 runs for three AMPs.

Figure 5 shows that the mapping method achieves low errors, with g_m and f_t errors below 1% and V_{ds} and ΔV_{gs} errors slightly higher but within 2%. The higher errors for V_{ds} and ΔV_{gs} are due to their stronger sensitivity to device operating points and channel effects, which vary more significantly with process scaling. Overall, the mapping method maintains acceptable accuracy, demonstrating robustness for different process nodes.

B. Evaluating Nonlinear Fitting with Limited Samples

We conducted comparative experiments to evaluate the nonlinear fitting capability of our proposed RBF-KAN model. We used BagNet [22] to generate 10,000 samples. Then, by gradually increasing the training set, we evaluated the fitting capability of RBF-KAN under small-sample scenarios. Figure.6 shows mean squared error (MSE) at 100 epochs versus the number of simulation runs across different models: GP [6], ANN [2], GNN [23], and our RBF-KAN, tested for both behavior-based and transistor size-based modeling.

As shown in the figure, behavior-based modeling consistently achieves higher accuracy than size-based modeling. This is attributed to its decoupling of behavior from process dependence, thereby minimizing potential underfitting to the PDK. Compared to other models, RBF-KAN consistently achieves the highest accuracy. This is due to the Gauss-Laplace RBFs, which blend global smoothness and localized behavior capture. KAN's learnable activation functions also enable efficient handling of long-range feature interactions, even with minimal nodes. In many objective scenarios, GP requires covariance design to capture objective correlations, which compromises accuracy and extends processing times to over 10 hours due to joint modeling demands.

C. Comparison of BCOA and Other Methods

To demonstrate the effectiveness of the proposed BCOA framework, we compare FOM_{AMP} of BCOA with three state-of-the-art circuit sizing algorithms, including reinforcement learning based on relational GNN (rGNNRL) [5], constrained Voronoi tree search Bayesian optimization (cVTSBO) [15], and RL based on g_m/I_D (g_m/I_D RL) [24].

As shown in Table II, the improvements of BCOA primarily stem from significant enhancements in $\mathrm{FOM_L}$, $\mathrm{FOM_S}$, area, and Ts. Other methods utilize the single-objective optimization paradigm, treating the transient response and area as constraints. However, the sparse and narrow solution space for stable transient response, coupled with the lack of quantifiable feedback from oscillations, often leads to inefficient exploration and suboptimal solutions. Our framework employs the MC-HV method for efficient PF evaluation, dynamically adjusting reference points to guide many-objective optimization. As a result, BCOA significantly improves $\mathrm{FOM_L}$ by **1.76**× to **2.64**×, $\mathrm{FOM_S}$ by **1.73**× to **2.48**×, Ts by **1.31**× to **27.33**×, and reduces area by **15%-30%**. These individual improvements collectively contribute to the overall enhancement of FOM_{AMP}, which achieves gains ranging from **5.74**× to **98.05**× over g_m/I_D RL, **9.95**× to **176.60**× over cVTS, and **14.84**× to **184.37**× over rGNNRL.

TABLE III
COMPARISON OF HV AND MC-HV METHODS WITHIN THE BCOA

	Processing time (min) ↓			FOM_{AMP} ↑		
Topology	NMCF	NMCNR	DFCFC	NMCF	NMCNR	DFCFC
HV	226 ± 11	313 ± 23	241 ± 19	182.4 ± 42.1	137.9 ± 37.7	3952 ± 884
MC-HV	**6.1 ± 0.9**	**7.4 ± 1.2**	**6.8 ± 1.4**	**283.5 ± 21.4**	**186.2 ± 17.2**	**5863 ± 184**
Improvement	97.3%	97.6%	97.2%	1.55 ×	1.35 ×	1.48 ×

As shown in Figure 7, While cVTSBO uses domain decomposition to accelerate convergence with fewer function evaluations (FEs), it often gets trapped in local optima under complete specifications, limiting further improvement after initial convergence. We also observe that other methods significantly improve in FOM_{AMP} after integrating behavior-level optimization. This is due to behavior-level representation, which establishes a smoother decision space, enabling more efficient exploration.

As shown in Table III, the MC-HV method significantly improves runtime and optimization guidance over the HV method. Using Monte Carlo-based sampling for the PF estimation, MC-HV reduces complexity in many-objective scenarios, decreasing runtime by about **97%**. Adaptive reference point adjustments and targeted high-curvature sampling allow MC-HV to concentrate on suboptimal regions, enhancing convergence and solution diversity, with an average FOM_{AMP} improvement of **1.35**× to **1.55**× across the three topologies.

The effectiveness and efficiency of BCOA are attributed to two factors. 1) We decouple the analog block from the process using behavior-level representation and employ RBF-KAN to build accurate, comprehensive surrogate models with limited simulations. 2) Using the MC-HV method to approximate the Pareto front, we guide multi-objective trade-offs, enabling efficient global optimization.

D. Design Porting among Process Nodes

To validate the efficiency and effectiveness of behavior-level representation in design porting, we conducted experiments across 180nm, 130nm, and 90nm process nodes. Specifically, we compared BCOA with knowledge transfer-based methods, such as GCNRL [4], to evaluate FOM_{AMP} and FE consumption across nodes. In GCNRL, technology transfer is accomplished by training an RL agent to size circuits in a simpler node and then applying the same agent to new nodes.

We use process-independent device electrical properties to represent circuit behavior, enabling optimized designs to be reused across nodes. Specifically, design porting is achieved by mapping optimized behavior to transistor sizes using the LUT of the new process. Furthermore, the behavior-based surrogate model captures general electrical characteristics, enabling efficient knowledge transfer. As shown in Figure 8, BCOA reduces FE consumption by **6.6**× to **7.8**× compared to size-based knowledge transfer in GCNRL, achieving over **16.2** × improvement in FOM_{AMP} across multiple nodes.

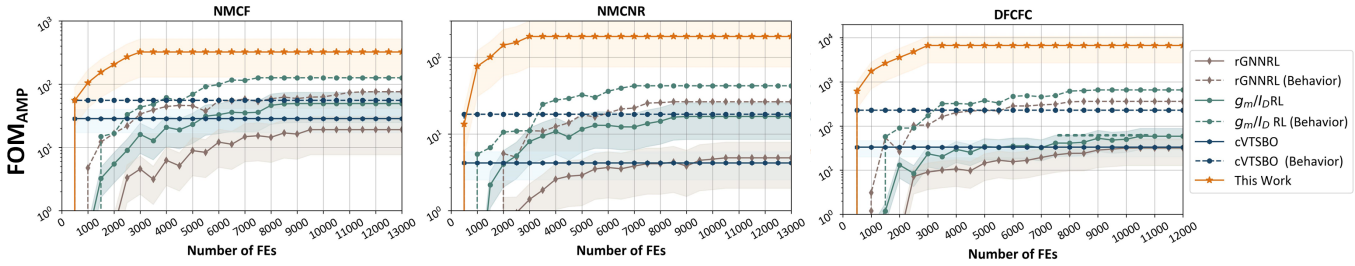


Fig. 7. Learning curves of different algorithms. The proposed BCOA consistently achieves the highest FOM_{AMP}.

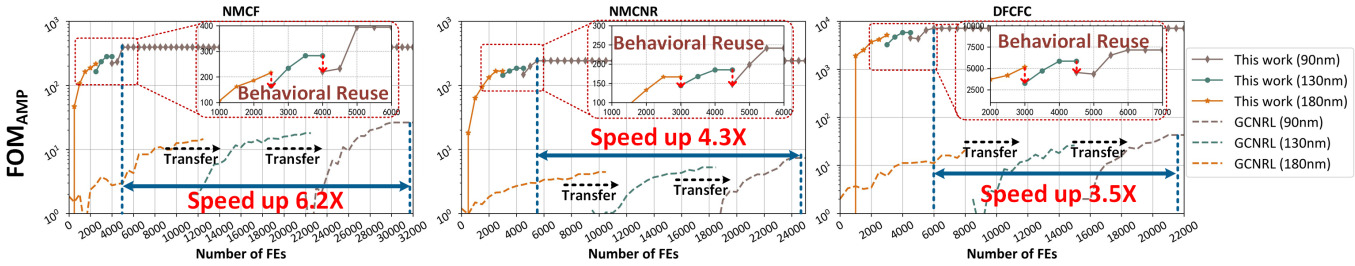


Fig. 8. Design porting among process nodes on AMPs. Behavioral representation enables cross-process reuse, while size-based representation demands substantial re-optimization at new node.

V. Conclusion

We present BCOA, a behavior-level optimization framework for analog blocks. We decouple the design from specific processes by representing circuit behavior with transistor electrical properties rather than size. This enables accurate surrogate modeling with RBF-KAN under limited simulations and facilitates global optimization. We also employ the MCHV method for rapid PF evaluation across complete specifications. Extensive experiments show that BCOA outperforms other methods in FOM_{AMP} and provides strong knowledge transfer ability. Therefore, BCOA offers a more effective and efficient approach to analog block optimization and design porting.

Acknowledgment

The project was supported by the National Key Research and Development Project of China (Grant No. 2023YFB4503002) and the Natural Science Foundation of China (Research Fund for International Senior Scientists, Grant No. W2431044).

References

[1] S. Maji, A. F. Budak, S. Poddar, and D. Z. Pan, "Toward end-to-end analog design automation with ml and data-driven approaches (invited paper)," in *Proc. ASPDAC*, 2024, pp. 657–664.

[2] A. F. Budak, M. Gandara, W. Shi, D. Z. Pan, N. Sun, and B. Liu, "An efficient analog circuit sizing method based on machine learning assisted global optimization," *IEEE TCAD*, vol. 41, no. 5, pp. 1209–1221, 2022.

[3] J. Li, H. Zhi, W. Shan, Y. Li, Y. Zeng, and Y. Li, "Multi-task evolutionary to PVT knowledge transfer for analog integrated circuit optimization," in *Proc. ICCAD*, 2023.

[4] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. DAC*, 2020, pp. 1–6.

[5] Z. Li and A. C. Carusone, "Design and optimization of low-dropout voltage regulator using relational graph neural network and reinforcement learning in open-source sky130 process," in *Proc. ICCAD*, 2023, pp. 01–09.

[6] W. Lyu, P. Xue, F. Yang, C. Yan, Z. Hong, X. Zeng, and D. Zhou, "An efficient bayesian optimization approach for automated optimization of analog circuits," *IEEE TCSI*, vol. 65, no. 6, pp. 1954–1967, 2018.

[7] J. Li, Y. Zeng, H. Zhi, J. Yang, W. Shan, Y. Li, and Y. Li, "Knowledge transfer framework for pvt robustness in analog integrated circuits," *IEEE TCSI*, vol. 71, no. 5, pp. 2017–2030, 2024.

[8] K. Hakhamaneshi, M. Nassar, M. Phielipp, P. Abbeel, and V. Stojanovic, "Pretraining graph neural networks for few-shot analog circuit modeling and design," *IEEE TCAD*, vol. 42, no. 7, pp. 2163–2173, 2023.

[9] P. Jespers, *The g_m/I_D Methodology, a sizing tool for low-voltage analog CMOS Circuits*, 2009.

[10] G. Shi, "Graph-pair decision diagram construction for topological symbolic circuit analysis," *IEEE TCAD*, vol. 32, no. 2, pp. 275–288, 2013.

[11] T. Gu, W. Li, A. Zhao, Z. Bi, X. Li, F. Yang, C. Yan, W. Hu, D. Zhou, T. Cui, X. Liu, Z. Zhang, and X. Zeng, "BBGP-sDFO: Batch bayesian and gaussian process enhanced subspace derivative free optimization for high-dimensional analog circuit synthesis," *IEEE TCAD*, vol. 43, no. 2, pp. 417–430, 2024.

[12] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "KAN: Kolmogorov-arnold networks," *arXiv preprint arXiv:2404.19756*, 2024.

[13] P. G. Jespers and B. Murmann, *Systematic design of analog CMOS circuits*. Cambridge University Press, 2017.

[14] T. Gu, J. Wang, Z. Bi, C. Yan, F. Yang, Y. Qin, T. Cui, and X. Zeng, "tSS-BO: Scalable bayesian optimization for analog circuit sizing via truncated subspace sampling," in *Proc. DATE*, 2024, pp. 1–6.

[15] A. Zhao, X. Wang, Z. Lin, Z. Bi, X. Li, C. Yan, F. Yang, L. Shang, D. Zhou, and X. Zeng, "cvts: A constrained voronoi tree search method for high dimensional analog circuit synthesis," in *Proc. DAC*, 2023, pp. 1–6.

[16] J. Li, H. Zhi, R. Lyu, W. Li, Z. Bi, K. Zhu, Y. Zeng, W. Shan, C. Yan, F. Yang, Y. Li, and X. Zeng, "Analoggym: An open and practical testing suite for analog circuit synthesis," 2024. [Online]. Available: https://arxiv.org/abs/2409.08534

[17] M. Stein, "Large sample properties of simulations using latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.

[18] B. Liu, F. V. Fernández, Q. Zhang, M. Pak, S. Sipahi, and G. Gielen, "An enhanced moea/d-de and its application to multiobjective analog cell sizing," in *Proc. CEC*, 2010, pp. 1–7.

[19] H. Wendland, *Scattered data approximation*. Cambridge university press, 2004, vol. 17.

[20] J. Baxter, "A bayesian/information theoretic model of learning to learn via multiple task sampling," *Machine learning*, vol. 28, pp. 7–39, 1997.
[21] "google/skywater-pdk," Aug. 2024, original-date: 2020-05-06T23:06:58Z. [Online]. Available: https://github.com/google/skywater-pdk
[22] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović, "Late breaking results: Analog circuit generator based on deep neural network enhanced combinatorial optimization," in *Proc. DAC*, 2019, pp. 1–2.
[23] K. Hakhamaneshi, M. Nassar, M. Phielipp, P. Abbeel, and V. Stojanovic, "Pretraining graph neural networks for few-shot analog circuit modeling and design," *IEEE TCAD*, vol. 42, no. 7, pp. 2163–2173, 2023.
[24] M. Choi, Y. Choi, K. Lee, and S. Kang, "Reinforcement learning-based analog circuit optimizer using gm/id for sizing," in *Proc. DAC*, 2023, pp. 1–6.

GEM: GPU-Accelerated Emulator-Inspired RTL Simulation

Zizheng Guo[1,3], Yanqing Zhang[2], Runsheng Wang[1,3], Yibo Lin[1,3], Haoxing Ren[2]
[1]School of Integrated Circuits, Peking University, Beijing, China
[2]NVIDIA Corporation [3]Institute of Electronic Design Automation, Peking University, Wuxi, China
{gzz,yibolin,r.wang}@pku.edu.cn, {yanqingz,haoxingr}@nvidia.com

***Abstract*—In this paper, we present a GPU-accelerated RTL simulator addressing critical challenges in high-speed circuit verification. Traditional CPU-based RTL simulators struggle with scalability and performance, and while FPGA-based emulators offer acceleration, they are costly and less accessible. Previous GPU-based attempts have failed to speed up RTL simulation due to the heterogeneous nature of circuit partitions, which conflicts with the SIMT (Single Instruction, Multiple Thread) paradigm of GPUs. Inspired by the design of emulators, our approach introduces a novel virtual Very Long Instruction Word (VLIW) architecture, designed for efficient CUDA execution. We also design a flow that maps circuit logic to the architecture in a process analogous to the FPGA CAD flow. This architecture mitigates issues of irregular memory access and thread divergence, unlocking GPU potential for RTL simulation. Our solution achieves up to 64× speed-up over the best CPU simulators, democratizing high-speed RTL simulation with accessible hardware and establishing a new frontier for GPU-accelerated circuit verification.**

I. Introduction

As design complexity of VLSI circuits scale aggressively, the complexity of verifying their functionality at the Register Tranfer Level (RTL) also rockets up. However, traditional CPU-based RTL simulators face scalability bottlenecks due to limited computation power, caches, and parallelism inside multi-core execution [1]. For large-scale designs, even high-performance CPU simulators struggle to deliver the speed necessary for rapid and iterative verification.

Field Programmable Gate Array (FPGA)- and custom-processor-based emulation has emerged as a powerful solution to accelerate logic verification, achieving orders-of-magnitude speedup over CPU simulation [2], [3]. By leveraging parallelism inherent in hardware, emulators can validate large designs more efficiently than software-based simulators. Despite their advantages, FPGA- and custom-processor-based emulators [4], [5] are costly in terms of both financial costs and setup time, limiting their accessibility and adaptability. With these barriers, GPU-accelerated simulation has the potential to offer a compelling alternative, given GPUs' high degree of parallelism, availability, and versatility. Yet, despite the evident advantages, GPUs are rarely used for simulation due to specific architectural challenges.

The barriers to GPU-based simulation arise from fundamental differences between circuit characteristics and GPU architectures. First, GPUs operate under a Single Instruction, Multiple Thread (SIMT) model where threads execute the same instructions in lockstep, which is optimal for workloads with homogeneous parallelism. In contrast, digital circuits are inherently heterogeneous; each circuit partition can have a unique graph structure and functional requirements that challenge SIMT compatibility [6]. Although prior work has sought to make circuits GPU-compatible by mapping them into LUT-based gate level structures [7], [8], [9], [10], [11], [12], this approach falls short in performance compared with highly optimized CPU simulators. Additionally, some GPU-based methods simulate multiple independent testbenches in parallel [13]. While this strategy improves simulation *throughput*, it cannot help in reducing *latency* which is critical for rapid turnaround in iterative design cycles.

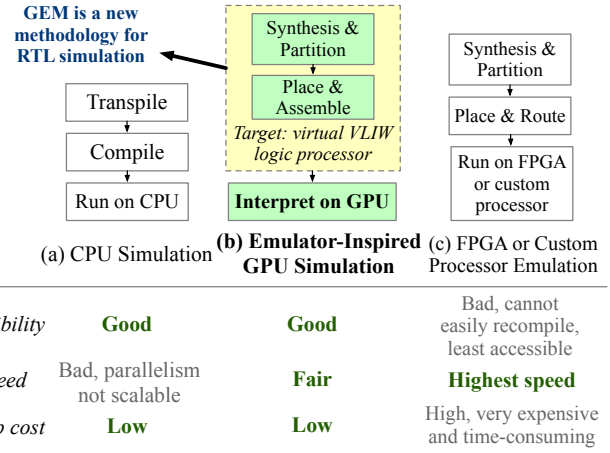


Fig. 1: GEM is a new, emulator-inspired methodology for flexible, fast, and low setup cost RTL simulation.

Another challenge lies in the irregular nature of circuit graphs, which complicates efficient data access on GPUs. GPU memory systems are optimized for throughput, relying on coalesced and regular memory accesses to achieve peak efficiency. However, circuit graphs are sparse and irregular, causing frequent irregular memory accesses that degrade GPU performance. These factors have, until now, prevented GPUs from being viable platforms for efficient, low-latency circuit simulation.

In this work, we propose a GPU-accelerated RTL simulator, **GEM**, that overcomes these challenges and redefines the potential of GPU-based circuit verification. The major contributions are summarized as follows.

1) Rather than simulating circuits using LUT-based methods, we introduce a technique that maps circuit logic into a specialized virtual Boolean processor with a Very Long Instruction Word (VLIW) architecture, optimized for execution on CUDA-compatible GPUs. This innovative architecture allows us to fully utilize the GPU threads even for irregular circuit structures and enabling coalesced memory accesses that are critical to high-performance GPU operation.
2) We design a mapping flow from RTL to our virtual VLIW architecture that mimics the partitioning, synthesis, and physical design flow of hardware, enabling us to interpret the circuit functionality on GPU in a way similar to the computer-aided design (CAD) flow for FPGAs.
3) In our mapping flow, we address various critical challenges on simulation performance by designing a set of novel algorithms, including (i) deep long-tailed logic mapping that minimizes synchronization, (ii) RAM mapping and depth-optimized extended AIG synthesis, (iii) width-constrained replication-efficient partitioning, and (iv) iterative timing-driven bit placement.

Our GPU-accelerated simulator achieves a remarkable 38× and

64× speed-up compared to a leading commercial tool and Verilator, respectively. This substantial performance improvement validates our approach, demonstrating a successful use of GPUs in delivering RTL simulation speed-up. GEM offers a paradigm shift in GPU-based circuit verification, with potential applications extending into areas such as hardware-software co-design and rapid prototyping. Furthermore, this software-based method democratizes high-speed RTL simulation by making it accessible on readily available and relatively inexpensive GPU hardware, bringing high-speed simulation to a broader audience of designers and researchers. To this end, we made GEM open source under a permissive Apache license[1].

II. Preliminary

In digital circuit design, simulation and emulation serve as crucial processes for verifying and evaluating functionality before hardware manufacturing. Both approaches typically work in two stages: compilation and execution.

- In the *compilation* stage, the behavioral RTL netlist, represented in languages such as Verilog or SystemVerilog, is processed and transformed into an executable simulator. This simulator models the circuit behavior, either in software or as a hardware-mapped design, which will then be used to execute various test scenarios.
- The *execution* stage involves running the compiled simulator with specific input stimuli, provided as waveforms or recorded signal patterns (e.g., VCD or FSDB format). The simulator or emulator responds to these stimuli, producing output that reveals the circuit's expected behavior, helping designers validate functionality across test cases.

One key performance metric for evaluating simulators and emulators is the simulation speed, commonly measured in *simulated cycles per second* or Hertz (Hz). Depending on the underlying methodology, simulators are typically categorized as either oblivious/full-cycle simulators or event-based simulators [1].

- Full-cycle simulators process the entire circuit in each simulation cycle, regardless of whether specific parts of the circuit experience any activity. This approach is suitable for high-throughput FPGA-based emulators because the process is consistent across cycles.
- Conversely, event-based simulators, typically CPU-based, are optimized for efficiency by selectively updating only the circuit elements that are actively switching or affected by input changes in each cycle. This selective processing can save significant computation time when only parts of the circuit change, but it is generally slower than hardware-accelerated full-cycle methods.

Previous work in logic emulation and simulation has explored various hardware and software platforms, each with distinct advantages and limitations. FPGA- or custom-processor-based emulation is known for its impressive performance, often achieving emulation speeds of 1 MHz or more [4], [5]. However, the significant cost of dedicated FPGA hardware and the extensive compilation times make this approach less accessible and time-efficient for iterative design workflows. For example, it can take days to compile a design into a FPGA-based emulator. In contrast, CPU-based simulators offer flexibility and ease of setup but are generally slow, often failing to scale well with increasing circuit complexity.

Efforts to parallelize CPU-based simulators, primarily through circuit partitioning, the Chandy-Misra-Bryant (CMB) algorithm, and dataflow optimization [14], [15], [16], [17], have met with only moderate success. Although these approaches can exploit multicore CPUs, they face significant bottlenecks: the CPU frontend often becomes overwhelmed by the sheer volume of compiled circuit code, and parallelism is constrained by the limited memory bandwidth in standard multicore CPUs, capping the potential performance gains.

Efforts to leverage GPU's data parallelism for logic simulation have aimed to address the performance gap but have generally been confined to gate-level simulation using lookup-table-based methods [7], [8], [9], [10], [11], [12]. This approach relies on GPU threads querying precomputed truth tables, which align with the SIMT architecture of GPUs. However, this method presents serious limitations: gate-level simulations are typically much slower (10–100×) than RTL-level simulations, as the latter operate on a higher abstraction level. Consequently, while previous GPU-based methods show promise at the gate level, they fail to match the performance needed for practical RTL verification tasks.

Another line of GPU-accelerated simulation instead transpiles RTL directly to CUDA code. As this results in SIMT-incompatible GPU kernels, they either choose to forfeit data parallelism completely [6] or use independent workloads to fill the data parallelism dimension [13]. These methods require special inputs such as independent stimulis or simple blocking RTL code and are thus not good at reducing simulation latency which is critical in verification turnaround.

III. Algorithm

We present a novel GPU-accelerated RTL simulator, GEM, that addresses the fundamental mismatch between circuit simulation and GPU architecture. Our core idea is a virtual Boolean processor that acts as a reconfigurable and highly-parallel container of Boolean logic. This virtual Boolean processor is programmed with a VLIW instruction set, and is designed to fit GPU parallelism. Specifically, its bitstream can be interpreted using high performance GPU kernels on a CUDA-compatible GPU. Section III-A describes the key motivation behind the design of this architecture and how it solves the SIMT and memory access challenges. To map any input RTL design to the virtual Boolean processor, we design a process containing synthesis (Section III-B), partitioning (Section III-C), physical design (Section III-D), and finally bitstream generation and interpretation (Section III-E). This flow has a beautiful analogy to the CAD flow of FPGAs.

A. A Virtual Boolean Machine

GEM introduces a GPU-friendly virtual Boolean machine that is critical in bridging the gap between circuit functionality and GPU execution. Following are our key observations that lead to its design.

Observation 1. Every Boolean function can be implemented using only a fixed set of operators. Such an operator set is called functional complete. For example, {AND, INVERT} is a functional complete operator set. By compiling the logic into an intentionally limited set of operators, we can use GPU's built-in logic instructions to execute them, instead of having to query look-up tables from memory.

Observation 2. Irregular memory access is inherently unavoidable due to the heterogeneous nature of general circuit graph. However, irregular access inside GPU shared memory (only accessible by a local thread block) is much less costly than irregular global memory access. Rearranging the simulation so that most irregular accesses happen locally inside a thread block will thus help greatly in reducing memory overhead.

Based on Observations 1–2, our modeling of simulation is shown in Figure 2. We regard the general simulation task as a set of logic

[1]https://github.com/NVlabs/GEM

partitions, each in a reasonable size that can be handled with one Boolean processor core (i.e., one GPU thread block). Inside each partition there is an extended and-inverter graph (E-AIG) with AND gates, INVERT gates, D flip-flops (FFs), and RAM blocks. E-AIG can efficiently represent any synthesizable circuit with combinational and synchronous sequential logic. We note that although RAM blocks can be polyfilled using only FFs and decoder logic, this process is extremely costly for large RAMs and thus we introduce native support to RAM blocks in E-AIG.

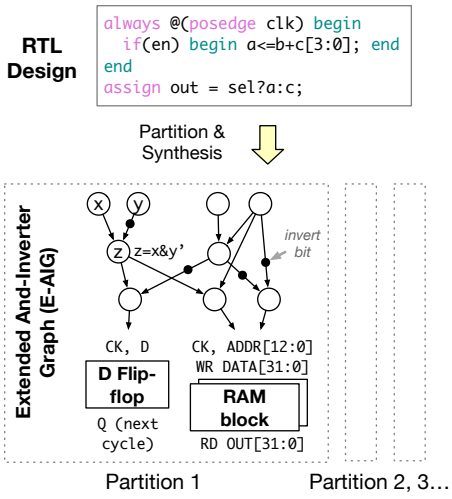


Fig. 2: GEM regards every RTL design as a set of partitions. Each partition is an extended and-inverter graph (E-AIG).

Inside the E-AIG, most of the computation lies in the combinational logic AIG. Our Boolean processor should support efficient execution of AIGs by exploiting the simplicity of gate choices and the common AIG properties.

Observation 3. In addition to thread-level parallelism, *word-level parallelism* can provide orders-of-magnitude more logic processing power. For example, suppose `a,b,c` are three 32-bit unsigned integers, calculating `r=(a AND b) XOR c` is performing 32 And-then-Invert instructions in parallel using `a,b` as inputs and `c` as a constant that encodes whether to flip each resulting bit.

Observation 4. The logic depth of an AIG can be 50–100 for common circuits. However, the gate distribution among the logic levels is extremely imbalanced. A large portion of the gates reside in a few frontier levels whereas only a few gates are accountable for the rest of the levels. We call this the *long-tailed* nature of circuit graphs.

Levelization-based GPU algorithms are frequently used to process circuit graph. They divide the circuit into a series of logic levels each containing a batch of independent calculations. Between two consecutive logic levels, there needs to be a bulk synchronization and a permutation to align the level outputs with the next-level's inputs. As a result, Observation 4 turns out to be very harmful because it leaves most of the levels underutilized and incurs large synchronization overhead. In light of Observations 3–4, we propose a boomerang-shaped executor layer as shown in Figure 3. It is the central reconfigurable logic executor in each virtual Boolean processor core, consisting of interleaved bit permutations and boomerang layers. The virtual Boolean processor core maintains up to 8,192 bits of circuit states. Inside each boomerang layer, the 8,192 bits are recursively folded using bitwise AND and then bitwise XOR with an external constant. The fold is repeated by 14 times until we get a single bit. It can be observed that even a single boomerang layer is able to simulate up to 14 levels of logic. Their shape also fits nicely with the long-tailed nature of circuit graph. Experimentally, boomerang layer reduces the number of bit permutations and synchronizations inside a GPU thread block by more than 5×.

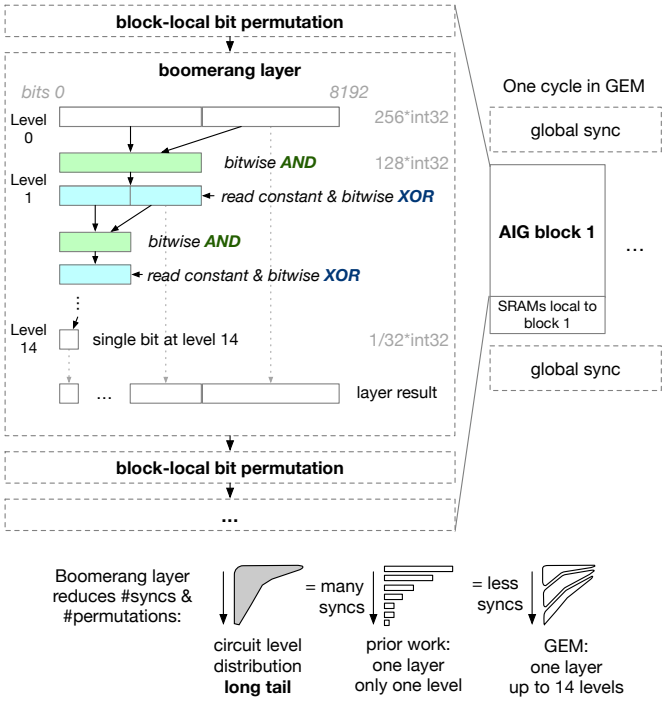


Fig. 3: The boomerang-shaped executor layer in GEM can greatly reduce the number of bit permutations and synchronizations for deep and long-tailed logic.

B. Synthesis

To compile an RTL design for GEM execution, our first step is to map it to the E-AIG format. While AIG is a widely used format in logic synthesis research, there is no existing flow that synthesizes RTL to *extended* AIG with clocked FFs and RAM blocks. The main challenges are two fold: (1) The behavioral RAM constructs in RTL netlist need to be identified and mapped to our fixed RAM block type (13 bits address and 32 bits data) to minimize thread divergence. For a general RAM in RTL, multiple such RAM blocks need to be instantiated and adapter logics need to be introduced automatically. (2) As we will show in later sections, our simulator requires high-quality synthesis results in order to execute efficiently. Specifically, we require the depth of AIG to be as low as possible.

To meet the above requirements, we develop a synthesis flow that exploits two existing synthesis flows for FPGA and ASIC, as shown in Figure 4. We use open-source Yosys synthesizer [18] to deal with RAM mapping. We create a fake FPGA target platform and define our available RAM block. Yosys will take the definition and handle the RAM mapping. Then without further LUT mapping, we can write out the intermediate RTL netlist that only has the RAMs mapped. Next, we handle the RAM-mapped RTL with another ASIC synthesis flow. This time we give the synthesizer a fake ASIC library that only contains AND, OR, INV, and FF gates. This fake library has a simple timing model that defines AND and OR gate delays as

1ps and INV gates as 0ps. Timing-driven synthesis is thus equivalent to depth optimization. In our experiments, we found commercial ASIC synthesizers outperform Yosys in this second step.

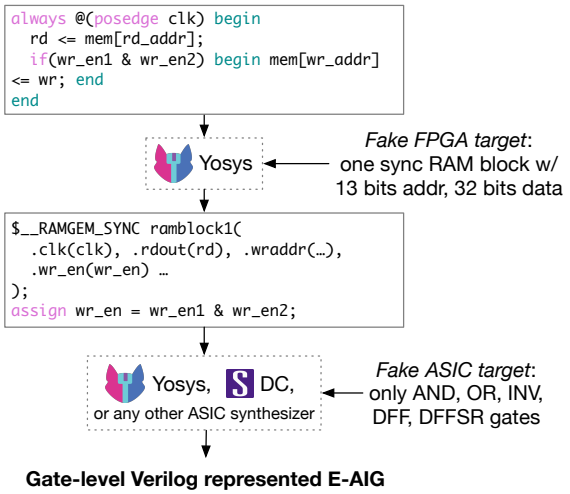


Fig. 4: We exploit existing FPGA and ASIC synthesis flows to efficiently transform RTL design into E-AIG.

C. Partitioning

Partitioning the hardware to multiple Boolean processor cores is a crucial step in our simulator because it ensures locality for data movement. As GPUs do not have efficient inter-block communication, we should ideally make partitions independent of each other and only communicate once per simulated cycle. The goal is made possible with a recent CPU-based parallel simulator RepCut [17], whose basic idea is to allow some duplicated logic in order to remove inter-partition dependency. To adapt the idea to a GPU simulator, we face two new challenges.

Firstly, the number of GPU thread blocks is much larger than the number of CPU cores. However, the replication cost (i.e., the relative size of duplicated logic over the original circuit size) grows quickly with the increased granularity of partitions. For example, RepCut [17] reports that only 1.30% cost is needed to partition a design into 8 threads, but the cost rises to 10.95% when 48 threads are used. In our experiments, we found this cost quickly surges to over 200% when we have to partition a design into 216 blocks, which is a minimal requirement to fully utilize a modern GPU.

To scale RepCut to over 200 partitions, our solution is to extend it to multiple stages as shown in Figure 5. We cut the circuit graph at one or more levels in the middle. We treat the nodes at the cut level as endpoints and run RepCut separately for each stage. We found that with the cost of 1 additional synchronization, we reduce the replication cost from 200% to less than 3% when partitioning a 500K gates design into 216 blocks. For even larger designs, more stages might be needed but it is easy to strike a balance between synchronization and replication costs with heuristics.

Secondly, the goal of partitioning to Boolean processors is different from the original RepCut, which aims at balancing the size of partitions. Instead, we require that all partitions are mappable to the boomerang-shaped executor layer. This effectively constrains the *width* (8192 bits, see Figure 3) instead of *total size* of partitions. It is difficult to modify a hypergraph partitioner's objective to logic widths as this metric does not have nice additive property. Instead, we make no change to the partitioner itself and keep the original size

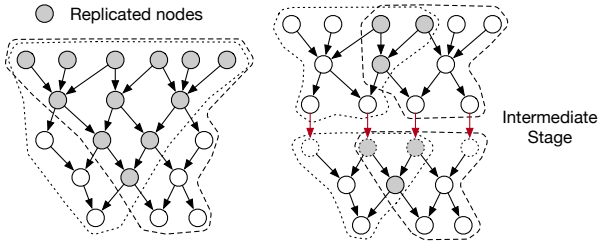


Fig. 5: By introducing one additional stage, the logic duplication of RepCut can be remarkably reduced, unlocking enough parallelism for GPU utilization.

objective. We run an additional postprocessing algorithm after the partitioner to align the objectives to our logic width need. We show our idea in Algorithm 1, which is based on empirically trying to merge the resulting partitions after a round of excessive partitioning. After running the algorithm, it is easy to guarantee that each partition has at least 50% effective bit utilization.

Algorithm 1: Partition merging

```
Partition the design excessively so that each partition is
  mappable;
for each partition p do
    Sort other unvisited partitions by overlap size with p;
    for partition q with large-to-small overlap do
        Try merging q with p, if the result is mappable,
          commit the merge;
```

D. Logic Placement

Given an E-AIG partition, our logic placement algorithm implements the logic onto a series of reconfigurable boomerang layers (Figure 3). Our objective is to map all AIG nodes while reducing the total number of boomerang layers required. In this section, we present an iterative timing-driven placement algorithm in GEM to map AIG to boomerang layers.

We show our basic idea of bit mapping in Figure 6 using a 4-level small boomerang layer as an example. We start with an empty boomerang layer as Figure 6 (1). Then, we choose a bit from E-AIG at logic level 4 and map it to the empty slot in boomerang executor at level 4. To do so, we recursively map the inputs of the being-mapped bit to upper levels of the executor. In the worst case, the fan-in cone of the being-mapped bit forms a perfectly balanced binary tree, thus occupying all bits of a boomerang layer. However, such worst case never happens in practice, as most bits in AIG have a pair of inputs that reside in different logic levels (i.e., imbalanced). As a result, there leaves a lot of free space after mapping one bit, and such vacant bits can be used to map other bits as shown in Figure 6 (2)–(4).

Based on the above bit mapping primitive, we use Algorithm 2 to map a whole AIG partition into a series of boomerang layers. We iteratively try to map the fan-in frontier of current AIG subgraph into a new empty layer. After the layer is full, we remove the nodes that are already realized by current layers, create a new AIG subgraph of remaining nodes, and repeat the process until all nodes are mapped. We define timing criticality of a node as its logic depth on the reversed AIG subgraph. To reduce the number of layers, we prioritize the mapping of nodes on timing-critical paths (Figure 6 (5)

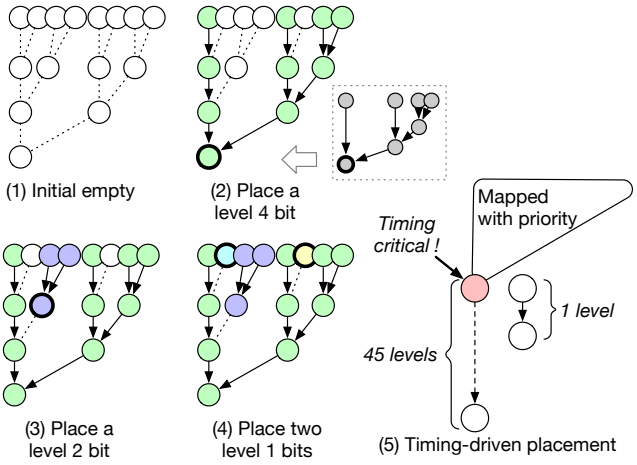


Fig. 6: Example of a placement iteration (1)–(4) and illustration of timing-driven bit placement (5).

and Algorithm 2 lines 7–8) according to the constantly updated timing criticality of nodes.

Algorithm 2: Iterative multi-boomerang-layers mapping

```
1  Input AIG;
2  Initialize node set frontier as input ports of the AIG;
3  while AIG not all mapped do
4      Make a new empty boomerang layer;
5      for Level i in boomerang layer from bottom to top do
6          while Level i is not full do
7              Update timing criticality (reverse logic depth) of
                the remaining AIG;
8              Choose the most timing-critical unmapped node
                in AIG with logic level i, and map it to the
                boomerang layer;
9      Add the boomerang layer to the list of layers;
10     Update frontier to the input ports of the current
        unrealized AIG subgraph;
```

E. Bitstream Generation and CUDA Interpretation

The synthesized and placed circuit from previous sections are then transformed into a binary format that will be loaded and interpreted by our GEM CUDA kernel. In one way, this process is analogous to the bitstream generation of FPGA designs as we are encoding and serializing the wiring (boomerang layer setups) of a reconfigurable hardware. In another way, this process might also be called a binary assembler of software programs, as the resulting bitstream will be interpreted on a GPU by a software-only approach like inside a virtual machine.

We design a domain-specific instruction set architecture (ISA) for GEM to assemble the Boolean processor programs. This is a VLIW ISA that has 3 instruction length variations: 8192, 16384, and 32768 bit. It is designed for a GPU thread block with fixed 256 threads to load and interpret with high throughput. A 8192-bit GEM ISA instruction is loaded by 256 threads in lockstep by performing one coalesced 32-bit global memory read. For 16384- or 32768-bit instructions, 256 threads similarly perform a coalesced 64- or 128-bit global memory read, respectively. All these memory loads are fully coalesced and natively supported in CUDA.

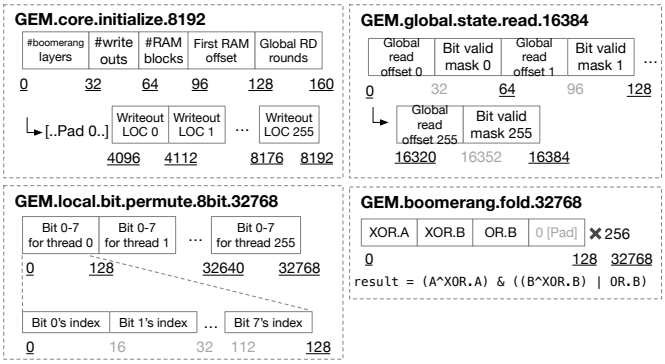


Fig. 7: Instruction bit layout: initialization, global state reading, local bit permutation, and boomerang folding.

TABLE I: Design statistics and GEM mapping results.

Design	#E-AIG Gates	#Levels	#Stages	#Layers	#Parts	Bitstream
NVDLA	668,746	62	1	9	52	11.2 MB
RocketChip	346,687	82	1	13	39	9.2 MB
Gemmini	1,831,381	148	1	19	143	44.4 MB
OpenPiton1	682,646	66	2	10	119	18.4 MB
OpenPiton8	5,479,795	66	2	13	947	162.4 MB

Due to page limit, we cannot include a complete ISA documentation. Figure 7 shows some examples of instruction bit layout. 8192-bit instruction is used to initialize a thread block with necessary information, including the number of boomerang layers, RAMs, and state size of a partition. This is followed by a few 16384-bit instructions to read the input ports of the E-AIG from global memory, only once per cycle. Then, bit permutations and boomerang folding constants are provided with interleaved 32768-bit instructions. The bit permutations are encoded as a compressed form of source bit locations indexed within 8192 thread-local bits in GPU shared memory. The boomerang folding provides 3 constants, `XOR.A`, `XOR.B`, `OR.B` for each thread, controlling the behavior of the word-parallel AND gates. The `OR.B` is used to bypass operand B to implement dashed lines in Figure 6 (4).

In the CUDA kernel implementation, we note two important optimizations. The first is the use of wide global read instructions and aligned structs that allow coalescing to work correctly. The second is the use of cooperative groups [19] to implement device-level synchronization in cycle and stage boundaries in order to bypass kernel launching overhead.

IV. Experimental Results

We implement GEM from scratch in Rust (for the mapping flow) and CUDA (for the VLIW interpreter kernel). We evaluate GEM's performance on a variety of open-source RTL benchmarks listed in Table I, including NVDLA [20] (a deep learning accelerator), RocketChip [21] (a RISC-V CPU), Gemmini [22] (another deep learning accelerator), and an internal multi-core CPU design derived from OpenPiton [23]. For Chisel designs like RocketChip and Gemmini, we generate RTL using Chipyard [24]. We use official benchmark workloads provided by these designs for evaluation.

Table I lists the size of the designs. To estimate the actual RTL circuit footprint, we show the number of logic gates and logic levels after our GEM synthesis (Section III-B). Table I also lists the statistics after finishing the GEM mapping flow, including the number of RepCut stages, boomerang layers, partitions, and the size of the bitstream. The number of boomerang layers is 6–8× smaller

TABLE II: Simulation speed (Hz) and speed-up (compared to GEM-A100) comparison between GEM, a commercial tool (Comm.), Verilator, and GL0AM. NVDLA only simulates correctly with an old 3.912 version of Verilator without multi-threading [20].

Design	Test Name	Comm.	Verilator		GL0AM	GEM		Comm./GEM	Verilator/GEM		GL0AM/GEM
			8 Threads	1 Thread	A100	A100	3090		8 Threads	1 Thread	
NVDLA	dc_...6x3x76x270_int8_0	2,956	N/A	1,010	2,175	65,385	55,716	22.12	N/A	64.76	30.06
	dc_...6x3x76x16_int8_0	4,712	N/A	1,060	3,534	65,385	55,716	13.88	N/A	61.69	18.50
	img_51x96x4..._int8_0	7,848	N/A	1,169	8,213	65,385	55,716	8.33	N/A	55.93	7.96
	cdp_8x8x32_lrn3_int8_2	1,683	N/A	1,512	7,443	65,385	55,716	38.85	N/A	43.24	8.79
	pdp_...max_int8_0	3,391	N/A	1,555	8,353	65,385	55,716	19.28	N/A	42.04	7.83
RocketChip	dhrystone	7,262	9,517	4,639	7,275	52,403	51,695	7.22	5.51	11.30	7.20
	mt-memcpy	11,672	8,845	4,790	6,584	52,403	51,695	4.49	5.92	10.94	7.96
	pmp	4,955	8,220	4,529	6,034	52,403	51,695	10.58	6.38	11.57	8.68
	qsort	6,764	8,342	4,657	7,142	52,403	51,695	7.75	6.28	11.25	7.34
	spmv	11,305	7,534	4,719	7,420	52,403	51,695	4.64	6.96	11.10	7.06
Gemmini	tiled_matmul_ws_full_C	5,188	9,638	2,460	11,618	25,608	17,889	4.94	2.66	10.41	2.20
	tiled_matmul_ws_perf	13,205	10,554	2,537	13,227	25,608	17,889	1.94	2.43	10.09	1.94
OpenPiton1	ldst_quad2	13,871	5,355	3,415	8,400	36,583	31,339	2.64	6.83	10.71	4.36
	fp_mt_combo0	10,569	5,402	3,358	7,303	36,583	31,339	3.46	6.77	10.90	5.01
	asi_notused_priv	5,167	5,025	3,157	4,624	36,583	31,339	7.08	7.28	11.59	7.91
OpenPiton8	ldst_quad2	4,820	1,078	315	5,172	7,285	4,694	1.51	6.76	23.14	1.41
	fp_mt_combo0	7,666	1,080	316	7,203	7,285	4,694	0.95	6.74	23.06	1.01
	asi_notused_priv	1,441	1,004	306	1,920	7,285	4,694	5.05	7.25	23.85	3.79
Average Speed-up				-				9.15	5.98	24.87	7.72

than the logic depth (e.g., reduced from 148 to 19 for Gemmini). We note that the GEM bitstream is a very concise format for circuit logic. It takes only 162.4 MB of GPU memory to store the whole assembled GEM bitstream (Section III-E) even for our largest design OpenPiton8 which has over 5 million logic gates and is over 800 MB in flattened gate-level Verilog. As a result, even lowest-end GPUs have enough GPU memory to simulate large designs with GEM.

We compare the performance of GEM against a set of strong baselines, including a leading commercial tool[2], Verilator latest 5.028 [14], and the current state-of-the-art GPU-accelerated gate-level logic simulator GL0AM [12]. All experiments are run on 48 cores of Intel Xeon Gold 6136 CPU. We run Verilator with up to 8 threads as we observe that 16-threaded Verilator is only 80%–95% the speed of 8 threads. This performance *degradation* highlights the inherent scalability bottleneck of CPU parallelism on highly-complex real designs as also identified by prior works. We run the commercial tool with default single core similar to the settings in [12], [11] as multi-threaded version is unstable and crashes during some simulations. We evaluate GEM on both one NVIDIA A100 and one NVIDIA RTX 3090. A100 demonstrates GEM's current peak performance and 3090 shows its performance with a more accessible GPU alternative.

Table II gives a comprehensive performance comparison. We are on average 9.15×, 5.98×, and 24.87× faster than the leading commercial tool, 8-threaded Verilator and 1-threaded Verilator respectively. The peak speed-ups happen on the deep-learning accelerator NVDLA where we are 64.76× and 38.85× faster than Verilator 1 thread and the commercial tool. GEM on 3090 has comparable performance with A100 except on the largest design, OpenPiton8, with the highest resource pressure on GPU.

The current version of GEM has **limitations** which we plan to address in the future. For example, we found that the speed-up ratio of OpenPiton8 is inferior to OpenPiton1. This is because the workload of OpenPiton8 does not keep all 8 cores busy. Reported by the commercial tool, we observe 8,612 signal events per cycle for 1 core but only 28,789 events (3.3×) for 8 cores. As an oblivious full-cycle simulator, GEM has a consistent simulation speed for any stimuli, whereas our baselines are event-based simulators that run faster if the design is not actively switching. In the future, we plan to explore event-based pruning in GEM. NVDLA shows the best speed-up GEM can achieve because all RAMs inside it are mapped to E-AIG RAM blocks, but the other 4 designs have RAMs with *asynchronous* read ports that can only be implemented inefficiently with FFs and decoder logic. As asynchronous RAMs are not available in ASIC or FPGA deployment, we regard NVDLA's performance as representative for real-world designs.

[2]We do not disclose the name of the commercial tool due to license agreements.

V. Conclusion

This paper presents GEM, a novel emulator-inspired methodology for GPU-accelerated RTL simulation. GEM bridges the fundamental discrepancy between GPU's SIMT data parallelism and the heterogeneous and irregular circuit logics, by designing a CUDA-interpretable virtual VLIW logic processor and a mapping flow analogous to the CAD flow of FPGAs, with novel algorithms addressing deep long-tailed logic, RAM mapping, width-constrained partitioning, timing-driven bit placement, etc. GEM achieves superior performance compared to cutting-edge commercial and open-source simulators.

The framework of GEM is extensible and many improvements are possible as future works, including native arithmetic operations, multi-GPU support, CUDA software pipelining, 4-state simulation, etc. We will open-source GEM for democratizing high-performance RTL verification as well as fostering further research.

VI. Acknowledge

This work is supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002), the National Natural Science Foundation of China (Grant No. T2293701), and the 111 project (B18001).

REFERENCES

[1] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[2] M. D. Moffitt, G. E. Günther, and K. A. Pasnik, "Place and route for massively parallel hardware-accelerated functional verification," in *Proc. ICCAD*, 2013, pp. 466–472.

[3] W. N. Hung and R. Sun, "Challenges in Large FPGA-based Logic Emulation Systems," in *Proc. ISPD*. Monterey California USA: ACM, 2018, pp. 26–33.

[4] "Synopsys ZeBu Server," https://www.synopsys.com/verification/emulation.html.

[5] "Cadence Palladium," https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html.

[6] H. Qian and Y. Deng, "Accelerating RTL simulation with GPUs," in *Proc. ICCAD*. San Jose, CA, USA: IEEE, 2011, pp. 687–693.

[7] D. Chatterjee, A. DeOrio, and V. Bertacco, "GCS: High-performance gate-level simulation with GPGPUs," in *Proc. DATE*. IEEE, 2009, pp. 1332–1337.

[8] D. Chatterjee, A. Deorio, and V. Bertacco, "Gate-level simulation with GPU computing," *ACM TODAES*, vol. 16, no. 3, pp. 1–26, 2011.

[9] Y. Zhu, B. Wang, and Y. Deng, "Massively parallel logic simulation with GPUs," *ACM TODAES*, vol. 16, no. 3, pp. 1–20, 2011.

[10] Y. Zhang, H. Ren, A. Sridharan, and B. Khailany, "GATSPI: GPU accelerated gate-level simulation for power improvement," in *Proc. DAC*. IEEE, 2022.

[11] Z. Guo, Z. Zhang, X. Jiang, W. Li, Y. Lin, R. Wang, and R. Huang, "General-purpose gate-level simulation with partition-agnostic parallelism," in *Proc. DAC*. ACM, 2023.

[12] Y. Zhang, H. Ren, and B. Khailany, "GL0AM: GPU Logic Simulation Using 0-Delay and Re-simulation Acceleration Method," in *Proc. ICCAD*. New York, NY, USA: IEEE, 2024, pp. 1–9.

[13] D.-L. Lin, H. Ren, Y. Zhang, and T.-W. Huang, "From rtl to cuda: A gpu acceleration flow for rtl simulation with batch stimulus," in *Proc. ICPP*, 2022.

[14] W. Snyder, "Verilator 4.0: open simulation goes multithreaded," in *Open Source Digital Design Conference (ORConf)*, 2018.

[15] S. Beamer and D. Donofrio, "Efficiently Exploiting Low Activity Factors to Accelerate RTL Simulation," in *Proc. DAC*. San Francisco, CA, USA: IEEE, 2020, pp. 1–6.

[16] K. Zhou, Y. Liang, Y. Lin, R. Wang, and R. Huang, "Khronos: Fusing Memory Access for Improved Hardware RTL Simulation," in *Proc. MICRO*. Toronto ON Canada: ACM, 2023, pp. 180–193.

[17] H. Wang and S. Beamer, "RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning," in *Proc. ASPLOS*. ACM, 2023, pp. 572–585.

[18] "Yosys." [Online]. Available: https://yosyshq.net/yosys/

[19] M. Harris and K. Perelygin, "Cooperative groups: Flexible cuda thread programming," 2017. [Online]. Available: https://developer.nvidia.com/blog/cooperative-groups/

[20] "Nvidia deep learning accelerator," 2017. [Online]. Available: https://nvdla.org/

[21] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," UC Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[22] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. DAC*, 2021.

[23] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "Openpiton: An open source manycore research framework," *SIGARCH Comput. Archit. News*, vol. 44, no. 2, p. 217–232, 2016.

[24] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

Design and Technology Co-optimization Utilizing Flip-FET (FFET) Standard Cells

Jaehoon Ahn and Taewhan Kim
Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea
jhahn97@snucad.snu.ac.kr, tkim@snucad.snu.ac.kr

***Abstract*—With the continued scaling of VLSI technology beyond 3nm, a consistent demand for layout reduction in standard cells has been made. CFET (Complementary-FET) has been accepted as a promising device technology, stacking N-FET on P-FET (or vice versa) to achieve this goal while providing metal interconnects on both the front and backside of the wafer through BEOL (back-end-of-line) processing. However, the layout synthesis of CFET based standard cells and its use in physical design implementation are not fully compatible with the effective exploitation of backside interconnects. This is because of a considerable overhead on the allocation of special vias in FEOL (front-end-of-line) referred to as *tap-cells*, which are essential for the net routes using backside wire. To overcome this drawback of using CFET cells, a new technology called FFET (Flip-FET) has been proposed, which *flips* the lower FET in CFET to enable direct pin accessibility on both sides of BEOL with no tap cells. In this context, *we propose an FFET cell based DTCO methodology to fully utilize backside wires with minimal tap-cells*. Precisely, we propose a three-step approach: (1) *synthesizing multiple FFET standard cells with diverse styles of pin distribution* for each of primitive logic gates, (2) *performing a tap-cell avoiding cell replacement* during placement optimization by using the cells obtained from Step 1 to prevent tap-cell allocation for net routing, and (3) *rebalancing the wire usage between the frontside and backside* to mitigate net congestion. Experiments with benchmark circuits show that our FFET cell based DTCO methodology scales up chip size beyond CFET designs with much fewer routing failures.**

Keywords: CFET, FFET, standard cells, DTCO, physical design

I. Introduction

With aggressive advancement in technology nodes which drives FinFET to reach its structural limits in scaling, interest has shifted toward vertically stacked 3D architectures for further scaling. One noticeable technology innovation is Complementary FET (CFET) [1], which physically stacks N-FET on top of P-FET or vice versa (e.g., Fig. 1(b)), for which installing backside power delivery network (BSPDN) with buried power rails (BPR) [2] is essential to secure a few more metal tracks available at the frontside for net routing.

Despite the use of BSPDN, chip routing for designs with CFET based standard cells still suffers a considerable shortage of metal resources. To tackle this issue, a recent work [3] and TSMC patent [4] suggested using backside wires not only for BSPDN but also for signal net routing.

However, for chip-level double-sided routing, whenever a net uses a backside wire to make a route, it is unavoidable to allocate a special via called *tap-cell* to connect the net pins to the backside wire since all cell pins are exclusively located on a metal layer at the frontside, Thus, if a balanced use of double-sided metals were made for the routes of n nets, it is required to allocate roughly $n/2$ tap-cells, which is an enormous overhead on chip PPA. (Note that the works in [5], [6] proposed a netlist planning that distributes nets across both sides to mitigate the occurrences of routing failure. On the other hand, the work in [7] utilized backside wires for ECO routing. However, all the works in [5]–[7] inevitably suffered from the tap-cell overhead.)

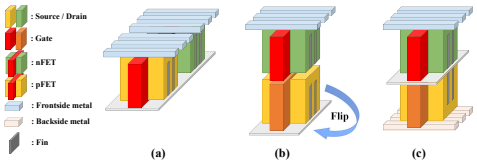


Fig. 1. Structures of FinFET, CFET, and FFET. (a) FinFET: routing tracks only at the frontside. (b) CFET: much smaller footprint than (a), but routing tracks only at the frontside. (c) FFET: a little smaller footprint than (b) and routing tracks at both the front and back sides.

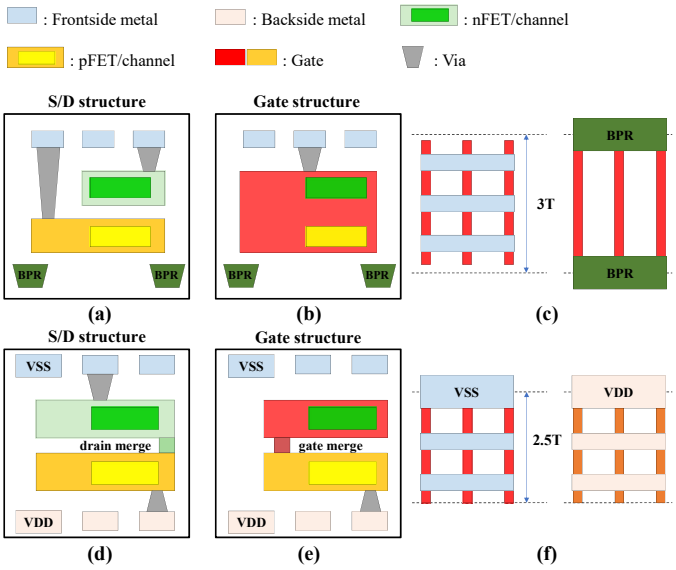


Fig. 2. Cross sectional view of 3T-height CFET ((a), (b), (c)) and 2.5T-height FFET ((d), (e), (f)).

Recently, a new 3D transistor technology called Flip-FET (FFET) [8] has been proposed, which *flips* the lower FET in CFET such that the locations of cell I/O pins can provide direct pin accessibility through either frontside or backside BEOL with no tap cells. Fig. 1(c) shows an illustrative FFET structure, in which two layers of FETs are formed and stacked

back-to-back, offering a manufacturing-compatible process flow with a lower aspect ratio in comparison to that of CFET structure in Fig. 1(b). Fig. 2 shows the structural differences between FFET and CFET with three horizontal metal tracks, in which each of the front and back sides in FFET runs a single power rail and 2 RTs (routing tracks), resulting in a total of 4 RTs and 2.5T cell height[1] while CFET runs 3 RTs only at the frontside, resulting in 3T cell height.

Note that regarding the automatic generation of standard cell layouts, many algorithms and frameworks have been proposed in the literature. Some of the representative works on FinFET based cell synthesis are found in [9]–[12], and the DTCO (design technology co-optimization) process using a cell generator as a core engine is well described in [13]. In addition, CFET based cell synthesis methods with SMT (satisfiability modulo theory) formulation and with an exploration of dynamic transistor foldings are proposed in [14], [15], and [16], respectively. However, none of the prior works have taken into account the exploitation of backside wires to improve pin accessibility, thus taking no consideration of tap-cell overhead at all.

In this work, we propose an FFET cell based DTCO methodology to fully make use of backside wires with minimal use of tap-cells. The contributions of our work can be summarized as:

1. We propose *an effective method of synthesizing FFET based standard cells*, the problem of which has never been addressed as yet, though a couple of automatic synthesis methods of CFET based standard cells have been proposed in the literature. Our cell generator exactly considers two differences in in-cell routing: (1) The flexible (front/back) side selection for individual I/O pin locations, and (2) The availability of additional metals i.e., the metals on BM1 (backside metal 1) layer.
2. We propose *a DTCO driven tap-cell avoiding cell replacement algorithm* that utilizes multiple cells with diverse styles of I/O pin distributions to eliminate the need for tap-cells during the placement optimization stage.
3. We propose *a net redistribution method*, which determines metal sides for all nets whose routes can be made by using either the frontside metals exclusively or the backside metals exclusively, thus unnecessary to allocate tap-cells so that the net connection on both sides should be mitigated in an unbiased manner.
4. Through experiments with benchmark circuits, it is shown that our proposed FFET cell based DTCO methodology is very promising, scaling up the chip size further over that of the CFET cell based design implementations while incurring much fewer routing failures.

II. Preliminary, Motivation, and Net Classification

CFET cell vs. FFET cell: We itemize the architectural differences between CFET [1] based and FFET [8] based cells in the following.

1. *Cell height*: For the same number of frontside metal tracks available for routing (e.g., 3 RTs (routing tracks) in Fig. 2), FFET allocates one track on each side for the power rails, allowing neighboring cells to share the power rails effectively. In contrast, CFET utilizes two backside tracks designated for BPR. Therefore, as illustrated in Fig. 2, CFET employs only three frontside RTs for BEOL processes, while FFET utilizes two RTs on each side (total of 4 RTs). As a result, FFET architecture has 0.5T smaller cell height while using the same amount of routing tracks in comparison with that of CFET architecture.
2. *Cell via*: CFET requires long (deep) vias to connect frontside metals to the FET in the lower layer, as illustrated in Fig. 2(a), introducing complex spacing rules. On the other hand, since FFET is formed by merely flipping the lower FET in CFET, the frontside FET is allowed to connect directly to the frontside metals and the backside FET to connect directly to backside metals.[2] This enables FFET to position I/O pins on either side. This flexible I/O pin positioning is particularly useful for making double-sided routing without tap-cells.
3. *Gate merge*: The implementation of split-gate structures allows for a reduction in the cell width, as shown in Figs. 3(b) and (c). FFET can make split-gate structures easily by *gate merge* with no area and wire overhead, as illustrated in Fig. 2(e). While split-gate CFET can also accommodate different signals on each gate, it requires inevitable area overhead. (Note that the split-gate of the lower FET in CFET is only accessible through frontside metals, resulting in additional spacing whereas that of FFET is accessible through backside metals.)

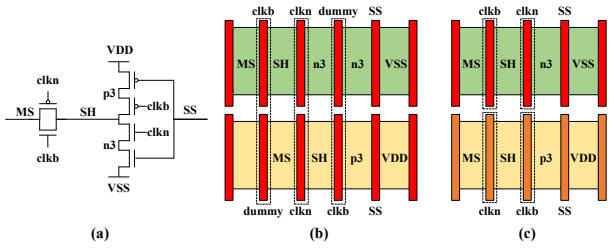


Fig. 3. Difference in gate split for CFET and FFET. (a) Netlist example: C^2MOS in DFFHQNx1. (b) Layout w/o split-gate on CFET: requiring dummy gate polys. (c) Layout with split-gate on FFET.

4. *Drain merge*: Both CFET and FFET can make *drain merge* with no overhead, as shown in Fig. 2. However, due to the fact that the output signal of the cell is always derived from a merged (common) drain, the signal can be delivered directly through metals on both sides for FFET, and it becomes possible to locate the output pin pattern on both sides to boost the pin accessibility as well as to minimize the number of tap-cell allocations.

[1]In FFET, one of the three horizontal metals is used as a power rail, which will be shared by neighboring cells.

[2]In this work, we assume the NFET on the frontside and the PFET on the backside, though they could be placed on either side.

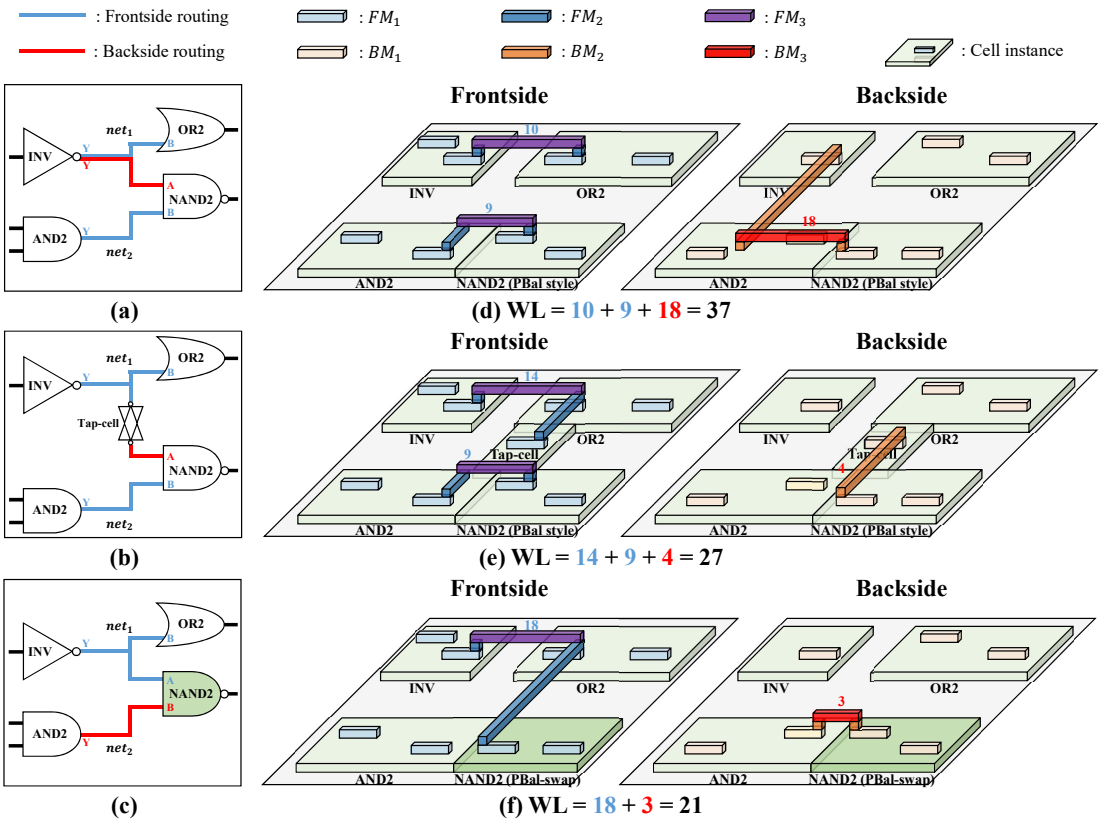


Fig. 4. Impact of BEOL side assignment for FFET cells' (input) pins on the number of tap-cells and WL (wirelength). Using NAND2 cell instances (a) with A on backside and B on frontside, (b) with A on backside and B on frontside connecting to a tap-cell (c) with A on frontside and B on backside. Feasible placement/routing results (d) WL = 37, #tap-cells = 0 for (a), (e) WL = 27, #tap-cells = 1 for (b), (f) WL = 21, #tap-cells = 0 for (c).

Motivation (tap-cells and WL): We describe, using an example, how the BEOL side (i.e., front/back sides) assignment of FFET cells' I/O pins affects the number of tap-cells and WL (wirelength) to be used. Figs. 4(a, d) and (b, e) show two different BEOL side assignments of input pins A and B on the NAND2 FFET cell in a part of the logic netlist where the OR2 cell commonly locates pin B on the frontside. Then, since output pins can be allocated to both sides, as observed in *item 4* of CFET vs. FFET in this section, the pin distribution of the NAND2 cell in Fig. 4(a, d) causes double-sided routing for net_1, which otherwise, requires one tap-cell as shown in Fig. 4(b, d). On the other hand, the pin distribution of the NAND2 cell in Fig. 4(c, f) makes only frontside routing for net_1 with no tap-cell. Thus, the clever selection (i.e., side assignment for pins) of the NAND2 cell instance can reduce the resulting WL (from 37 to 21 in Figs. 4(a, d) and (c, f) and the number of tap-cells.

Net classification: As observed in Fig. 4, all the nets to route can be classified into three types:

- *Frontside-only net (FS-net)*: A net whose all driven pins are accessed through the metals on the frontside (e.g., net_2 in Fig. 4(a) and net_1 Fig. 4(c)).
- *Backside-only nets (BS-net)*: A net whose all driven pins are accessed through the metals on the backside (e.g., net_2 in Fig. 4(c)).
- *Dual-side nets (DS-net)*: A net that belongs to neither *FS-net* nor *BS-net* (e.g., net_1 in Fig. 4(a)).

Note that unlike the double-sided routing for designs with non-FFET cells (e.g., [4]–[7]), which assumes low resistance on backside wires to account for routing timing-critical nets only with backside wires, but no consideration of tap-cell delay, we assume exactly identical metal configurations on both sides. This symmetry offers a significant technological advantage, as it enables the reuse of the same process for both N-FET and P-FET, as described in [8].

III. The Proposed DTCO Method

A. Overall DTCO Flow

Fig. 5 shows the overall flow of our DTCO framework, which is composed of three steps: (**Step 1**) we *build up an FFET based standard cell synthesizer*, which *systematically generates multiple FFET cells of diverse styles of pin distribution* for each primitive logic gate by considering the FFET's distinct feature that individual I/O pins on an FFET cell can be distributed on either side; (**Step 2**) at the placement optimization, we perform *a tap-cell avoiding cell replacement* by using the cells of diverse pin distributions prepared in Step 1 while minimizing metal resources and eliminating the need for allocating tap-cells simultaneously; (**Step 3**) following the cell replacement at the placement optimization, we perform *a congestion-aware net rebalancing*, so that the net congestion on both sides should be equally mitigated. After completing all steps, *FS-nets* as well as frontside segments of *DS-nets*

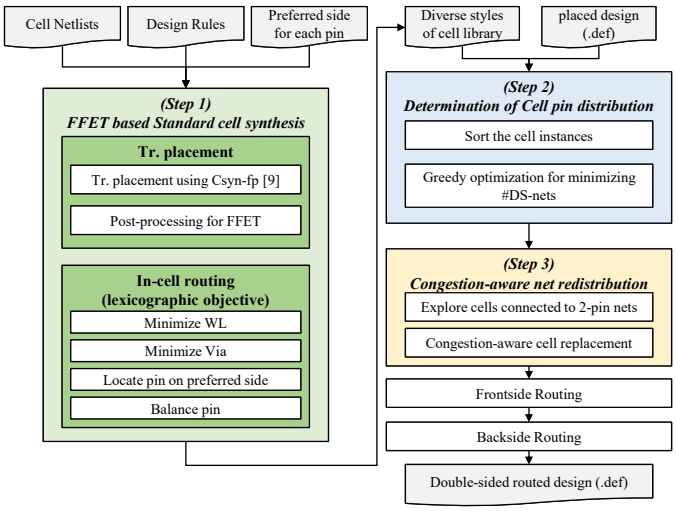


Fig. 5. Overall flow of our DTCO framework.

are routed at the frontside while *BS-nets* as well as backside segments of *DS-nets* are routed at the backside.

B. Synthesis of FFET based Standard Cells

1) ***Transistor placement****:* Given the cell's netlist information and design rules (e.g., OD jog rule) as inputs, as our initial transistor placements, we use the transistor placements produced by Csyn-fp [9] installed OpenROAD framework [17], which ensures cell area optimality. Then, we update the transistor placements by considering the following two factors: (1) locating cell output pins on both sides to support double-sided routing and (2) preferring *gate merge* to allow vertically merging dummy and non-dummy poly gates to increase the number of gate contact points.

2) ***In-cell routing****:* Our in-cell router generates a legal cell layout by incorporating design rules and the preferred BEOL side of individual input pins. As the conventional SMT formulation of in-cell routing, our SMT formulation includes, as constraints, representative design rules including minimum area rule (MAR), end-of-line (EOL) spacing, via enclosure (VE), and multiple patterning design rules such as parallel run length (PRL) and step height rule (SHR). In addition, our formulation uses the metals on *both backside and frontside metal layers*.

Our in-cell router is designed to optimize multi-objectives in a lexicographical order. The primary objective is to minimize metal length, followed by reducing the number of vias. The third objective prioritizes the input pin's preferred side, maximizing metal usage on that side. (If nothing is given as input, this objective is ignored.) The last objective is to balance the number of pins placed on both sides as double-sided routing could be the most effective when nets are distributed equally on both sides.

3) ***Generate multiple FFET cells of diverse styles****:* For each logic gate type, our cell synthesizer produces four styles of cell layout as follows.

- **Pin balanced (PBal)**: The cell layout with balanced pin distribution on both sides (e.g., Fig. 6(a)).
- **Pin balanced-swapped (PBal-swap)**: The cell layout with pin distribution sides on **PBal** swapped (e.g., Fig. 6(b)).
- **Pin-frontside (PFront)**: The cell layout with all pins preferred on frontside (e.g., Fig. 6(c)).
- **Pin-backside (PBack)**: The cell layout with all pins preferred on backside (e.g., Fig. 6(d)).

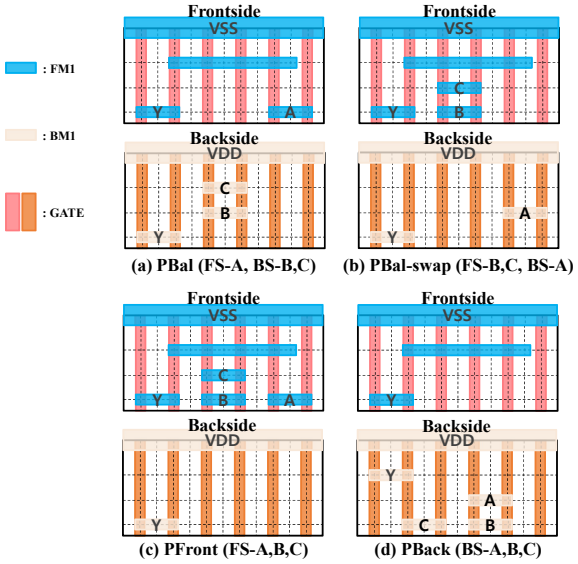


Fig. 6. Four styles of cell layout of OR3x1 generated by our cell synthesizer.

C. Determination of Cell Pin Distribution for WL Minimization

At the post-place, we determine, for each cell instance, the cell layout styles among **PBal**, **PBal-swap**, **PFront**, and **PBack** in a way to produce *DS-nets* (dual-side nets) as minimal as possible since routing *DS-nets* should use wires on *both* sides, incurring a longer total length of wires than that of *FS-nets* (frontside-only nets) and *BS-nets* (backside-only nets). To this end, we employ a greedy approach:

1. Sort the cell instances in placement in a non-increasing order according to the number of input pins. We break ties by the total amount of HPWL of the nets connecting the pins, placing priority on the cell instance with higher HPWL value.
2. For each cell instance c_i in the sorted list with input pins $p_1, \cdots$, we compute cost $\#DS(c_i, c_i')$ for each cell style $c_i' \in$ {**PBal**, **PBal-swap**, **PFront**, **PBack**}:

$$\#DS(c_i, c_i') = \sum_{j=1,\cdots} \rho(c_i, c_i', p_j), \tag{1}$$

where $\rho(c_i, c_i', p_j) = 1$ if replacing c_i with c_i' enables the net connecting p_j to become *DS-net*, and 0, otherwise. Then, we select the cell style which has the smallest $\#DS(c_i, c_i')$ value and perform cell replacement. For the same $\#DS(c_i, c_i')$ value, we choose the cell style with the smallest HPWL value.

TABLE I

COMPARISON OF CELL METRICS BETWEEN CFET BASED 4T STANDARD CELLS PRODUCED BY CFET CELL SYNTHESIZER [16] AND OUR FFET BASED 3.5T STANDARD CELLS. #CPP, WL, M1, M2, M3 INDICATE THE NUMBER OF CONTACTED POLY PITCHES (I.E., CELL WIDTH), TOTAL WIRELEGNTH, THE METAL USAGE ON M1, M2, M3, RESPECTIVELY.

Cell	#Cell	#track	CFET based 4T Standard Cells [16]							FFET based 3.5T Standard Cells (Ours)						
			Height	#CPP	Area	WL	M1	M2	M3	Height	#CPP	Area	WL	M1	M2	M3
AND	3	4	180	6.33	1140.00	71.00	16.00	11.00	0.00	141	6.33	893.00	58.67	24.00	0.00	0.00
AOI	2	4	180	10.00	1800.00	170.00	56.50	13.50	0.00	141	9.50	1339.50	126.00	50.00	0.00	0.00
BUF	4	4	180	7.50	1350.00	71.75	19.75	6.00	0.00	141	7.50	1057.50	70.50	26.50	0.00	0.00
DFFHQN	3	4	180	16.00	2880.00	230.00	84.67	13.00	7.00	141	16.67	2350.00	231.00	111.00	0.00	0.00
DHL	3	4	180	12.00	2160.00	143.67	57.33	9.00	0.00	141	13.00	1833.00	161.33	70.67	0.00	0.00
INV	4	4	180	5.75	1035.00	50.25	13.25	6.00	0.00	141	5.75	810.75	50.00	19.00	0.00	0.00
NAND	4	4	180	12.00	2160.00	173.25	63.75	10.50	0.00	141	12.00	1692.00	152.25	64.25	0.00	0.00
NOR	4	4	180	12.00	2160.00	186.00	64.00	13.75	1.25	141	11.75	1656.75	149.50	65.50	0.00	0.00
OAI	2	4	180	10.00	1800.00	166.00	54.50	13.50	0.00	141	10.00	1410.00	131.00	51.00	0.00	0.00
OR	3	4	180	6.33	1140.00	71.00	16.00	11.00	0.00	141	6.33	893.00	58.67	24.00	0.00	0.00
XNOR	1	4	180	11.00	1980.00	204.00	63.00	16.00	9.00	141	12.00	1692.00	158.00	74.00	0.00	0.00
XOR	1	4	180	11.00	1980.00	188.00	58.00	13.00	9.00	141	12.00	1692.00	162.00	78.00	0.00	0.00
Avg.		4	180	9.99	1798.75	143.74	47.23	11.35	2.19	141	10.24	1443.29	125.74	54.83	0.00	0.00
Norm.		4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.78**	1.02	**0.80**	**0.87**	1.16	**0.00**	**0.00**

WL: The total wirelength is a weighted sum of the wire length and the number of Vias used, following the same weighting factors in [15], [16].
M1: The M1 metal usage on FFET based cells includes the metal utilization on *both* of FM1 and BM1 layers.

D. Congestion-aware Net Redistribution

This step redistributes nets across the BEOL sides without increasing the number of *DS-nets* which have already been minimized in Step 2. We only consider cells with all input pins connected to 2-pin nets for redistribution since their impact on *DS-nets* is the least. Specifically, on a net-congested local region[3], we extract these cells and attempt to change their styles to **PFront** or **PBack** until both sides are equally congestion-balanced. For example, in a local region in Fig. 7(a) where high congestion occurs at the backside, replacing the **PBal** cell in the middle in Fig. 7(a) with **PFront** reduces BS-nets count, resulting in a reduction of congestion on the backside.

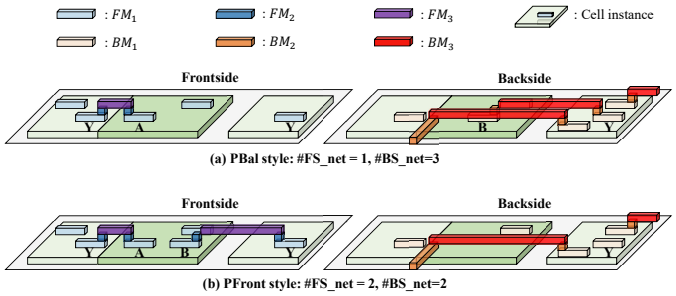


Fig. 7. Example of redistributing nets for mitigating net congestion on both sides.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

Our FFET based standard cells were implemented in C++ on a Linux system equipped with 128 GB of memory and an AMD EPYC 7452 CPU operating at 2.35 GHz. The input netlists and relevant design rules for synthesizing FFET cells were taken from ASAP 7nm process design kit (PDK) in [18], and our SMT formulation for in-cell routing was solved using Z3 solver package in [19]. We performed logic synthesis using Synopsys Design Compiler [20], and performed block-level placement and routing Cadence Innovus [21].

[3]We used *net density* as our pre-route congestion metric.

B. Cell Quality Comparison

Table I shows a comparison of the cell metrics between CFET based 4T (4 tracks for routing) standard cells produced by CFET-fp [16] which is available in [22] and follows the same design rules in ASAP 7nm PDK, and FFET based 3.5T cells produced by our proposed cell synthesizer.[4] In terms of cell area, our FFET cells are 20% smaller on average than that of CFET cells, which is mainly caused by the shorter height of FFET cells (i.e., 3.5T) than CFET cells (i.e., 4T). In addition, FFET cells use 13% less total wirelength than that of CFET cells. In particular, the comparison shows that the in-cell routing for FFET cells is fully completed with the use of FM1 and BM1 only while CFET cells use all metals in M1, M2, and M3. This clearly implies that FFET cells provide a considerable advantage of pin accessibility. (In fact, our cell synthesizer has successfully generated FFET layout cells for 174 netlists in ASAP 7nm PDK, using the metals on the FM1 and BM1 layers only.)

On the other hand, Table II shows a comparison of the cell metrics between CFET based 3T (3 tracks for routing) standard cells produced by CFET-fp [16] following the same design rules in ASAP 7nm PDK, and FFET based 2.5T cells produced by our proposed cell synthesizer.[5] The comparison shows that FFET cells have a 25% smaller area on average than that of CFET cells while using 31% less total wirelength.

C. Chip Quality Comparison

Table III shows a quality comparison between the chip implementations produced by (*Flow 1*) the conventional non-DTCO physical design flow using the CFET based cells in Table I, (*Flow 2*) the conventional non-DTCO flow using our FFET based cells in Table I, and (*Flow 3*) our proposed DTCO flow using our FFET based cells in Table I for the OpenCore benchmark designs in [23].

[4]For design rules, we set to MAR/EOL/VR/PRL/SHR/MPL = 1/2/1/1/2/1 on 4T CFET cells as well as 3.5T FFET cells.

[5]For design rules, we set MAR/EOL/VR/PRL/SHR/MPL = 1/1/0/1/0/1 on 3T CFET cells as well as 2.5T FFET cells.

TABLE II

COMPARISON OF CELL METRICS BETWEEN CFET BASED 3T STANDARD CELLS PRODUCED BY CFET CELL SYNTHESIZER [16] AND OUR FFET BASED 2.5T STANDARD CELLS. #CPP, WL, M1, M2, M3 INDICATE THE NUMBER OF CONTACTED POLY PITCHES (I.E., CELL WIDTH), TOTAL WIRELEGNTH, THE METAL USAGE ON M1, M2, M3, RESPECTIVELY.

Cell	#Cell	#track	CFET based 3T Standard Cells [16]							FFET based 2.5T Standard Cells (Ours)						
			Height	#CPP	Area	WL	M1	M2	M3	Height	#CPP	Area	WL	M1	M2	M3
AND	3	3	144	6.33	912.00	71.00	16.00	11.00	0.00	105	6.33	665.00	54.00	19.33	0.00	0.00
AOI	1	3	144	8.00	1152.00	188.00	35.00	20.00	21.00	105	9.00	945.00	104.00	40.00	0.00	0.00
NAND	4	3	144	11.75	1692.00	236.00	58.25	19.75	19.00	105	12.25	1286.25	146.00	59.00	0.00	0.00
NOR†	4	3	144	9.00	1296.00	145.00	37.33	12.67	7.00	105	9.00	945.00	102.00	39.33	0.00	0.00
OR	3	3	144	6.33	912.00	71.00	16.00	11.00	0.00	105	6.33	665.00	54.00	19.33	0.00	0.00
BUF	4	3	144	7.50	1080.00	71.75	19.75	6.00	0.00	105	7.50	787.50	69.25	25.25	0.00	0.00
INV	4	3	144	5.75	828.00	50.50	13.50	6.00	0.00	105	5.75	603.75	48.75	17.75	0.00	0.00
Avg.		3	144	7.81	1124.57	119.04	27.98	12.35	6.71	105	8.02	842.50	82.57	31.43	0.00	0.00
Norm.		3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.73**	1.03	**0.75**	**0.69**	1.12	**0.00**	**0.00**

† CFET-fp [16] could not produce CFET cell layout for gate NOR3x2.

TABLE III

CHIP QUALITY COMPARISON BETWEEN THE IMPLEMENTATIONS PRODUCED BY (*Flow 1*) NON-DTCO CONVENTIONAL FLOW USING CFET BASED CELLS, (*Flow 2*) NON-DTCO CONVENTIONAL FLOW USING OUR FFET CELLS, AND (*Flow 3*) OUR DTCO FLOW USING OUR FFET CELLS. *Util.*: CHIP UTILIZATION; *#DRV*: THE NUMBER OF DESIGN RULE VIOLATIONS; *WL*: TOTAL WIRELENGTH.

Design	Util.	#Cell	#Net	(1) Non-DTCO + CFET cells [16]			(2) Non-DTCO + FFET cells (Ours)			(3) DTCO (Ours) + FFET cells (Ours)		
				#DRV	Area	WL	#DRV	Area	WL	#DRV	Area	WL
AES_CIPHER	0.7	10021	10330	8546	10931.35	165475.73	0	8857.82	150115.73	0	8857.82	130193.74
USB_PHY	0.9	387	417	0	512.59	1810.68	0	402.14	2010.14	0	402.14	1914.94
DES3	0.75	51650	51985	1	65416.22	501568.14	3	51334.56	546286.69	4	51334.56	515168.51
FPU	0.9	33369	33666	385	36744.09	278735.02	22	29721.28	324966.30	11	29721.28	292088.43
Avg.				2233	28401.06	236897.39	6.25	22578.95	255844.71	3.75	22578.95	234841.41
Norm.				595.47	1.26	1.01	**1.67**	**1.00**	1.09	**1.00**	**1.00**	**1.00**

In summary, *Flow 2* is able to reduce the chip size by 26% over that of *Flow 1* while significantly reducing the DRV count (i.e., 356x). (Note that such a large number of DRVs in *Flow 1*, particularly for AES_CIPHER, is due to the fact that M2 and M3 metal resources have been used up for CFET in-cell routing, as statistically shown in Tables I and II, thereby become very limited M2 and M3 metals available for chip-level routing.

On the other hand, *Flow 3* outperforms *Flow 2* and *Flow 1*. *Flow 3* uses 9% less total wirelength and produces 67% fewer DRVs over that of *Flow 2*. This means that our DTCO flow is very effective in improving chip quality. To validate the effectiveness, we repeatedly applied *Flow 2* and *Flow 3* to design FPU by varying the value of the chip utilization parameter, and compared the implementation results, as shown in Fig. 8. The comparison shows that *Flow 3* consistently reduces the number of DS-nets (dual-sided nets) (by 63% on average), the DRV count (by 51% on average), and total wirelength (by 10% on average) over that of *Flow 2*. Note that our DTCO in *Flow 3* never increases the number of tap-cells over that of *Flow 2* during the (wire-hungry) DS-net minimization in Steps 2 and 3 while *Flow 2* allocates 2, 12, 113, and 0 tap-cells for the implementations of AES_CIPHER, USB_PHY, DES3, and FPU, respectively.

V. CONCLUSION

This work addressed two tasks: (1) FFET (Flip-FET) based standard cell synthesis and (2) DTCO methodology utilizing FFET based cells. A strong and distinct advantage of FFET cells, as opposed to CFET cells, was the convenience of using backside metals as in-cell routes (i.e., no tap-cell overhead) as well as using them as chip routes (i.e., well controllable for tap-cells), the objective of this work was to develop

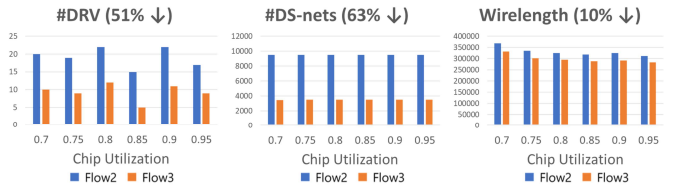


Fig. 8. Design quality comparison of the implementations produced by *Flow 2* (non-DTCO + FFET cells) and *Flow 3* (DTCO + FFET cells) for design FPU by varying the chip utilization value.

an effective solution to the problem of how the backside metals could be fully utilized as interconnects with minimal tap-cell overhead. To this end, we proposed a three-step solution: (1) synthesizing multiple FFET based cells with diverse styles of pin distribution on front/backside BEOLs for each primitive logic gate; (2) performing a tap-cell avoidance cell replacement during placement optimization by exploiting the cells obtained in Step 1; (3) rebalancing the wire usage between the frontside and backside to mitigate net congestion. Experimental results with benchmark circuits showed that our FFET cell-based DTCO flow was able to significantly improve chip quality in terms of chip area, wirelength, and routing failures over that of the CFET/FFET cell-based conventional non-DTCO flows.

ACKNOWLEDGEMENT

This work was supported by National Research Foundation by Korea Government (2021-R1A2C2008864), IITP-2023-RS-2023-00256081, Samsung Electronics Co., Ltd (IO201216-08205-01, LSI250306-0010, IO241217-11507-01), SAIT (IO250331-12476-01), SK hynix Inc. (MDUS010D), and ISRC (Inter-University Semiconductor Research Center) in SNU. EDA tool was supported by IDEC.

REFERENCES

[1] J. Ryckaert, P. Schuddinck, P. Weckx, G. Bouche, B. Vincent, J. Smith, Y. Sherazi, A. Mallik, H. Mertens, S. Demuynck *et al.*, "The complementary fet (cfet) for cmos scaling beyond n3," in *2018 IEEE Symposium on Vlsi Technology*. IEEE, 2018, pp. 141–142.

[2] D. Prasad, S. T. Nibhanupudi, S. Das, O. Zografos, B. Chehab, S. Sarkar, R. Baert, A. Robinson, A. Gupta, A. Spessot *et al.*, "Buried power rails and back-side power grids: Arm® cpu power delivery network design beyond 5nm," in *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019, pp. 19–1.

[3] J. Ryckaert, A. Gupta, A. Jourdain, B. Chava, G. Van der Plas, D. Verkest, and E. Beyne, "Extending the roadmap beyond 3nm through system scaling boosters: A case study on buried power rail and backside power delivery," in *2019 Electron Devices Technology and Manufacturing Conference (EDTM)*. IEEE, 2019, pp. 50–52.

[4] S.-H. Chen, J. C. J. Kao, K. Yang, and J. Liu, "System and method for back side signal routing," U.S. Patent 11,423,204, 2022, filed on April 14, 2021.

[5] T.-C. Lin, F.-Y. Hsu, W.-K. Mak, and T.-C. Wang, "An effective netlist planning approach for double-sided signal routing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 288–293.

[6] F.-Y. Hsu, T.-C. Lin, W.-K. Mak, and T.-C. Wang, "A bounding box-based net partitioning method for double-sided routing," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 397–402.

[7] C.-P. Tsai, F.-Y. Hsu, W.-k. Mak, and T.-C. Wang, "An effective eco methodology for reducing back-side design rule violations in double-sided signal routing," in *2024 43rd International Conference On Computer Aided Design (ICCAD)*. IEEE/ACM, 2024.

[8] H. Lu, Y. Ge, X. Jiang, J. Sun, W. Peng, R. Guo, M. Li, Y. Lin, R. Wang, H. Wu *et al.*, "First experimental demonstration of self-aligned flip fet (ffet): A breakthrough stacked transistor technology with 2.5 t design, dual-side active and interconnects," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[9] K. Baek and T. Kim, "Simultaneous transistor folding and placement in standard cell layout synthesis," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[10] Y.-L. Li, S.-T. Lin, S. Nishizawa, H.-Y. Su, M.-J. Fong, O. Chen, and H. Onodera, "Nctucell: A dda-aware cell library generator for finfet structure with implicitly adjustable grid map," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[11] D. Lee, D. Park, C.-T. Ho, I. Kang, H. Kim, S. Gao, B. Lin, and C.-K. Cheng, "Sp&r: Smt-based simultaneous place-and-route for standard cell synthesis of advanced nodes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 10, pp. 2142–2155, 2020.

[12] H. Cho, H. Seo, S. Chung, K.-M. Choi, and T. Kim, "Standard cell layout generator amenable to design technology co-optimization in advanced process nodes," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[13] K. Jo, S. Ahn, J. Do, T. Song, T. Kim, and K. Choi, "Design rule evaluation framework using automatic cell layout generator for design technology co-optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1933–1946, 2019.

[14] C.-K. Cheng, C.-T. Ho, D. Lee, B. Lin, and D. Park, "Complementary-fet (cfet) standard cell synthesis framework for design and system technology co-optimization using smt," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 6, pp. 1178–1191, 2021.

[15] C.-K. Cheng, C.-T. Ho, D. Lee, and B. Lin, "Multirow complementary-fet (cfet) standard cell synthesis framework using satisfiability modulo theories (smts)," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 7, no. 1, pp. 43–51, 2021.

[16] S. Kim and T. Kim, "Optimal transistor folding and placement for synthesizing standard cells of complementary fet technology," in *2024 Design Automation Conference (DAC)*. IEEE/ACM, 2024, pp. 1–6.

[17] T. Ajayi and D. e. a. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[18] L. T. Clark *et al.*, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[19] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*. Springer, 2008, pp. 337–340.

[20] *Synopsys Design Compiler and PrimeLib User Guide*, https://www.synopsys.com.

[21] *Cadence Innovus User Guide*, https://www.cadence.com.

[22] S. Kim, "Cfet_fp," https://github.com/SNU-suwan/CFET_fp, 2024.

[23] "Opencores: Open source ip-cores," https://www.opencores.org.

DataMaestro: A Versatile and Efficient Data Streaming Engine Bringing Decoupled Memory Access To Dataflow Accelerators

Xiaoling Yi[1*], Yunhao Deng[1*], Ryan Antonio[1], Fanchen Kong[1], Guilherme Paim[2], Marian Verhelst[1]
[1]MICAS-ESAT, KU Leuven, Leuven, Belgium
[2]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

***Abstract*—Deep Neural Networks (DNNs) have achieved remarkable success across various intelligent tasks but encounter performance and energy challenges in inference execution due to data movement bottlenecks. We introduce *DataMaestro*, a versatile and efficient data streaming unit that brings the decoupled access/execute architecture to DNN dataflow accelerators to address this issue. *DataMaestro* supports flexible and programmable access patterns to accommodate diverse workload types and dataflows, incorporates fine-grained prefetch and addressing mode switching to mitigate bank conflicts, and enables customizable on-the-fly data manipulation to reduce memory footprints and access counts. We integrate five *DataMaestro*s with a Tensor Core-like GeMM accelerator and a Quantization accelerator into a RISC-V host system for evaluation. The FPGA prototype and VLSI synthesis results demonstrate that *DataMaestro* helps the GeMM core achieve nearly 100% utilization, which is 1.05-21.39× better than state-of-the-art solutions, while minimizing area and energy consumption to merely 6.43% and 15.06% of the total system.**

I. Introduction

Deep Neural Networks (DNNs) have significantly gained popularity for tackling complex intelligence tasks and potentially advancing humanity toward the Artificial General Intelligence (AGI) era. To meet the stringent performance and energy consumption requirements of DNN deployment, numerous contemporary domain-specific dataflow accelerators have been developed [2], [7]–[11], incorporating extensive optimizations in the processing element (PE) array design and dataflow strategies. However, as DNN workloads and PE arrays continue to scale, data communication between the accelerator datapath and the memory hierarchy emerges as a critical bottleneck in DNN dataflow accelerators [4], [12]–[16].

Despite various solutions having been proposed to address this bottleneck, three shortcomings hinder their widespread deployment toward efficient DNN dataflow accelerators: **1). Non-Reusable Design.** Data movement units are typically implemented as dedicated hardware blocks, tightly coupled to a specific accelerator, featuring fixed access patterns and bandwidths tuned to specific workloads [2]–[4], [6], [7]. Given the diverse sizes, topologies, and rapid evolution of DNN workloads, these dedicated data movement units lack the versatility to adapt to different dataflows' requirements, leading to suboptimal overall system performance and energy efficiency. For instance, BitWave [2] achieves very high PE array utilization for convolution operations in convolutional neural networks (CNNs) by leveraging specialized optimizations but falls short in general matrix-matrix multiplication (GeMM), which is pervasive in Transformer models. Moreover, the inability to reuse these dedicated data movement blocks across different accelerators results in redundant engineering efforts and hinders competitive time-to-market, especially considering the rapid evolution of DNNs and next-generation computing platforms. **2). Performance Decrease Caused by Bank Conflicts.** The on-chip memory bank conflict challenge is a primary obstacle to achieving peak system performance. For instance, Gemmini [1] exhibits a PE array utilization as low as 10% due to the lack of bank conflict management in the access of the different operands. To address this problem, many accelerators [2], [8], [17], [18] leverage private on-chip memory for each operand, such as dedicated input, weight, and output buffers. However, bank conflicts still arise when data needed in a single cycle is stored in different wordlines within the same bank, a scenario commonly encountered in strided convolution operations. Moreover, it imposes additional limitations on the tiling of the workload to meet the requirement of the smallest memory and introduces extra complexity to manage multiple address spaces [15]. **3). Expensive Data Manipulation.** Many accelerators require preprocessing of operand before computation, including permutation, data alignment, and *im2col*. A common approach is to design standalone data manipulation units to perform these data rearrangement operations [1], [9], [11]. These units require additional memory accesses and intermediate data storage, which again increases the likelihood of memory conflicts and elevates memory energy consumption. In conclusion, there is an absence of a versatile and efficient data movement solution to address the diverse dataflow and data manipulation requirements of different accelerators while also ensuring high system performance.

The decoupled access/execute (DAE) architecture [19] separates data access and computation processes into two independent streams to prevent mutual stalls. This approach has been applied in numerous works [5], [20], [21], demonstrating performance speed-ups of 2-5× for linear algebra workloads [5], [20]. The DAE architecture is well-suited for DNN workloads, as its regular multidimensional access patterns [22] enable data to be fetched ahead of computation. This facilitates continuous data stream consumption and production from the perspective of PE arrays, helping the accelerator system reach its theoretical peak performance. Previous works [6], [7] have successfully applied the DAE architecture in accelerator designs. However, they fail to meet the diverse dataflow, data access patterns, and high bandwidth requirements of DNN workloads and the aforementioned challenges, rendering them suboptimal for DNN dataflow accelerators.

In this paper, we present a versatile and efficient data streaming engine, called *DataMaestro*[1], which brings decoupled memory access to DNN dataflow accelerators. *DataMaestro* is highly optimized and can be reused across different dataflow accelerators with flexible and programmable access patterns while achieving nearly 100% PE array utilization. A comprehensive comparison between *DataMaestro* and state-of-the-art (SotA) solutions is listed in Table I. In summary, our main contributions lie in:

- We present *DataMaestro*, a generalized, versatile, and efficient data streaming engine for diverse DNN dataflow accelerators, with design-time and runtime configurability, enabling decoupled memory access for arbitrary dataflows (§ III-A and § III-B).
- We introduce fine-grained prefetch and runtime-configurable addressing mode switching features inside *DataMaestro*, which minimizes bank conflicts and fully exploits the available on-chip memory bandwidth (§ III-C and § III-D).
- We provide on-the-fly data manipulation capability inside the *DataMaestro* by customizable datapath extensions to effectively reduce memory footprints and access counts (§ III-E).

*Equal contribution.

[1]*DataMaestro* and the *DataMaestro*-boosted accelerator system are open-sourced at: https://github.com/KULeuven-MICAS/snax_cluster.

TABLE I: Comparison of the SotA data movement solutions with *DataMaestro*.

	Gemmini [1]	BitWave [2]	[3]	FEATHER [4]	SSR [5]	Buffet [6]	Softbrain [7]	*DataMaestro*
Open source	✓	✗	✗	✓	✓	✓	✗	✓
Reusable design	✗	✗	✗	✗	✗	✓	✗	✓
Decoupled access/execute	✗	✗	✗	✗	✓	✓	✓	✓
Programmable affine access	✓(2-*D*)	✗	✓(2-*D*)	✗	✓(4-*D*)	✓(2-*D*)	✓(2-*D*)	✓(*N*-*D*)
Fine-grained prefetch	✗	✗	✗	✗	✗	✗	✗	✓
Runtime addressing mode switching	✗	✗	✗	✗	✗	✗	✗	✓
On-the-fly data manipulation	✗	✗	✗	✓	✗	✗	✗	✓

- We demonstrate the versatility and efficiency of *DataMaestro* by integrating it with a Tensor core [23]-like GeMM accelerator and a Quantization accelerator into a RISC-V host system. The FPGA prototype and VLSI synthesis results show that *DataMaestro* enhances the GeMM PE array utilization to 95.45%-99.98% under real-world DNN workloads, boosting normalized throughput to 1.05-21.39× over SotA, while only consuming 6.43% and 15.06% of system area and energy (§ IV).

II. Background

A. Dataflow and Data Layout

Typical DNN kernels, such as GeMM, multi-head attention, convolution, and pooling, can be represented as nested loops that iterate over the input, weight, and output tensor dimensions. Dataflow optimizations, like splitting and reordering these loops, open up a vast mapping space for efficient hardware processing [14], [15]. Specifically, these optimizations include spatial unrolling (SU) for parallel execution and temporal unrolling (TU) to control data tiling and the order of loop iterations.

The data access pattern, which specifies the data required by the accelerator in each clock cycle, is inherently determined by the dataflow. However, the data layout, referring to the organization of tensor data in memory, also influences data access patterns. A proper data layout is crucial for optimizing DNN performance, as it directly impacts memory access efficiency and can lead to PE array underutilization if mismatched with the dataflow [4], [24].

B. Decoupled Access/Execute Architecture for Dataflow Accelerators

A decoupled access/execute (DAE) architecture [6], [7], [19] explicitly decouples memory access and computation by introducing data streaming engines between them, as illustrated in Figure 1. This architecture integrates three key components — memory subsystem, data streaming engine, and accelerator datapath—which collaborate coherently to produce/consume three types of streams: access streams, data streams, and execute streams. The memory subsystem serves as a data reservoir, responding to the memory requests via the access stream. The accelerator's datapath applies computation to the data streams and produces results via the execute stream. The data streaming engine bridges the memory subsystem with the accelerator datapath, orchestrating data flow between them. It transforms scattered data in memory which is arranged in a specific format (data layout), into a continuous data stream, which is organized in the format required by the accelerator (data flow) and vice versa, by interacting with the memory through the access stream.

Typically, the streaming engine consists of three main components: an Address Generation Unit (AGU), a memory interface controller, and a data FIFO. The AGU generates address sequences, which are then consumed by the memory interface controller to perform memory operations. The data FIFO temporarily stores data streams, decoupling memory access from computation. This decoupling enables data prefetch operations and thus minimizes computation interruptions caused by memory stalls. Such an approach is particularly beneficial for DNN workloads due to their regular and deterministic data access patterns. By incorporating this architecture, a harmonious access-data-execute stream orchestration is established within the accelerator system. In the next section, we present *DataMaestro*, a highly optimized data streaming engine designed for DNN accelerators, which efficiently streamlines the data streams to enhance the performance of dataflow accelerators.

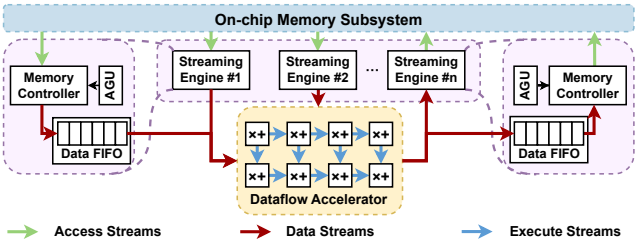


Fig. 1: Overview of decoupled access/execute architecture.

III. *DataMaestro* Architecture

In this section, we will first provide an overview of the *DataMaestro* architecture in § III-A, along with all of its configurable parameters, which are listed in Table II. Next, we introduce the three novel architectural features that maximize *DataMaestro*'s efficiency and minimize memory bank conflicts across a wide range of workloads: 1) highly configurable and area-efficient N-Dimensional (*N*-*D*) address generation unit (§ III-B); 2) fine-grained prefetch (§ III-C); and 3) runtime-configurable addressing mode switching (§ III-D). Finally, we discuss the datapath extension interface of *DataMaestro*, which enables customized on-the-fly data manipulation (§ III-E). These unique features of *DataMaestro* are key to its high efficiency and enhancement of the performance of DNN dataflow accelerators.

TABLE II: Design-time parameters and runtime configurations in *DataMaestro*.

Scope	Parameter	Description	Type †
Design Time	N_R	Num. of read *DataMaestro*	Int
	N_W	Num. of write *DataMaestro*	Int
	$Mode_{R/W}$	Read or Write mode	Bool.
	B_s	Spatial Bounds	[Int]
	D_s	Num. of Spatial Dimensions	Int
	D_t	Num. of Temporal Dimensions	Int
	N_C	Num. of Channels	Int
	D_{ABf}	Address Buffer Depth	Int
	D_{DBf}	Data Buffer Depth	Int
	DP_{ext}	Datapath Extensions	[Ext.]
	W_B	Bank Width	Int
	N_{BF}	Num. of Bank in Full Memory	Int
	N_{BG}	Num. of Bank in One Group	[Int]
Run Time	$Addr_B$	Base Address	UInt
	S_s	Spatial Strides	[UInt]
	B_t	Temporal Bounds	[UInt]
	S_t	Temporal Strides	[UInt]
	R_S	Addressing Mode Selection	UInt

† Type enclosed in square brackets [ ] means List.

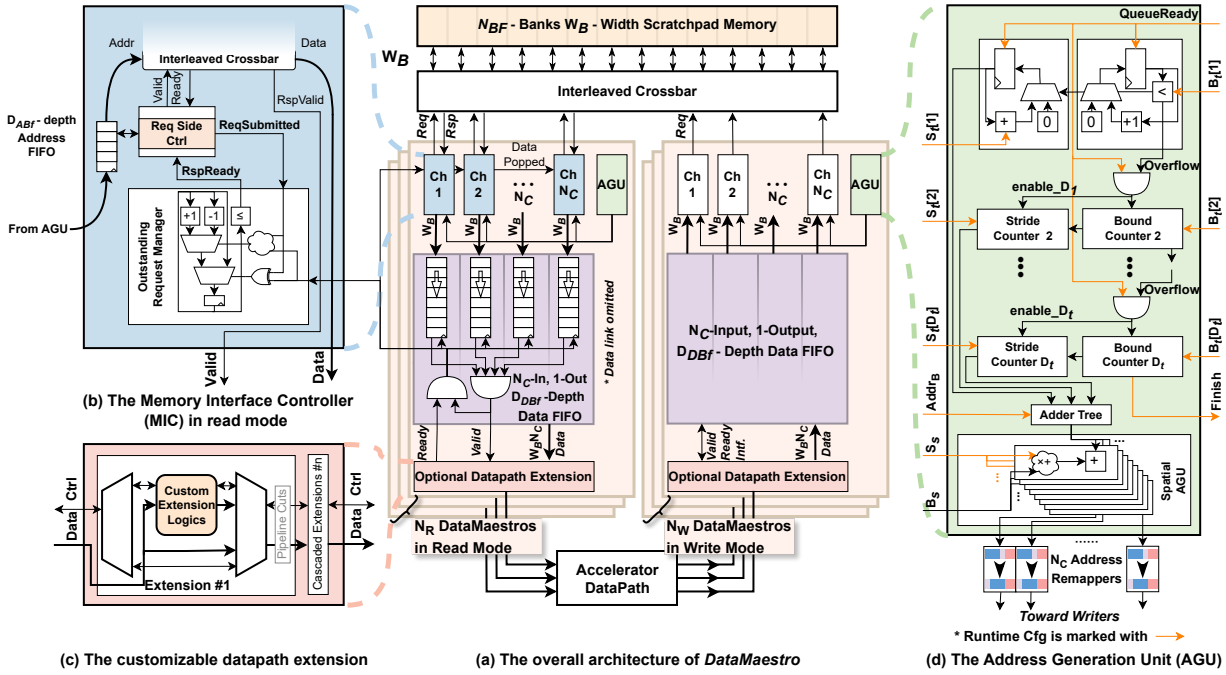


Fig. 2: *DataMaestro* architecture.

A. Architecture Overview of DataMaestro

Figure 2 (a) illustrates a scenario where N_R+N_W *DataMaestros* are connected to a multi-banked scratchpad memory subsystem (top) and an accelerator (bottom), effectively managing data streams between them to create a complete accelerator system. One read *DataMaestro* (middle, left) and one write *DataMaestro* (middle, right) are depicted in detail. The memory subsystem consists of an N_B-banked scratchpad memory that provides high memory bandwidth, along with an interleaved crossbar that ensures full accessibility from any request port. To meet the specific requirements of different accelerator ports, N_R read and N_W write *DataMaestros* are deployed, each of which can be configured independently at both design time and runtime, by providing the parameters listed in Table II.

A read or write *DataMaestro* primarily includes N_C independent memory interaction channels, breaking one wide memory request into multiple narrower operations for asynchronous, fine-grained memory access. Each channel is equipped with a dedicated Memory Interface Controller (MIC) (Figure 2 (b)) and a data FIFO. The MIC consumes addresses from the address generation unit (AGU) (Figure 2 (d)) to issue memory requests, and the data FIFO acts as a buffer between the *DataMaestro* and the accelerator. This facilitates a decoupled access/execute architecture for dataflow accelerators to effectively hide memory latency. At the output ports of the FIFOs, data fetched by multiple channels are gathered into a wide data stream and sent as a single wide data word to the accelerator datapath, enabling highly paralleled execution. Furthermore, at the interface between the *DataMaestro* and the accelerator (Figure 2 (c)), multiple user-customizable datapath extensions can optionally be inserted to perform on-the-fly data processing in a cascaded manner without requiring intermediate memory storage. In the following subsections, we will delve into each part of *DataMaestro* in more detail.

B. Programmable Affine Access Pattern by Address Generation Unit

The data access pattern for DNN workloads, determined by the dataflow and data layout, can typically be represented as an affine pattern [22]. Furthermore, DNN workloads encompass a

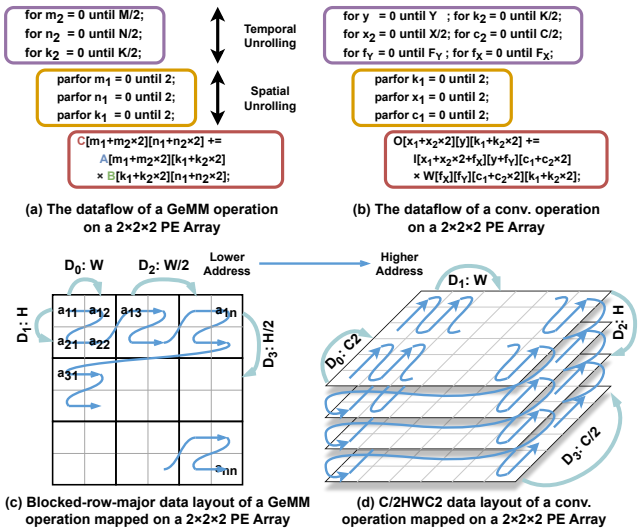


Fig. 3: Dataflow and data layout of GeMM and convolution operations mapped on $2 \times 2 \times 2$ PE array.

wide range of operation types (e.g., GeMM and various types of convolutional operations) and tensor shapes (e.g., height, width, and channel dimensions) [25], necessitating data access patterns with high flexibility. For example, as illustrated in Figure 3, the dataflow and data layout of GeMM and convolution workloads mapped on a simple $2 \times 2 \times 2$ PE array already differ significantly. For GeMM workload, the matrix A is stored using a 4-D block-row-major data layout [26], [27] (Figure 3 (c)), and exhibits a 3-D access pattern across different cycles (Figure 3 (a)). In contrast, for convolution, the input tensor is stored using a 4-D blocked data layout (Figure 3 (d)), i.e., $C/2HWC2$, and features a 6-D data access pattern (Figure 3 (b)). The diverse access pattern (while remaining affine) in DNN workloads motivates the design of a flexible and programmable AGU within *DataMaestro*.

The AGU in *DataMaestro* enables N-Dimensional (N-D) pro-

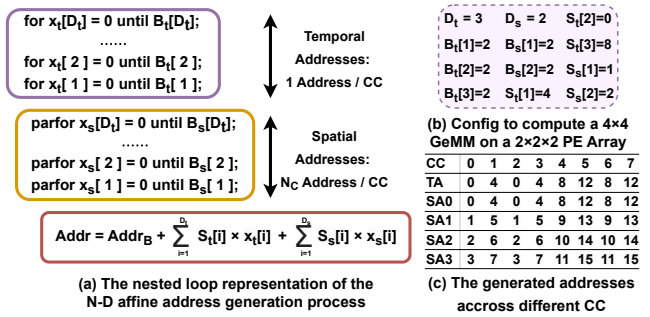


Fig. 4: The nested loop representation of the N-D affine address generation in *DataMaestro*'s AGU and a simple address generation example for a M=N=K=4 GeMM workload mapped on a $2 \times 2 \times 2$ PE array.

grammable affine address generation as shown in Figure 4 (a), using symbols from Table II. This affine function effectively describes the mapping of the N-D data access space to the 1-D address space. Our address generation process involves sequentially generating temporal addresses (TA) to represent temporal data access patterns and simultaneously generating multiple spatial addresses (SA) based on every temporal address for parallel data access, satisfying the requirements of temporal and spatial unrolling for a specific dataflow. A detailed example of the AGU configuration for an M=N=K=4 GeMM workload mapped on a $2 \times 2 \times 2$ PE array is presented in Figure 4 (b), with the corresponding address generation process depicted in Figure 4 (c).

Directly using a single counter to track the address generation progress necessitates modulo units and dividers to compute loop indices, followed by multipliers and adders to calculate temporal addresses. However, this leads to long combinatorial paths and significant hardware overheads that linearly scale with the number of dimensions. In *DataMaestro*, we incorporate microarchitectural optimizations to enable high-dimensional address generation with low overhead and shorter combinatorial paths. As shown at the top of Figure 2 (d), each dimension of the temporal AGU is implemented in a dual-counter structure, including a simple counter (Bound Counter) to record the loop index and a step-programmable counter (Stride Counter) to generate the temporal address offset. Then, address offsets of all dimensions are summed with the base address ($Addr_B$) and consumed by a multi-channel spatial AGU (the bottom of Figure 2 (d)), which generates distinct addresses for each channel. Based on this optimized hardware architecture, *DataMaestro* supports specifying arbitrary temporal address loop dimension, spatial address loop dimension, and spatial address loop bounds at design time, while the strides for both temporal and spatial addresses can be programmed at runtime, providing full flexibility to accommodate diverse data access patterns in DNN dataflow accelerators.

C. Fine-grained Prefetch by Memory Interface Controller

DataMaestro conducts fine-grained prefetch by breaking one wide memory request into multiple narrower channels and issuing them independently. During this process, Memory Interface Controllers (MICs) in read mode play a key role in sending fine-grained requests, gathering response data, and storing it in the data FIFO. The MIC in read mode includes an Outstanding Request Manager (ORM) and a Request Side Controller (RSC), as shown in Figure 2(b). The ORM tracks the utilization of the data FIFO and reserves the FIFO slots for in-flight memory requests, throttling the RSC if no available slots in the FIFO can be reserved. With permission from the ORM and valid

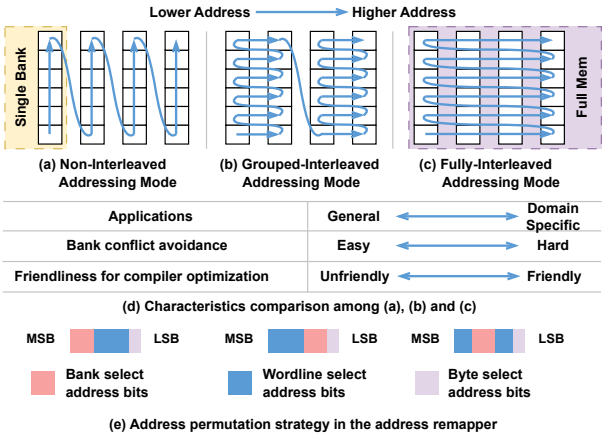


Fig. 5: Addressing modes supported by *DataMaestro*.

addresses from the AGU, the RSC immediately issues new memory requests. This aggressive and fine-grained data prefetch mechanism maximizes memory bandwidth utilization.

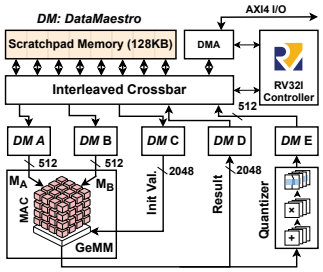
D. Low Overhead Addressing Mode Switching by Address Remapper

Two addressing modes are commonly used in computer systems for accessing multi-banked memory: Fully-Interleaved Memory Addressing (FIMA), where addresses are interleaved across all banks (Figure 5(a)), and Non-Interleaved Memory Addressing (NIMA), where contiguous addresses are allocated to a single bank (Figure 5(c)). Between these two lies an intermediate mode, termed Grouped-Interleaved Memory Addressing (GIMA) by us, where some banks are grouped, and the address is interleaved intra-group and remains contiguous inter-group (Figure 5(b)). The characteristics of these three addressing modes are summarized in 5 (d). While FIMA is widely used in general-purpose computing systems [28], it is more prone to bank conflicts in domain-specific accelerator (DSA) systems. As a result, contemporary dataflow accelerators often favor the NIMA mode [1]–[3]. However, the compiler needs to carefully allocate data for maximal performance and it constrains the tilings of the workload to meet the smallest memory requirement. Positioned between these two, GIMA strikes a trade-off between them. To accommodate diverse DNN workloads and dataflow accelerators, *DataMaestro* is designed to support all three addressing modes, with the capability to switch between them at runtime, offering the designer an extra degree of freedom to optimize the data allocation.

The main challenge of addressing mode switching is the overhead introduced in address decoding. However, addressing mode switching can be represented as a simple bit permutation of the address when the bank number in one group is a power of two. Based on this insight, *DataMaestro* leverages a memory address remapper that facilitates simple address bit permutation for addressing mode switching, as illustrated in Figure 5 (e). *DataMaestro* instantiates the bit permutations in hardware based on the number of banks in total (N_{BF}) and the number of banks in one group (N_{BG}) at design time. Then, the permuted addresses are connected to a multiplexer, allowing the user to select the addressing mode by setting R_S at the run time.

E. On-the-fly Data Manipulation by Datapath Extension

DataMaestro provides a mechanism to insert customizable extensions, such as data quantization and permutation, between *DataMaestro* and accelerator datapath in a plug-and-play manner to conduct on-the-fly data manipulation, as shown in Figure 2 (c). Datapath extensions are instantiated based on the design-time configuration DP_{ext}, with the output of one extension serving as the



Parameter	A	B	C	D	E
D_{spat}	1	1	2	2	1
B_{spat}	[8]	[8]	[8,4]	[8,4]	[8]
D_{temp}	6	3	3	3	3
W_B	64	64	64	64	64
N_C	8	8	32	32	8
D_{DBf}	8	8	1	1	1
DP_{ext}	[Trans.]		[Broad.]	[ ]	[ ]
N_{BF}	2048				
N_{BG}	[2048, 512]				

Fig. 6: The *DataMaestro* evaluation system (left) and *DataMaestro*s design-time parameters used in it (right).

input for the next. Furthermore, the logic to bypass any extension is automatically inserted so that users can disable any extension at runtime.

IV. Evaluation

A. Setup and Methodology

To assess the versatility and efficiency of *DataMaestro*, we integrate it with a Tensor Core [23]-like GeMM accelerator and a Quantization accelerator into a RISC-V host system [29], as depicted in Figure 6 (left). The GeMM accelerator features a 3-*D* 8×8×8 PE array, capable of executing workloads expressed as $D_{32} = A_8 \otimes B_8 + C_{32}$, where the subscripts denote operand data precision and $\otimes$ signifies either GeMM or convolution operations, depending on the runtime configuration of *DataMaestro*s. The output from the GeMM accelerator can be directed either to the Quantization accelerator for post-processing, represented as $E_8 = Rescale(D_{32})$, or sent back to the memory. The GeMM and Quantization accelerators feature five data stream ports in total, each with different data access patterns and bandwidth requirements, served by five *DataMaestro*, as shown in Figure 6 (right). The 6-*D* temporal AGU within *DataMaestro A* facilitates implicit *im2col* transformation [30] for convolution operations. Two datapath extensions are implemented: 1). Transposer, which performs on-the-fly matrix tile transposition, and 2). Broadcaster, which duplicates data across channels, particularly useful for DNN workloads with per-channel quantization. Additionally, a customized compiler is developed to generate runtime configurations for these *DataMaestro*s, considering workload specifications and tensor data layouts.

We implement *DataMaestro*, GeMM, and Quantization accelerators at the register-transfer level (RTL) using Chisel [31]. An ablation study is conducted using Verilator for cycle-accurate RTL simulation to evaluate utilization gains and memory access count reductions provided by each feature of *DataMaestro* (§ IV-B). Furthermore, an FPGA prototype is deployed on the AMD Versal™ VPK180 FPGA to benchmark model-wise performance on real-world DNN workloads (§ IV-C). The evaluation system is also synthesized using the Synopsys Design Compiler® with GlobalFoundries 22FDX® technology, operating at 1GHz and 0.8V with strict timing closure. Post-synthesis simulation is performed for power consumption analysis using Siemens QuestaSim™ and Synopsys PrimeTime® (§ IV-D).

B. Ablation Study for DataMaestro Evaluation

1) Workload and Evaluation Architecture Setting: We conduct an ablation study to evaluate the effectiveness of each innovation in *DataMaestro* using 260 different synthetic DNN workloads, categorized into three groups: 1) GeMM, 2) transposed GeMM, and 3) convolution. The synthetic benchmark set features various matrix sizes for GeMM and transposed GeMM, along with diverse feature map sizes, channels, kernel sizes, and strides for convolution, effectively representing typical Transformer and CNN layers. Initially, we disable

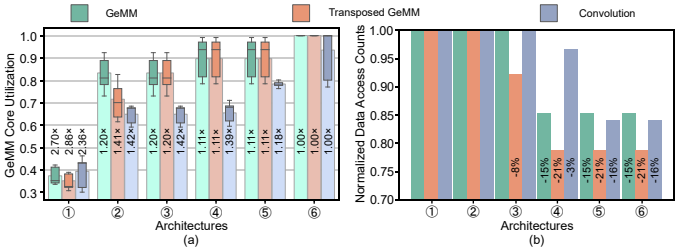


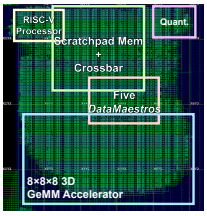
Fig. 7: Ablation study results of GeMM core's utilization distribution presented in box plot, with average utilization annotated in bar chart (a), and normalized data access counts (b) across architectures with different *DataMaestro* features under the synthetic DNN workload set. ① = Baseline (all features turned off); ② = ① + Fine-grained prefetch; ③ = ② + Transposer; ④ = ③ + Broadcaster; ⑤ = ④ + Implicit *im2col*; ⑥ = ⑤ + Addressing mode switching.

all features of *DataMaestro*s, architecturally similar to plain data movement units, inside the accelerator system as the baseline (①). Subsequently, we gradually introduce each feature, eventually forming an accelerator system with fully-featured *DataMaestro*s (② to ⑥).

2) Result Analysis: Figure 7 (a) presents the GeMM core utilization distribution for each DNN kernel group using a box plot, with the average utilization inside each group depicted in the accompanying bar chart. As demonstrated in the results, ② achieves a significant increase in GeMM core utilization, ranging from 1.65× to 2.21× higher than the baseline ① across all workloads, thanks to fine-grained asynchronous prefetch. Among all features, the Transposer is particularly effective for the transposed GeMM workload (③ compared to ②), improving utilization by 1.16×, while Broadcaster increases utilization of all workloads by up to 1.09× (④ compared to ③). The implicit *im2col* [30] transformation further delivers a 1.19× utilization improvement for convolution workloads (⑤ compared to ④). Finally, with addressing mode switching (⑥), the utilization for the two GeMM workloads reaches 100%, with minimal variation across all matrix sizes. For convolution workloads, the average utilization reaches 92.03% but the utilization distribution exhibits notable variation across different workload sizes. Upon further analysis, this fluctuation is attributed to strided convolution layers, which require fetching noncontiguous input data across wider address ranges, resulting in unavoidable bank conflicts. Fortunately, strided convolution layers, typically used for feature map downsampling, comprise only a small portion of real-world DNN workloads and thus have a limited impact, as demonstrated in § IV-C2. Meanwhile, datapath extensions for on-the-fly data manipulation significantly reduce redundant memory accesses, mitigating bank conflicts and saving energy. As illustrated in Figure 7 (b), Transposer reduces the data access counts by 15.86% for transposed GeMM workloads (③ compared to ②), and Broadcaster effectively reduces memory accesses by up to 14.58% across all three workloads (④ compared to ③). Overall, the fully-featured *DataMaestro* (⑥) achieves up to 2.89× performance speedup and up to 21.15% reduction in memory accesses across all test cases, compared to the baseline (①). These results highlight that *DataMaestro* enables versatile and highly efficient data streaming to the accelerator across a variety of DNN kernels with different sizes.

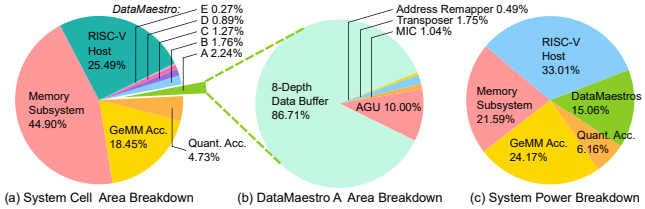
C. Real-world Neural Network Evaluation on FPGA

1) Evaluation Setup: We integrate our *DataMaestro* evaluation system into a customized SoC platform derived from [32] and prototype it on the AMD Versal™ VPK180 FPGA, which facilitates rapid and convenient real-world DNN evaluations. The annotated layout and resource utilization of our system are presented in Figure 8.



Platform	VPK180
Clock Frequency	125MHz
LUTs Total	265k
Regs Total	59k
LUTs GeMM	124k (46.79%)
Regs GeMM	8k (13.56%)
LUTs *DataMaestros*	14k (5.28%)
Regs *DataMaestros*	4.4k (7.46%)

Fig. 8: The FPGA implementation (left) and resource utilization (right) of the *DataMaestro* evaluation system.



(a) System Cell Area Breakdown (b) DataMaestro A Area Breakdown (c) System Power Breakdown

Fig. 9: Breakdown of cell area (a) and total power (c) for the evaluation system, and area composition of *DataMaestro A* (b).

2) DNN Performance Benchmarking: We benchmark ResNet-18 [33], VGG-16 [34], ViT-Base-16 [35], and BERT-Base [36] on our FPGA SoC system, with GeMM core utilization listed in Table III. All four networks achieve utilization above 95%, with VGG-16 and ViT-Base-16 reaching nearly 100%. This high GeMM core utilization across these real-world DNN workloads highlights the efficacy of *DataMaestro* in minimizing data stream interruptions, thereby improving the performance of the accelerator system.

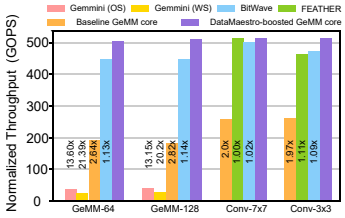
TABLE III: GeMM core utilization (in %) of *DataMaestro*-boosted accelerator under real-world DNN workloads on VPK180 FPGA.

	ResNet-18	**VGG-16**	**ViT-B-16**	**BERT-Base**
Type	CNN	CNN	Transformer	Transformer
Utilization (%)[†]	95.45	100.00	99.98	97.85

[†] Utilization is calculated as the ratio of theoretical computation cycles without memory stalls to the active cycles of *DataMaestro*.

D. Area and Power breakdown

When synthesized using 22nm FDX technology, our system occupies $0.61mm^2$ cell area and consumes $329.4mW$ of total power when executing an M=N=K=64 GeMM (GeMM-64) workload at 1 GHz, achieving a total system-level energy efficiency of 2.57TOPS/W. The detailed system area and power breakdowns are shown in Figure 9 (a) and (c), highlighting the five main components: the GeMM accelerator, Quantization accelerator, five *DataMaestro*s, the scratchpad memory subsystem, and the RISC-V host. The first four components form the core of the accelerator system, accounting for 74.52% of the total area and consuming 66.99% of the total power. The five *DataMaestro*s collectively occupy only 6.43% of the system area and 15.06% of the total power, with individual area usage ranging from 0.28% to 2.33%. The variation in area consumption demonstrates that *DataMaestro*'s design-time parameters alter the deployed hardware to meet the specific requirements of each data stream, showcasing its high versatility. Furthermore, to investigate the overhead of each feature inside *DataMaestro*, we conduct a detailed analysis of the area composition of a single *DataMaestro A*—the most advanced one among the five instantiations, as shown in Figure 9 (b). The majority of the area is occupied by data FIFOs (87.76%) and MICs (1.04%) to support decoupled fine-grained prefetch. The AGU, responsible for producing the 6-D temporal and 2-D spatial addresses, accounts for 10% of the total area. The Transposer takes up 1.75% of the total area. Lastly, the address remapper, which is essentially a multiplexer of permuted address bits, occupies a negligible 0.49% of the area.



	Area	Power
Buffet [6]	2%	14%
Softbrain [7]	4.3%	15.3%
Bitwave [2]	11.9%	25.5%
FEATHER [4]	8.9%	N/A
DataMaestro	**6.43%**	**15.06%**

Fig. 10: Comparison of normalized throughput (left) and data movement area and power cost (%) inside the whole accelerator system (right) between SotA solutions and *DataMaestro*. Some data is not revealed in the literature; therefore, the corresponding bars are omitted in (a).

E. State of the Art Comparison

1) Comparison with SotA DNN Dataflow Accelerators: We compare the throughput of our *DataMaestro*-boosted accelerator with SotA DNN dataflow accelerators, including Gemmini in output-stationary (OS) and weight-stationary (WS) modes [1], [37], FEATHER [4], and BitWave [2] using representative DNN kernels. For all systems, throughput is normalized to the same number of PEs (512) and clock frequency (1 GHz) of the accelerator for a fair comparison. As the results in Figure 10 (left) show, our *DataMaestro*-boosted accelerator achieves throughput gains ranging from 1.05× to 21.39× compared to SotA accelerators across diverse DNN kernels. This significant performance improvement results from the combined efforts of *DataMaestro*'s design-time and runtime flexibility, fine-grained prefetch, and addressing mode switching techniques.

2) Comparison with SotA Data Streaming Engines: We further compare data streaming engines in Buffet [6], Softbrain [7] and data movement units in BitWave [2], and FEATHER [4] with *DataMaestro* and summarize their area and power overhead results in Figure 10 (right). The five *DataMaestro*s in our evaluation system jointly occupy 5% and 15.58% of the system area and power consumption, which is competitive with SotA solutions, while flexibly and efficiently streamlining data to boost the dataflow accelerator's performance.

V. Conclusion

In this paper, we present *DataMaestro*, a versatile, efficient, and open-source data streaming unit that brings the decoupled access/execute architecture to DNN dataflow accelerators. *DataMaestro* supports a programmable N-Dimensional access pattern to accommodate diverse workload types and dataflows, incorporates fine-grained prefetch and addressing mode switching to mitigate bank conflicts, and enables customizable on-the-fly data manipulation to reduce memory footprints and accesses. We integrate five *DataMaestro*s with a Tensor Core-like GeMM accelerator and a Quantization accelerator into a RISC-V host system. FPGA prototyping and VLSI synthesis results demonstrate that *DataMaestro* helps accelerators achieve nearly 100% utilization, which is 1.05 – 21.39× higher than SotA solutions, with area and energy cost of merely 6.43% and 15.06% of the total system.

Acknowledgment

This project has been partly funded by the European Research Council (ERC) under grant agreement No. 101088865, the European Union's Horizon 2020 program (CONVOLVE) under grant agreement No. 101070374, the Flanders AI Research Program, and KU Leuven.

REFERENCES

[1] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[2] M. Shi, V. Jain, A. Joseph, M. Meijer, and M. Verhelst, "Bitwave: Exploiting column-based bit-level sparsity for deep learning acceleration," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 732–746.

[3] F. Schneider, M. Karagounis, and B. Choubey, "Energy and bandwidth efficient sparse programmable dataflow accelerator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[4] J. Tong, A. Itagi, P. Chatarasi, and T. Krishna, "Feather: A reconfigurable accelerator with data reordering support for low-cost on-chip dataflow switching," in *Proceedings of the 51th Annual International Symposium on Computer Architecture*, ser. ISCA '24. Argentina: Association for Computing Machinery, 2024.

[5] F. Schuiki, F. Zaruba, T. Hoefler, and L. Benini, "Stream semantic registers: A lightweight risc-v isa extension achieving full compute utilization in single-issue cores," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 212–227, 2020.

[6] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 137–151.

[7] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam, "Stream-dataflow acceleration," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 416–429.

[8] P. Houshmand, G. M. Sarda, V. Jain, K. Ueyoshi, I. A. Papistas, M. Shi, Q. Zheng, D. Bhattacharjee, A. Mallik, P. Debacker, D. Verkest, and M. Verhelst, "Diana: An end-to-end hybrid digital and analog neural network soc for the edge," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 203–215, 2023.

[9] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. P. Jouppi, and D. A. Patterson, "Google's training chips revealed: Tpuv2 and tpuv3." in *Hot Chips Symposium*, 2020, pp. 1–70.

[10] C.-Y. Du, C.-F. Tsai, W.-C. Chen, L.-Y. Lin, N.-S. Chang, C.-P. Lin, C.-S. Chen, and C.-H. Yang, "A 28nm 11.2 tops/w hardware-utilization-aware neural-network accelerator with dynamic dataflow," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 1–3.

[11] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti *et al.*, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[12] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[13] Q. Huang, P.-A. Tsai, J. S. Emer, and A. Parashar, "Mind the gap: Attainable data movement and operational intensity bounds for tensor algorithms," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 150–166.

[14] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[15] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1160–1174, 2021.

[16] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 37–47.

[17] Y. Tortorella, L. Bertaccini, L. Benini, D. Rossi, and F. Conti, "Redmule: A mixed-precision matrix–matrix operation engine for flexible and energy-efficient on-chip linear algebra and tinyml training acceleration," *Future Generation Computer Systems*, vol. 149, pp. 122–135, 2023.

[18] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 789–801.

[19] J. E. Smith, "Decoupled access/execute computer architectures," *ACM SIGARCH Computer Architecture News*, vol. 10, no. 3, pp. 112–119, 1982.

[20] J. M. Domingos, N. Neves, N. Roma, and P. Tomás, "Unlimited vector extension with data streaming support," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 209–222.

[21] Z. Wang and T. Nowatzki, "Stream-based memory access specialization for general purpose processors," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 736–749.

[22] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," 2017. [Online]. Available: https://arxiv.org/abs/1703.09039

[23] N. Corporation, "Nvidia turing architecture whitepaper, https://images.nvidia.com," 2018.

[24] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "Hybriddnn: A framework for high-performance hybrid dnn accelerator design and implementation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[25] M. Shi, S. Colleman, C. VanDeMieroop, A. Joseph, M. Meijer, W. Dehaene, and M. Verhelst, "Cmds: Cross-layer dataflow optimization for dnn accelerators exploiting multi-bank memories," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[26] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *ACM SIGOPS Operating Systems Review*, vol. 25, no. Special Issue, pp. 63–74, 1991.

[27] H. Zheng, S. Oh, H. Wang, P. Briggs, J. Gai, A. Jain, Y. Liu, R. Heaton, R. Huang, and Y. Wang, "Optimizing memory-access patterns for deep learning accelerators," *arXiv preprint arXiv:2002.12798*, 2020.

[28] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, 2000, pp. 32–41.

[29] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845–1860, 2020.

[30] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 214–225.

[31] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[32] G. Paulin, P. Scheffler, T. Benz, M. Cavalcante, T. Fischer, M. Eggimann, Y. Zhang, N. Wistoff, L. Bertaccini, L. Colagrande *et al.*, "Occamy: A 432-core 28.1 dp-gflop/s/w 83% fpu utilization dual-chiplet, dual-hbm2e risc-v-based accelerator for stencil and sparse linear algebra computations with 8-to-64-bit floating-point support in 12nm finfet," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[34] K. Simonyan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[35] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[36] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[37] A. Gonzalez, J. Zhao, B. Korpan, H. Genc, C. Schmidt, J. Wright, A. Biswas, A. Amid, F. Sheikh, A. Sorokin *et al.*, "A 16mm 2 106.1 gops/w heterogeneous risc-v multi-core multi-accelerator soc in low-power 22nm finfet," in *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2021, pp. 259–262.

Power-Grid Structure Exploration with Unified Sequence-based Learning Framework

Yi-Lin Chuang, Hao-Wei Chan, Chih-Yun Yen, Shih-An Hsieh, Ching-Feng Chen and Sheng-Te Lai
MediaTek Inc., HsinChu, Taiwan
nick-yl.chuang@mediatek.com

***Abstract*—As increasing gap of the shrinking ratio between device and metal pitches under advanced nodes, power-grid (PG) structure plays a critical role in circuit performance considering power integrity. Initial PG structures, as initiating implementation fundamentals, dominate optimization space. To address initial structures on industrial flow, we propose *the first* unified framework fueled by novel sequence-based structure generator and Transformer-based predictor, providing accurate static voltage-drop estimation. Learned embeddings are then adopted to determine promising candidates with Pareto frontier during structure optimization. The predictor is demonstrated in different scenarios under $3nm$ and $2nm$ technology with average 0.011% maximum absolute error of drop percentage while the optimized structures reduce 15% PG utilization with 34% worst timing-slack improvement on industrial designs.**

I. INTRODUCTION

As transistor size keeps shrinking, PG structures impose a crucial role in circuit performance for increasing voltage (IR) drop and reserving sufficient routability. In industrial design flow, designers carefully design a set of global PG structures with different sustainable static IR (SIR) drops to initiate physical implementation.

Since global PG design is conducted before real cell placement, SIR is usually adopted as healthiness index, and different power densities (PD) of IP designs lead to rich possibilities of PG structures. In industrial practice, designers could mix different power density structures to address local hotspots or dynamic IR drop after cell placement with much smaller design overheads.

A complete structure is defined by an architecture and corresponding pattern configuration. The architecture determines PG-style combination in stripes or pillars as illustrated in Fig. 1(a). As observed in the figure, layer architecture has various forms and the architecture hooks up PG metals among layers through proper via connections. To avoid PG occupying excessive routing resource under advanced nodes, stagger and inline pillars are the most common stub architectures, as shown in Fig. 1(b) and (c). On the other hand, the pattern configuration further details how to implement such architecture, e.g., the needed vertical spacing (pitch y) between two VIA4 of an inline pillar in Fig. 1(c).

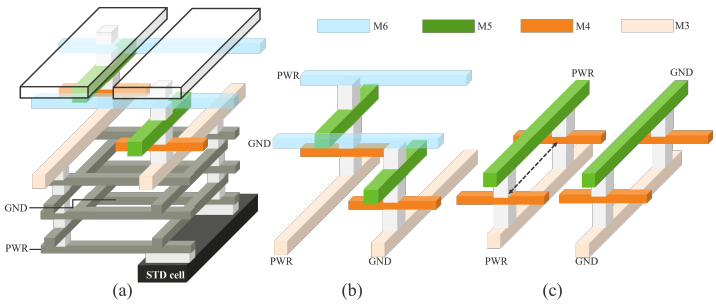


Fig. 1: Power-grid structure overview. (a) Grid stacking adopted in advanced nodes with two most common pillar architectures, (b) stagger and (c) inline pillar.

Typically IR analysis models PG as a resistor network and attaches corresponding current loads at each network node. To evaluate the quality of a global PG structure, the common practice assigns average power density simulating current loads with commercial sign-off tool. During physical implementation, finer-grain adjustment based on a global structure is made to address dynamic IR drop from local hotspots. As a result, the global PG structure has significant impacts in physical convergence.

Considering global structure optimization, it is mandatory to address the requirement of increasing and diverse power densities under deep sub-micron technology. Power density increases dramatically because of smaller logic area but much complicated circuit functions. In addition, SoC IPs have different power densities, causing unnecessary area waste or systematic IR-drop risk with single structure.

To tackle aforementioned two problems, scalable structure exploration is essential. Existing related works include two major categories, (1) IR-change prediction of different cell placements under one fixed PG [2], [4], [5], [7], [9], [11], [12], [15]–[17], and (2) PG structure adjustment with limited styles under different cell placements [1], [8], [10]. First category focuses on accurate prediction but does not address much about how to reduce IR violations. Second category aims at PG adjustment at specific current distribution strengthening local robustness; however, the explored structures are difficult to consider rich pillar combinations.

There are multiple learning models proposed in related works, including linear regression [16], XGBoost [8], [12], and U-Net [4], [5], [15]. A recent work [17] adopted tree-based model to make prediction while examining feature importance by attention calculation and pruning non-important features. Although these models are effective in static or dynamic IR prediction, we found that the model sensitivity on finer-grain structures is insufficient. Considering PG adjustment in existing works, only specific layer or style changes are made, and thus it is infeasible to be adopted in industrial cost-effective designs.

In this work, we propose scalable learning-based PG structure exploration, with SIR-driven global structure optimization, to offer sufficiently good initial solutions with various power densities reducing adjustment iterations. In industrial practice, it is less preferred changing global PG structures constantly to reset floorplan utilization. Moreover, with various density offerings, we could easily address local dynamic hotspots by mixed density structures.

Our framework addresses multi-density optimization with different structures recommended from Transformer-based learning model. To enable flexible structure exploration, sequence-based architecture perturbation is developed followed by sets of detailed pattern configuration generation. The proposed sequence-based structure operations are carefully designed to run through the model training and optimization. Compared with existing IR-drop models, this is *the first* work to formulate the structure features in transformer-based architecture, where the attention mechanism is further strengthened on filtered samples by Maximin Latin Hypercube (MLHS). Our model effectively captures inter-layer relationships over common regression models to greatly improve IR-drop prediction. Promising structure patterns are generated through non-

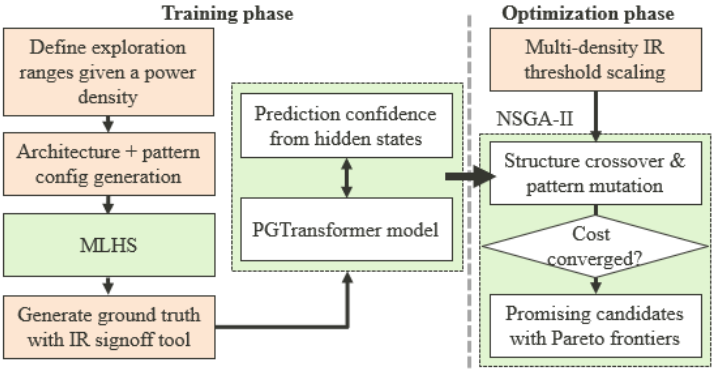


Fig. 2: Proposed framework to create optimized PG structures.

dominated sorting derived from augmented drop prediction and associated resources.

We summarize the key contributions of our proposed framework as follows:

- Compared with limited structure supports in related works, our explorer builds flexible PG pattern stacking in sequence representation to generate vast candidates followed by MLHS for representative training samples.
- To better capture subtleties of multi-layer structure changes over existing works, different pattern configurations are translated into a sequence each layer. We present PGTransformer on proposed sequence representation as core predictor to capture inter-layer relationships with self-attention mechanism.
- Proposed framework equips capability to generate optimized *multi-layer* structure patterns from Pareto frontiers. New patterns are derived from balanced trade-off between augmented drop value with prediction confidence and resource.

The rest of this paper is organized as follows. Sec. II illustrates our exploration framework and key procedures. Section III details our algorithms. Experimental results are presented in Section IV. Finally, conclusions are given in Section V.

II. Proposed Unified Framework

To offer efficient structure optimization, the framework combining both training and optimization is proposed as illustrated in Fig. 2. The purpose of training phase builds a robust IR-drop prediction model to examine power-integrity quality of various structures. The optimization phase adopts the trained model along with resource and prediction confidence to explore different architecture patterns.

In model training phase, we transform multi-layer PG structures into a set of sequences for efficient perturbation and enable sequence-based model to learn structure contexts. Legal layer configurations are defined depending on target metal scheme. For each candidate layer, a valid pitch search range is defined, and we set the range in $\{8, 16, \ldots, 48\}$ times of default pitch in this work. The search range can be changed to further expand solution space or fit target technology rules.

With defined search range, our proposed structure generator creates various layer pattern stacking with pillar template combinations. The stacking operations are conducted with string manipulation and thus the generator provides highly efficient sample generation. However, to manage costly training sample collection with sign-off IR values, it is mandatory to reduce training sample sizes. After sets of structures are generated, we apply MLHS from multivariate structure parameter distribution for equidistant sample space. Commercial IR sign-off tool is then adopted to generate golden IR values of sampled training structures. The IR predictor is trained with Transformer-based architecture by forwarding sets of layer features to its embedding encoder.

As mentioned in Sec. I, it is essential to prepare structures considering different power densities for minimized area overhead. Considering IR drop mainly determined by the current multiplication, direct proportional to assigned power, IR drop is inversely proportional to structure sustaining certain power density. As a result, we expand power density supports during optimization phase by changing target drop threshold as

$$D_{th}^{j} = \frac{PD_i}{PD_j} \times D_{th}^{i}, \tag{1}$$

where PD_i is the structure sustaining power density i and D_{th}^{i} is the target IR-drop threshold under power density i.

In major optimization iterations, we intend to figure out most promising structures approaching D_{th}^{j} with best trade-off between resource and IR quality. Multi-objective optimization with Non-dominated Sorting Genetic Algorithm (NSGA-II) [6] is adopted to address the sequence-based structure representation. The optimizer identifies Pareto frontiers considering reserved routing resource, and augmented IR prediction by comparing learned embeddings.

III. Proposed Algorithm

A. Sequence-based Structure Generation

Proposed structure generation aims to create flexible structures among all layers efficiently. The generation includes two stages, vertical stacking to determine an architecture and pattern configuration mapping within the architecture. During generation, we define a set of discrete candidates to determine the width, number, and spacing of PG metals at each layer considering technology rules. To address metal-pitch alignment as a feasible PG, template-based permutation is proposed.

A set of pillar architecture templates is defined as illustrated in Fig. 3 and note that similar concepts can be easily extended to more complicated template designs. Each pillar template AT_i defines how a pillar is formed between two stripes under stacking layers i. To provide flexible combination, the stacking could be either bottom-up AT_i^{R0} or top-down AT_i^{MX}. Given stacking direction R0, MX, the corresponding x-axis and y-axis pitch are revised based on the preferred direction of the metal layer where template stripe lies. Pitches x and y describe cross-layer alignment with respect to an anchor layer, which could be inside a template or defined by other stacking one.

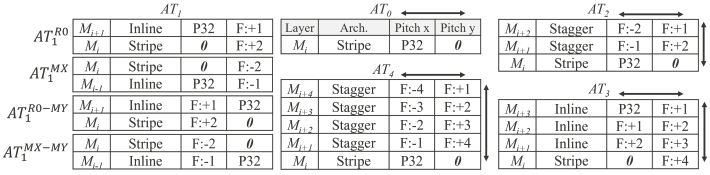


Fig. 3: Pillar template examples considering five-layer combinations.

At first stage, architectures are constructed with defined templates based on sequence permutation as depicted in Fig. 4(a). The sequences manage possible pillar template combinations in vertical stacking direction, comprising pairs of two-digit characters. The first character denotes the template index i among candidate set AT and second one describes the stacking direction, "D" for bottom and "U" for top. Every stacking needs to make template pillars enclosed by two stripes considering stacked templates or boundary conditions for a feasible pillar structure. As the example described in the figure, AT_2^{MX} is chosen and stacked from the bottom side at first operation, and similar concepts are applied to second and third operations. For third operation, although the stacking direction is from the bottom, AT_3^{MX} is chosen instead of AT_3^{R0} to enclose template pillars.

After a complete architecture is formed, detailed pattern configurations are generated at second stage. To form a valid pillar pattern, we define proper pitch relationships among related neighboring layers. As depicted

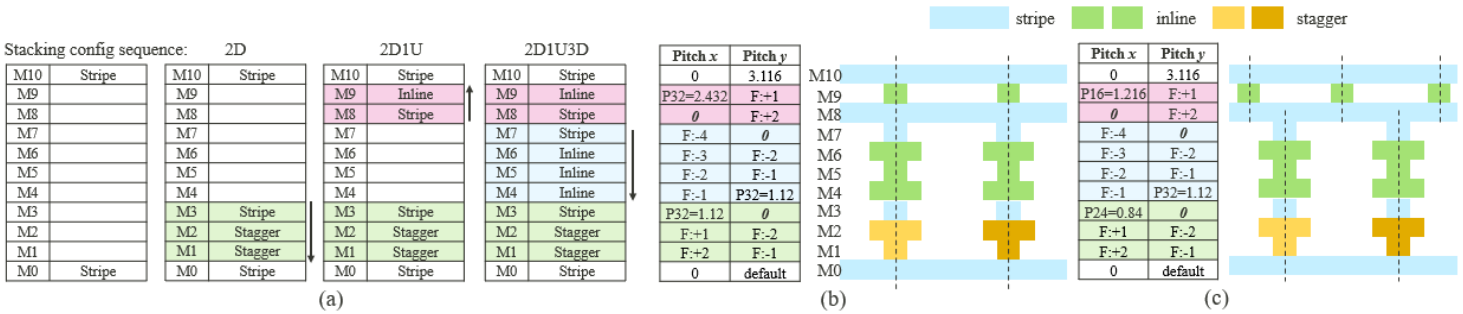


Fig. 4: Architecture construction and corresponding pattern configuration mapping. (a) Architecture construction with template stacking. (b) One pattern configuration candidate, and (c) another configuration by changing anchor pitches under an identical architecture.

in Fig. 4(b), certain template layers define drawing PG pitches while the pitches of other layers "follow" different anchor layers to create pillar patterns. The drawing pitch is a set of discrete numbers, described by "$\texttt{P}m_{tp}$" where m_{tp} denotes the multiple of default layer pitch. The follow relationship is described by a string with an important index "$\texttt{F} : d_{floor}$", where d_{floor} denotes the floor offset with respect to the alignment anchor layer, e.g., 0 implies current layer, 1 (-1) implies one layer above (below).

As illustrated in the example, considering vertical preferred direction, the pitch x of template string at `M7` matters, and thus AT_3^{MX} is further revised to AT_3^{MX-MY} defining explicit horizontal PG pitch. Same concepts are also applied to other two templates. After template revision, by tracing follow relationships, it is clear that the pattern configuration can be manipulated through different anchor pitches, i.e., $1.12um$ from 32X `M4` layer pitch $0.035um$. By changing anchor pitches on `M3` and `M9` in the figure, another different patterns are formed given one identical architecture as depicted in Fig. 4(c).

Pitch configurations among all layers are then translated to sets of explicit numbers. Combined with aforementioned PG-layer width, number, and spacing, vast amounts of implementable structure configurations can be easily generated with Cadence *PSDL*, PG Structure Description Language.

B. Feature Designs and Sampling

After pattern configuration is translated, the style and numbers at each layer are formed as a sequence of features. The features used for model training and prediction in this work are listed in Table I. The features directly derived from explicit configuration well describe a PG structure to help our predictor capture accurate inter-layer weights.

In addition to the values in pattern configuration, we also calculate MN_i and VN_i to facilitate model learning electrical behaviors because via counts are highly correlated with power-integrity robustness. Considering vertical preferred direction of Layer i, and S_{pillar} is the set containing stagger and inline styles given a floorplan dimension W_{fp} by H_{fp},

$$MN_i = \begin{cases} \frac{W_{fp}}{Pitch_{x,i}} \times \frac{H_{fp}}{Pitch_{y,i}} \times GN_i, & \text{if } Arch_i \in S_{pillar} \\ \frac{W_{fp}}{Pitch_{x,i}} \times GN_i, & \text{otherwise} \end{cases} \quad (2)$$

and then $VN_i = MN_i \times \rho$, where $\rho = GN_i$ if either one via layer belongs to S_{pillar} or $\rho = MN_{i+1}$.

To manage costly training sample collection with ground truths from commercial IR sign-off tool, it is essential to have minimized dataset that adequately represents majority feature space. After a set of random pattern configurations are generated, MLHS is adopted to figure out minimized set of training samples.

MLHS begins by generating a random Latin Hypercube sample, and then iteratively improves the sample by swapping pairs of data points to increase the minimum distance between them. The swapping process continues until no further improvement can be made. Considering the features comprising both numerical and categorical types, we measure their distances differently. The pairwise distance between two samples s_i and s_j, is defined as follows,

$$d_\lambda(s_i, s_j) = \sqrt{\sum_n |s_i^n - s_j^n|^2} + (1-\lambda) \times \sum_c |s_i^c - s_j^c| \quad (3)$$

where s_i^n and s_i^c represents the numerical and categorical features, respectively, and λ is the weight to balance these two types. We intend to find the minimal distance from sample set $\mathcal{S}$ with total N features,

$$d_\lambda(\mathcal{S}) = min(\{0 \le i < j \le N : d_\lambda(s_i, s_j)\}) \quad (4)$$

Different λ in a discrete step ranged in [0, 1] are swept with random generated sample candidates, and the sample set with the maximal $d_\lambda(\mathcal{S})$ among iterations is chosen as final training set.

TABLE I: Definition of important features used in power/ground nets on Layer i. Power and ground nets could have different values.

Notation	Type	Description
$Arch_i$	Cat.	Architecture type (e.g., stripe, stagger, inline)
W_i	Num.	Metal width of this PG net
L_i	Num.	Metal length of this PG net
GN_i	Num.	Metal number in one group
GS_i	Num.	Spacing between metals within a group when $GN_i >= 2$
OS_i	Num.	First metal offset within a group
$Pitch_{x,i}$	Num.	Horizontal pitch between metal groups
$Pitch_{y,i}$	Num.	Vertical pitch between metal groups
R_i	Num.	Resistance of Layer i
MN_i	Num.	Total metal counts of this PG net
VN_i	Num.	Total via counts of this PG net

C. PGTransformer Architecture

As described in Sec. III-B, each layer of a PG structure is characterized by distinct features such as pitch, spacing, and width. We formulate layer features into a sequence with length L and trained on Transformer-based model [14] for its superior capability of capturing temporal dependencies in sequence data. Specifically, the sequence representing a structure with L layers is defined as $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_L\}$ where $\mathbf{x}_i = \{Arch_i, W_i, \cdots\}$.

Transformer architecture comprises N blocks, each containing a multi-head attention layer, a fully connected feed-forward network, and two normalization and residual addition layers. The Transformer can be viewed as a function TF($\cdot$): $\mathbb{R}^{L\times d} \rightarrow \mathbb{R}^{L\times d}$ that accepts a sequence of length L of input tokens $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$, $\mathbf{x}_i \in \mathbb{R}^d$ as input, and outputs a sequence of token embeddings $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_L\}$, $\hat{\mathbf{x}}_i \in \mathbf{R}^d$, where each $\hat{\mathbf{x}}_i$ encodes the dependencies between $\mathbf{x}_i$ and other input tokens.

The detailed structure of the Transformer encoder is depicted in Fig. 5(a). Positional embeddings are added to $\mathbf{X}$ to encode the position information of each token. The multi-head self-attention layer with total H heads, learns independent transformations $\mathbf{W}_h^Q$, $\mathbf{W}_h^K$, and $\mathbf{W}_h^V \in \mathbb{R}^{d\times d}$ at the h-th head and calculates the attention weights $\mathbf{A}_h$ among all input tokens. Token embeddings are updated with the context information

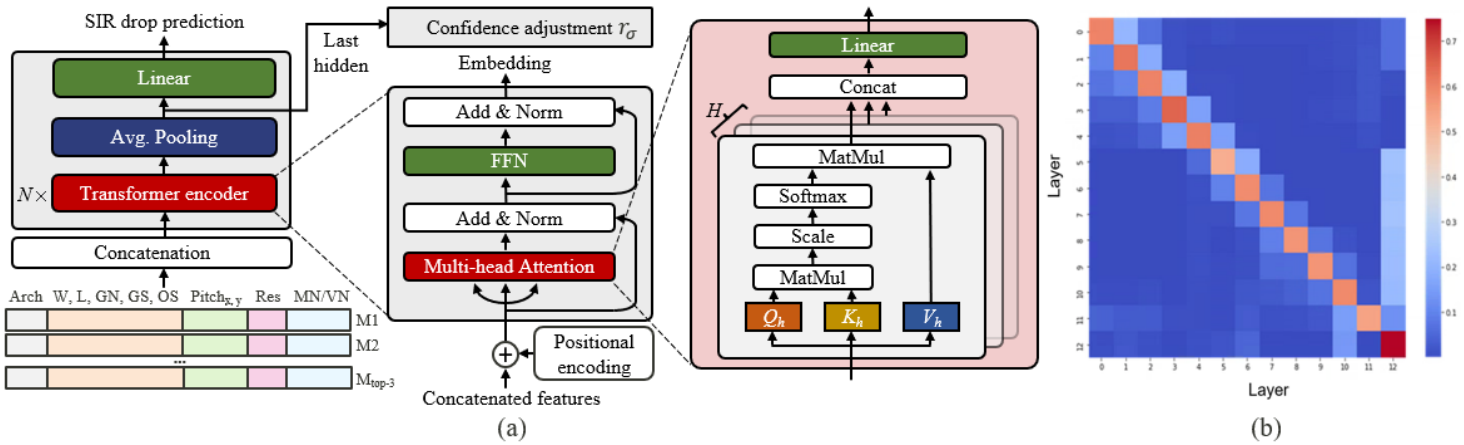


Fig. 5: Proposed PGTransformer architecture. (a) Encoder and linear layer for IR-drop prediction, (b) and the heatmap of cross-layer attention after training.

from their attention tokens, and resulting embeddings from all heads are concatenated and projected to the input dimension via learnable matrix $\mathbf{W}^O \in \mathbb{R}^{(H\times d)\times d}$. For each $x_i \in \mathbf{X}$, the output embedding $\hat{x_i}$ through the multi-head attention, can be derived as follows,

$$\mathbf{A}_h\left(x_i, x_j\right) = \text{softmax}\left(\frac{\mathbf{W}_h^Q\mathbf{X}\cdot\left(\mathbf{W}_h^K\mathbf{X}\right)^T}{\sqrt{d}}\right) \tag{5}$$

$$\hat{\mathbf{X}} = \text{concat}_{h=1}^{H}\left\{\sum_{j=1}^{N}\mathbf{A}_h\left(x_i, x_j\right)\mathbf{W}_h^V\mathbf{X}\right\}\cdot\mathbf{W}^O \tag{6}$$

Output embedding $\hat{\mathbf{X}}$ is then passed through a feed-forward network (FFN) for non-linear mapping, allowing the model to capture complex dependencies of different categorical and numerical features in a input sequence. For output layer, our model applies AvgPooling on learned embeddings transforming $\mathbb{R}^{N\times d}$ to $\mathbb{R}^{1\times d}$, and subsequently followed by a linear layer projecting to final IR prediction value $\in \mathbb{R}^1$. In this work, the embedding dimension is 64 and FFN dimension is 256 with $N = 2$ and $H = 8$. The encoder produces hidden states with a dimensionality of 64, which are further used to adjust prediction values by proposed quadratic error regressor in Sec. III-D.

The heatmap shown in Fig. 5(b) represents the attention map of internal attention states. The i-th row corresponds to the attention map to predict the IR value of Layer i. In addition, it is clear that the model also refers to states from all layers to form its prediction. The layer influences are adaptively adjusted through our sequence modelling.

D. Structure Optimization

Multi-objective Pareto frontier is adopted to create optimized structures. To reach best design performance, not only IR robustness but left signal routing resource needs to be considered. To maximize left resource, the PG utilization should be minimized. Two objectives, PG resource utilization c_{res} and augmented IR-drop prediction c'_{ir} incorporating prediction value c_{ir} and associated confidence, are formulated in our structure optimizer. The PG utilization estimation considering architecture is defined as,

$$c_{res} = \left(W_{fp}\times H_{fp}\right)^{-1}\sum_{i}^{L} MN_i \times A_i \times TP_i, \tag{7}$$

where $A_i = L_i \times W_i$ and TP_i is default pitch divided by default width of Layer i.

As PG structures have vast variants and thus not every prediction has identical accuracy, the prediction confidence measured from model hidden states is proposed to provide another perspective during structure cost evaluation. As the last layer of PGTransformer is a linear layer, the mapping from the hidden state to final prediction is also linear mapping. If the hidden state is far from the distribution of training data, it implies

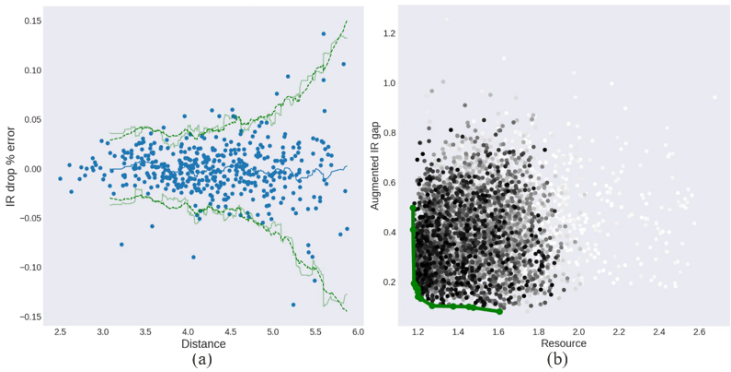


Fig. 6: Confidence cost quadratic regression and Pareto frontiers of structure optimization. (a) Rolling error mean curves with window 40 regarding $d(x_i, S)$, and dark green curves are proposed quadratic polynomial regressor. (b) Optimized structure frontiers after iterations.

that the prediction could be extrapolated, leading to unreliable results. We approximate prediction confidence by fitting σ-curve of hidden-state distance during model validation.

For each validation sample x_i, the minimum distance of its hidden states toward training dataset S as $d(x_i, S) = min\{\|x_i - s_j\|\}$ and its associated prediction error are measured. As shown in Fig. 6(a), the validation error along with different distances is illustrated, where x-axis is $d(x_i, S)$. The blue dots (y-axis) represent validation errors of IR-drop percentage, V_{ir}/V_{ideal} where V_{ideal} is supply voltage and V_{ir} is the drop value. To capture proper confidence penalty, the corresponding rolling mean and standard deviation σ are examined. The blue curve denotes corresponding rolling mean μ and light green curves are $\mu \pm 2\sigma$ from validation set. By observing $\mu \pm 2\sigma$ curves, we adopt quadratic regression to determine the parameters as confidence cost function.

As observed in the figure, the 2-σ quadratic regression as confidence evaluation is positively correlated with both distance and original $\mu \pm 2\sigma$ distribution with 95% sample coverage, reaching correlation coefficient 0.82 and R-squared value 0.92, respectively. To address the confidence directly into predicted IR toward a defined drop threshold D_{th}, we define the IR-drop objective in our structure optimizer as

$$c'_{ir} = |c_{ir} - D_{th}| + r_\sigma(d(x_i, S)), \tag{8}$$

where r_σ is 2σ quadratic regressor of minimum state distance.

With c_{res} and c'_{ir}, we employ modified NSGA-II as multi-objective optimizer. NSGA-II works by generating a population of candidate solutions and evolving population over generations. The architecture sequence is used to form chromosomes, and corresponding pillar templates provide

good crossover points. The crossover points offer strategic perturbation, potentially combining both parent strengths.

Fig. 7 illustrates the configuration evolution process. The pairs of template stacking with equal length from each configuration pair are collected as crossover candidates. One of the pairs is randomly selected to produce new configurations. After architecture crossover, the valid pitches x and y are examined based on the preferred direction of template stripe layers, as described in bold blue in the example. In addition, the anchor pitches of two child configurations are changed with mutation probability n_{AT}^{-1}, where n_{AT} is total template number of parent configurations. With aforementioned operations, structure configurations are gradually optimized.

After the optimization converges, we determine a set of "non-dominance" candidates with Pareto frontiers. In the context of multi-objective optimization, these frontiers represent the set of all Pareto optimal solutions. Considering c_{res} and c'_{ir} as our objectives, a structure x^* is considered Pareto optimal if there does not exist another solution x such that $c_{res}(x) \leq c_{res}(x^*)$ and $c'_{ir}(x) \leq c'_{ir}(x^*)$ for both objectives and $(c_{res}(x^*), c'_{ir}(x^*))$ is strictly better than at least one objective than other structures.

As depicted in Fig. 6(b) with two axes c_{res} and c'_{ir}, scattering points represent gradual converging structure candidates toward D_{th}, and iterations are illustrated in white toward black dots. Among these candidates, the frontiers are identified in green lines to provide opportunities to best fit different design characteristics.

IV. EXPERIMENTAL RESULTS

The prediction accuracy of PGTransformer is examined compared with several prominent models, and then optimization results empowered by the model sustaining different power densities are discussed.

A. Model Accuracy Comparison

To validate flexibility of the proposed framework, we prepared five sets of data in different cell heights, top-metal mesh designs, and metal schemes in TSMC N3E and N2 process nodes. The designs are implemented in flip-chip package with $155um$ bump pitch and supply voltage is set to $0.75V$. Separate models are trained for each of these two nodes.

Proposed structure generator randomly generates 12000 structure samples, and 3600 samples are selected by MLHS at each set where ground truths are generated by sign-off tool Cadence *Voltus*. For one set, the pool generation and sampling are done in 10 minutes from efficient string manipulation depicted in Sec. III-A, and ground truth collection takes 118 hours with distributed 2500 machine CPUs. The dataset uses 3:1 to partition into training and validation sets, respectively.

The model is implemented in PyTorch and trained on NVIDIA RTX 3090 with batch size of 16. We employ Lion [3] as the optimizer and set the weight decay to $1e^{-2}$. To train PGTransformer, we adopt an initial learning rate of 0.0001 with a cyclic learning rate schedule [13] and training epoch is set to 2500. For model comparison, three model architectures in related works, U-Net, XGBoost, and LSTM are also implemented.

To properly utilize U-Net, PG patterns and target PD are converted to tile-based map as [5], and we employ a bi-directional LSTM to align with the two-layer encoder in PGTransformer. For XGBoost, we use linear booster, max depth 10, subsample 1, eta 0.1, and number of boost round 100 with same learning rate. The structure generator and Pareto frontier are implemented in TCL and Python with 24-core Intel Xeon CPUs at 2.7GHz, respectively.

Table. II shows the validation error comparison of predicted IR-drop percentages. Column "M_{top}" describes the top-mesh PG in either mesh or mixed with signal routes. Five metrics are used to measure the quality, mean absolute error (MAE), root mean square error (RMSE), mean absolute percentage error (MAPE), and Pearson correlation coefficient (CC) and passing rate (PR). The PR is defined as the percentage among entire validation set where the gap between predicted drop percentage and ground truth lies in defined constraint (0.1%, 0.05%, 0.03% over ideal supply voltage). Column "IT" describes model inference time on one structure.

It is clear that PGTransformer achieves smallest errors among different data scenarios. Through the sequence formulation, we further strengthen PGTransformer capturing temporal dependencies in variable-length layer features. This trait is particularly useful to understand inter-layer IR impacts with subtle pattern changes over XGBoost and U-Net, and LSTM with potential vanishing gradient problem.

In addition to the smallest loss compared with other models, Fig. 8 also shows better convergence during validation. As observed, PGTransformer consistently reaches converged and smallest loss after 500 epoch; however, XGBoost and LSTM even have worse loss before 1000 (500) epoch than U-Net in N2 (N3E).

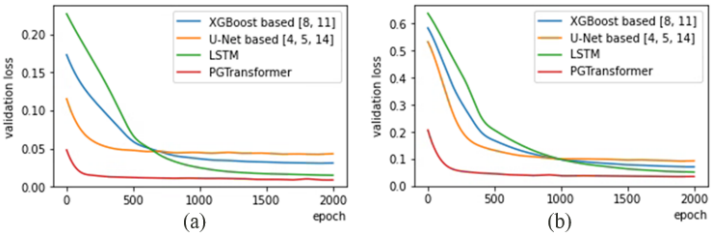


Fig. 8: Comparison of validation loss convergence between PGTransformer and prominent models in, (a) N3E 143H under 14M mixed M_{top}, and (b) N2 H130 under 16M mesh M_{top}.

B. Comparison with Well-Optimized Manual PG

To examine the effectiveness of structure optimization, the optimization framework is adopted on industrial designs compared with well-optimized manual works used in a N3E production and N2 test chip. The design statistics and target power density are summarized at left-hand side of Table III including high-speed CPU and power-efficient SoC IPs. Column "Period (ns)" denotes the target clock period, "CH" denotes adopted cell height, "PD" denotes target power density (mW/mm^2), "UR (%)" denotes initial floorplan utilization setup, "#Gates (10^6)" and "#Nets (10^6)" are total instance and net counts respectively.

We set base N3E IR threshold to 1.9% with PD 6000 and N2 1.7% over ideal supply voltage to align the objective of manual PG, and corresponding IR thresholds of other densities are scaled following Eqn. 1. Physical implementation with PGs are initiated with commercial tool Cadence *Innovus* in identical flow setup and adopt Synopsys *PrimeTime PX* to report power.

The right-hand side of Table III illustrates the circuit performance (QoR for short) comparison between manual PG and the optimized one achieving smallest WNS from Pareto frontiers. Column "$V_{ir}(\%)$" denotes reported drop percentage from *Voltus*, "$D_{mx}(\%)$ (D_{my})" denotes total PG metal density lower (above) `M5`, "Area (um^2)" denotes post-route cell area, DRC is routing violation count, and "Opt time (min)" depicts the time reaching an optimized PG. As shown in the table, after the optimization, a new structure is created with comparable IR-drop margin while releasing more routing resource and better timing. The framework also offers great efficiency improvement to facilitate global PG designs, where the Pareto optimization with presented sequence manipulation takes average 65 minutes to converge over 2–3 week manual efforts.

Through our optimization empowered by PGTransformer, the sign-off IR quality is closer toward the defined threshold for better trade-off between threshold constraint and resource utilization. For `design 5`, we even identify a solution satisfying the threshold than manual one. The generated structures with averagely comparable IR-drop margins use less PG resources of lower layers by 10% and upper layers by 21%, leading to better pin access and timing optimization. With more released routing

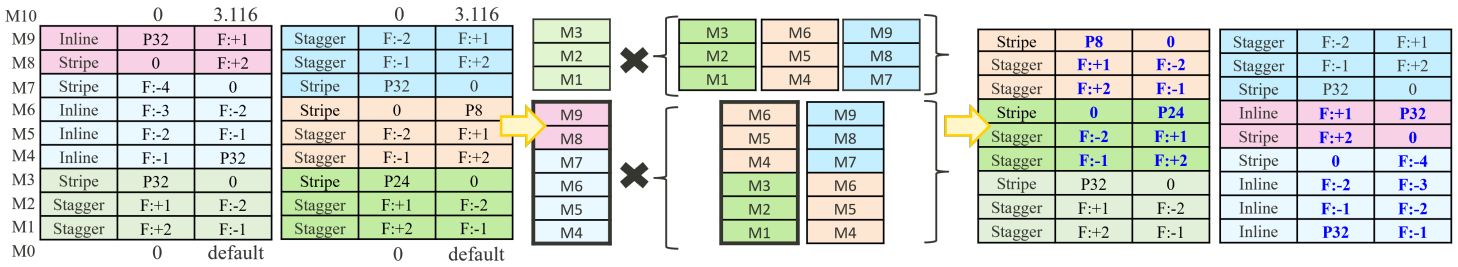


Fig. 7: Key structure crossover procedures used in modified NSGA-II to produce next generation.

TABLE II: Prediction accuracy comparison between PGTransformer and state-of-the-art models, showing PGTransformer better captures subtle structure changes. Note that it is inapplicable to compare exact models in previous works because the needed feature set and assumption (e.g., features only exist in dynamic IR analysis) are different. For fair comparison, key model architectures in related works are implemented with identical feature source.

Cell height	Scheme	M_{top}	Model	MAE	RMSE	MAPE	CC	PR $\leq$ 0.1%	PR $\leq$ 0.05%	PR $\leq$ 0.03%	IT (sec)
N3E 143H	13M	Mixed	U-Net based [4], [5], [15]	0.0142	0.0179	0.0075	0.9348	1.0000	0.9840	0.9280	0.0013
			XGBoost based [8], [12]	0.0168	0.0207	0.0089	0.9304	0.7040	0.3360	0.1840	0.0142
			LSTM	0.0091	0.0125	0.0048	0.9976	1.0000	1.0000	0.9680	0.0061
			PGTransformer	0.0060	0.0100	0.0032	0.9985	1.0000	1.0000	0.9838	0.0035
N3E 195H	13M	Mesh	U-Net based [4], [5], [15]	0.0059	0.0090	0.0058	0.8758	0.8645	0.8294	0.7824	0.0011
			XGBoost based [8], [12]	0.0052	0.0080	0.0045	0.9189	0.9669	0.8430	0.5207	0.0105
			LSTM	0.0028	0.0042	0.0024	0.9983	1.0000	1.0000	1.0000	0.0021
			PGTransformer	0.0023	0.0040	0.0023	0.9988	1.0000	1.0000	1.0000	0.0034
N3E 143H	14M	Mixed	U-Net based [4], [5], [15]	0.0426	0.0575	0.0119	0.8948	0.5351	0.2813	0.1120	0.0013
			XGBoost based [8], [12]	0.0309	0.0415	0.0096	0.9077	0.5932	0.3051	0.2203	0.0112
			LSTM	0.0151	0.0208	0.0047	0.9951	1.0000	0.9661	0.8644	0.0043
			PGTransformer	0.0088	0.0106	0.0034	0.9973	1.0000	0.9792	0.9153	0.0083
N3E 143H	14M	Mesh	U-Net based [4], [5], [15]	0.0107	0.0261	0.0089	0.8226	0.8589	0.7834	0.4829	0.0014
			XGBoost based [8], [12]	0.0069	0.0155	0.0053	0.8975	1.0000	0.8333	0.6500	0.0131
			LSTM	0.0035	0.0072	0.0025	0.9929	1.0000	0.9234	0.8838	0.0026
			PGTransformer	0.0030	0.0055	0.0022	0.9956	1.0000	0.9765	0.9234	0.0039
N2 H130	16M	Mesh	U-Net based [4], [5], [15]	0.0931	0.1223	0.1821	0.9596	0.6621	0.3841	0.3170	0.0013
			XGBoost based [8], [12]	0.0720	0.0972	0.1130	0.9765	0.7330	0.5417	0.3192	0.0149
			LSTM	0.0526	0.0554	0.0240	0.9923	0.9352	0.6801	0.4544	0.0045
			PGTransformer	0.0360	0.0517	0.0217	0.9962	0.9411	0.7738	0.5199	0.0082

TABLE III: QoR comparison between ours and manual PG structures adopted in real tapeout.

Circuit	Node	Period	CH	PD	UR	#Gates	#Nets	PG	V_{ir}	D_{mx}	D_{my}	WNS	TNS	Area	DRC	Opt time
design 1	N3E	1.111	143	3500	71	1.22	1.53	Manual	1.7	35.45	98.06	-0.029	-5.983	84339	20	$\geq$ 2400
								Ours	1.84	30.71	71.22	-0.014	-2.942	84425	53	30.42
design 2	N3E	0.714	143	4000	64	0.38	0.41	Manual	1.87	35.19	115.09	-0.016	-3.490	22648	127	$\geq$ 2400
								Ours	1.87	37.52	81.18	-0.005	-1.400	22486	132	33.33
design 3	N3E	1.429	143	5500	70	0.74	0.87	Manual	1.9	53.86	107.27	-0.002	-0.510	45760	186	$\geq$ 4800
								Ours	1.9	50.02	86.95	-0.002	-0.399	45466	102	62.38
design 4	N3E	0.307	195	3000	64	1.23	1.26	Manual	1.93	23.55	60.23	-0.006	-1.204	78741	152	$\geq$ 2400
								Ours	1.85	21.39	61.41	-0.004	-0.977	78928	158	13.72
design 5	N3E	0.290	195	4500	59	1.16	1.20	Manual	1.99	39.00	78.14	-0.014	-8.135	76129	199	$\geq$ 4800
								Ours	1.87	23.75	63.96	-0.013	-7.145	75318	128	60.78
design 6	N3E	0.307	195	6000	51	1.34	1.37	Manual	1.87	83.21	111.44	-0.014	-7.551	85177	337	$\geq$ 7200
								Ours	1.9	80.89	102.29	-0.009	-4.164	83998	256	92.78
design 7	N2	0.891	130	7500	65	0.35	0.42	Manual	1.64	32.57	137.79	-0.053	-19.526	21280	183	$\geq$ 7200
								Ours	1.68	31.95	77.28	-0.033	-15.924	21330	188	163.63
Avg reduction (%)									-	-9.5%	-20.6%	-33.5%	-32.4%	-0.5%	-	-

resources, cell proximity has better chances to be closer and thus enables *Innovus* to conduct easier timing-slack improvement by 34%. In addition, the resource advantage further reduces usages of large driving cells with 0.5% smaller cell area.

V. Conclusions

We have proposed a unified framework to predict SIR drop and optimize global PG structures, with novel sequence-based structure representation. Through MLHS and direct pattern modeling into PGTransformer, the prediction well captures multi-layer interactions. In industrial designs, our model outperforms state-of-the-art models by average 51% MAE reduction and 29% error improvement under small tolerance. In addition, generated structures from Pareto-frontier optimization achieves 15% smaller PG utilization and 34% timing-slack improvement on average under defined drop threshold.

References

[1] W.-H. Chang, L.-Y. Lin, Y.-G. Chen, and M. C.-T. Chao. Power Distribution Network Generation for Optimizing IR-Drop Aware Timing. In *Proc. of ICCAD*, 2020.

[2] J.-X. Chen, S.-T. Liu, Y.-T. Wu, M.-T. Wu, C.-M. Li, N. Chang, Y.-S. Li, and W.-T. Chuang. Vector-based Dynamic IR-drop Prediction using Machine Learning. In *Proc. of ASPDAC*, 2022.

[3] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, Q. V. Le. Symbolic discovery of optimization algorithms. In *arXiv preprint arXiv:2302.06675*, 2023.

[4] VA. Chhabria, Y. Zhang, H. Ren, B. Keller, B. Khailany, SS. Sapatnekar. MAVIREC: ML-Aided Vectored IR-Drop Estimation and Classification. In *Proc. of DATE*, 2021.

[5] VA. Chhabria, V. Ahuja, A. Prabhu, N. Patil, P. Jain, SS. Sapatnekar. Thermal and IR Drop Analysis using Convolutional Encoder-Decoder Networks. In *Proc. of ASPDAC*, 2021.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. In *Proc. of IEEE Transactions on Evolutionary Computation*, 2002.

[7] Y.-C. Fang, H.-Y. Lin, M.-Y. Sui, C.-M. Li, and J.-W. Fang. Machine-learning-based Dynamic IR Drop Prediction for ECO. In *Proc. of ICCAD*, 2018.

[8] C.-T. Ho and A. B. Kahng. IncPIRD: Fast Learning-based Prediction of Incremental IR Drop. In *Proc. of ICCAD*, 2019.

[9] S. Kundu, M. Prasad, S. Nishad, S. Nachireddy and H. K. MLIR: Machine Learning based IR Drop Prediction on ECO Revised Design for Faster Convergence. In *Proc. of VLSID*, 2022.

[10] A. B. Kahng, S. Kang, S. Kim, K. Samadi, and B. Xu. Power Delivery Pathfinding for Emerging Die-to-Wafer Integration Technology. In *Proc. of DATE*, 2019.

[11] S.-Y. Lin, Y.-C. Fang, Y.-C. Li, Y.-C. Liu, T.-S. Yang, S.-C. Lin, C.-M. Li, and J.-W. Fang. IR Drop Prediction of ECO-Revised Circuits using Machine Learning. In *Proc. of VTS*, 2018.

[12] C.-H. Pao, A.-Y. Su, and Y.-M. Lee. XGBIR: An XGBoost-based IR Drop Predictor for Power Delivery Network. In *Proc. of DATE*, 2020.

[13] L. N. Smith. Cyclical Learning Rates for Training Neural Networks. In *Proc. of WACV*, 2017.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In *Proc. of NIPS*, 2017.

[15] B.-Y. Wu, S.-Y. Fang, H.-W. Chang and P. Wei. SpeedER: A Supervised Encoder-Decoder Driven Engine for Effective Resistance Estimation of Power Delivery Networks. In *Proc. of MLCAD*, 2022.

[16] Y. Yamato, T. Yoneda, K. Hatayama, and M. Inoue. A Fast and Accurate Per-cell Dynamic IR-Drop Estimation Method for At-speed Scan Test Pattern Validation. In *Proc. of ITC*, 2012.

[17] W. Zhu and W. Liu. Attention-based Model for Dynamic IR Drop Prediction with Multi-view Features. In Electronics Letters, 2023.

Hardware Generation with High Flexibility using Reinforcement Learning Enhanced LLMs

Yifang Zhao*, Weimin Fu†, Shijie Li*, Yi-Xiang Hu*, Xiaolong Guo†✉, Yier Jin*
*University of Science and Technology of China, {zhaoyifang, shijie_li, yixianghu}@mail.ustc.edu.cn, jinyier@ustc.edu.cn
†Kansas State University, {weiminf, guoxiaolong}@ksu.edu

***Abstract*—The increasing complexity of integrated circuit design requires customizing Power, Performance, and Area (PPA) metrics according to different application demands. However, most engineers cannot anticipate requirements early in the design process, often discovering mismatches only after synthesis, necessitating iterative optimization or redesign. Some works have shown the promising capabilities of large language models (LLMs) in hardware design generation tasks, but they fail to tackle the PPA trade-off problem. In this work, we propose an LLM-based reinforcement learning framework, PPA-RTL, aiming to introduce LLMs as a cutting-edge automation tool by directly incorporating post-synthesis metrics PPA into the hardware design generation phase. We design PPA metrics as reward feedback to guide the model in producing designs aligned with specific optimization objectives across various scenarios. The experimental results demonstrate that PPA-RTL models, optimized for Power, Performance, Area, or their various combinations, significantly improve in achieving the desired trade-offs, making PPA-RTL applicable to a variety of application scenarios and project constraints.**

***Index Terms*—Hardware generation, PPA, Reinforcement learning.**

I. Introduction

With the broader application of Integrated circuits (ICs) in various scenarios, IC design requirements have become increasingly diverse [1]. Different applications demand tailored Power, Performance, and Area (PPA) metrics, often resulting in designs that fulfill similar functions but with vastly different optimization priorities. For instance, high-performance computing prioritizes performance over power efficiency, while IoT devices are susceptible to power consumption.

The diversity of design requirements presents challenges for hardware engineers. Hardware designs are described at the behavioral or Register Transfer Level (RTL), focusing on functional logic, while customized PPA optimization requires physical implementation with precise layout, routing, and cell library. Engineers begin with high-level hardware designs and then leverage Electronic Design Automation (EDA) tools for logic synthesis [2], verification [3], layout and routing [4], and timing analysis [5]. By defining appropriate constraints, fine-tuning tool settings, and adjusting outputs, they strive to meet the stringent power, performance, and area demands of modern ICs. However, these EDA tools are expertise-driven, raising the entry barriers for circuit design. Most engineers cannot anticipate requirements early in the design process, often discovering mismatches only after synthesis, necessitating iterative optimization or redesign. Thus, creating a PPA-oriented hardware design generation tool can greatly improve development efficiency.

With the advent of ChatGPT, its powerful natural language processing capabilities have brought new perspectives to many fields [6], making large language models (LLMs) a highly promising automation tool. In the field of hardware design generation, several studies [7]–[10] have shown that LLMs can autonomously produce hardware designs that are both syntactically and functionally accurate, offering convenience to hardware engineers. However, these models fail to consider the practical applicability of designs across different scenarios. To integrate post-synthesis metrics with the generation model, we aim to construct a hardware generation LLM with high flexibility.

LLM research technologies include prompting engineering, supervised fine-tuning (SFT), and Reinforcement Learning from Human Feedback (RLHF). Prompting methods rely on commercial general-purpose LLMs, struggling with the hallucination problem in general LLMs [11]. SFT typically enhances an LLM's performance by training the model to mimic behaviors on domain-specific datasets, but it falls short of meeting specific needs in specialized scenarios, such as generating hardware designs that align with PPA preferences tailored to specific application requirements. Moreover, SFT is highly data-intensive [12], requiring datasets of over $10,000$ examples to induce noticeable shifts in model behavior. Constructing such large-scale datasets for a specific scenario is nearly impossible, making it challenging to achieve tailored optimization using SFT alone. RLHF optimizes model behavior by incorporating human feedback and reinforcement learning (RL), which helps mitigate hallucinations [13] and reduces the need for large-scale datasets, typically requiring only a few thousand samples focused on quality rather than quantity.

In this work, we leverage RLHF-based LLM technology to generate hardware designs that better suit the diverse needs of real-world applications. We design PPA metrics as rewards, enabling the model to iteratively adjust its parameters based on feedback. Through continuous updates to its policy, the model progressively optimizes outputs to align with specific PPA preferences. However, this process faces two challenges: **(C1)** The complex trade-off between power, performance, and area typically demands sophisticated strategies and human intervention, while RL needs to formally represent these trade-offs in the reward feedback to meet different PPA objectives. **(C2)** Performing PPA calculations in every iteration would significantly slow down the model training process, as generating PPA evaluation feedback requires logic synthesis and simulation, which are computationally intensive steps that consume significant time and resources. One solution is to use machine learning (ML)–based models to approximate PPA metrics, which address real-time concerns but may lack precision. Another approach separates PPA calculations from model training by creating an offline PPA metric dataset, which balances computational demands with real-time requirements.

To overcome these challenges, we propose the PPA-RTL framework, which constructs an offline dataset of *Function Description*⇔*Reference Code* pairs with PPA metrics and introduces a PPA weighted formula to represent optimization rewards across diverse scenarios. Integrating PPA into the hardware generation process through RL allows models to more effectively navigate the complex trade-offs in hardware design, such as balancing chip performance and power consumption or optimizing the layout to reduce area without compromising functionality.

The main contributions are outlined as follows:

1) For the first time, we optimize key post-synthesis metrics, PPA, in

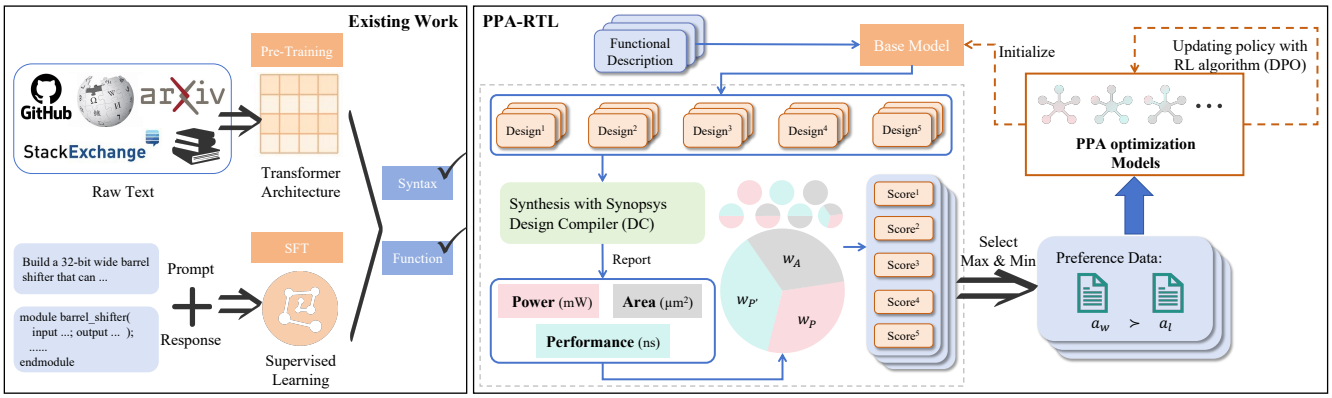


Fig. 1: PPA-RTL Framework. Existing hardware code generation LLMs primarily focus on two training methods (Pre-Training and SFT) to acquire syntactic and functional correct output. The PPA-RTL framework focuses on post-synthesis PPA metrics to enhance hardware generation quality. It first extracts PPA values from the Power, Performance, and Area synthesis report generated by RTL synthesis tools for each design. Adjusting the weights of PPA generates various preference datasets that adapt to different PPA preferences, thereby obtaining hardware code generation LLMs guided by different PPA preferences.

the context of hardware code generation using LLMs. We develop an RL-based hardware generation framework, named **PPA-RTL**, that directly integrates PPA considerations into the design process, allowing users to prioritize one or more of these metrics based on specific project requirements.

2) We configure various optimization objectives to reflect the common trade-offs in real-world hardware designs, such as Power-only, Performance-only, Area-only, and their combinations (e.g., Power & Performance, Power & Area, Performance & Area, and Power & Performance & Area). The experimental results demonstrate that PPA-RTL models significantly enhance the ability to achieve desired trade-offs, making PPA-RTL adaptable to a range of application scenarios and project constraints. Compared to the SFT-only model, the SFT-RL-PPA model improves Power by an average of 20.97%, Performance by 14.68%, and Area by 29.05%.

II. Background and Related Work

A. LLMs for Hardware Code Generation

As early as 2020, the hardware field began experimenting with fine-tuned GPT-2 models for generating hardware designs from natural language. DAVE [14] demonstrated the potential of language models in automated hardware design. However, DAVE has notable limitations: difficulties with complex, multi-task instructions, limited generalization to tasks outside its training templates, and a restriction on output length (1024 tokens).

After OpenAI released ChatGPT in 2022, it quickly attracted many researchers, sparking studies on the hardware domain, primarily focusing on two research methods: Prompt-based and SFT. Prompt-based methods [10], [15]–[17] guide advanced general-purpose LLMs to perform tasks within the hardware domain by providing domain-specific knowledge as prompts, relying on powerful commercial models like GPT-4 [6] and Claude [18]. However, the inability to modify these closed-source commercial models makes it difficult to address hallucination issues, particularly in the hardware domain [19]. SFT methods [8], [9], [20]–[22] train LLMs on hardware-specific datasets consisting of functional descriptions and corresponding hardware code to achieve higher syntactic and functional accuracy. However, generating codes is only the first step in the IC design process, which must also undergo logical synthesis, layout and routing, timing analysis, and physical verification. In these stages, the post-synthesis metrics PPA directly impact the chip's efficiency, speed, and cost. While SFT-based models can generate syntactically correct hardware designs, they do not account for post-synthesis metrics, limiting their practical applicability.

B. Reinforcement Learning from Human Feedback

In standard RLHF workflow, an LLM is modeled as a policy π, which takes a prompt $x \in \mathcal{X}$ and generates a response $a \in \mathcal{A}$ from the distribution $\pi(\cdot|x)$. The initial model, π_0, is typically fine-tuned on instruction-following data after pre-training [23]. The central idea of RLHF is to train the model using *relative feedback*, rather than relying on an absolute reward signal commonly used in traditional RL. It addresses the challenge that human evaluators often struggle to provide consistent absolute ratings but tend to be more reliable when tasked with comparing two responses and selecting the preferred one [24]. To formalize preference feedback, we introduce the concept of *Preference Oracle*.

Definition 1 (Preference Oracle). A preference oracle $\mathbb{P} : \mathcal{X} \times \mathcal{A} \times \mathcal{A} \to [0, 1]$ is a function that, when queried with a prompt x and two candidate responses a^1 and a^2, returns a preference signal. Specifically, we can sample a binary outcome y from the Bernoulli distribution $\text{Ber}(t)$: $y \sim \text{Ber}\left(\mathbb{P}\left(a^1 \succ a^2 | x, a^1, a^2\right)\right)$, if $y = 1$, it indicates that a^1 is preferred over a^2; if $y = 0$, it indicates that a^2 is preferred.

To simplify the learning process, it is often assumed that human preferences can be modeled using the Bradley-Terry (BT) model, a widely-used framework in preference learning [25].

Definition 2 (Bradley-Terry Model). Assume the existence of a ground-truth reward function r^*. Given two options a^1 and a^2, the preference probability is calculated using a sigmoid function, which reflects the difference in reward values between the two options. Specifically, the probability of preferring a^1 over a^2 given prompt x is modeled as: $\mathbb{P}(a^1 \succ a^2 | x, a^1, a^2) = \sigma(r^*(x, a^1) - r^*(x, a^2))$, where $\sigma(z) = \frac{1}{1+\exp(-z)}$ is the sigmoid function, which maps the difference in reward values to a preference probability.

Although the BT model may not fully capture the complex human preference, it provides a reasonable approximation and has proven to play a critical role in the development of ChatGPT [6] and Claude [18] by combining RLHF with reward maximization.

III. Methodology

We propose the **PPA-RTL** framework, which leverages RL to incorporate feedback from post-synthesis PPA metrics, enabling LLMs to generate hardware designs adaptable to various optimization objectives. PPA-RTL addresses the limitations of existing hardware code generation LLMs, which primarily focus on syntactic and functional correctness. It aims to improve the quality of hardware generation and ensure that the generated designs meet specific physical scenarios.

A. Motivation

LLMs are initially pre-trained on extensive universal datasets, equipping them with understanding ability. Then, the pre-trained LLM was fine-tuned on a specific task, which is IC design here, to become a domain expert. However, achieving optimal performance in real-world IC design involves navigating complex trade-offs among PPA. Decisions, such as prioritizing power efficiency over performance or balancing area constraints with power demands, require sophisticated decision-making, which labeled data cannot fully address. Therefore, SFT, which relies on labeled data, can not handle the flexible optimization to achieve PPA requirements. This leads to our first challenge: **(C1)** *How can we design a flexible framework that accommodates different PPA requirements?*

A straightforward approach for guiding LLMs in meeting diverse PPA requirements is to align models with preference via RL. Unfortunately, real-time PPA computing contradicts the overhead of LLM training, which raises our second challenge: **(C2)** *How can we avoid real-time evaluation to reduce the time delay during LLM training?*

B. PPA-RTL Framework

To construct PPA-oriented flexible hardware generation LLMs, we propose the PPA-RTL framework, which incorporates PPA metrics into the hardware code generation stage to target different application optimization objectives, as shown in Fig. 1. We address two challenges as follows.

Challenges 1 (C1) Different applications require tailored optimization strategies to balance power, performance, and area, while a single optimization strategy cannot meet the requirements of various applications. PPA-RTL needs to formally represent these trade-offs in the reward feedback to effectively guide model parameter updates.

To address **(C1)**, PPA-RTL formulates the hardware design generation as an RLHF workflow, denoted by $\langle \pi, \mathcal{X}, \mathcal{A}, \mathbb{P} \rangle$. **Policy** π represents a hardware design generation model, with the base model denoted as π_{ref}. **Prompt set** $\mathcal{X}$**:** Each prompt $x \in \mathcal{X}$ represents a functional description. **Response set** $\mathcal{A}$**:** The response $a \in \mathcal{A}$ is generated design from the distribution $\pi(\cdot|x)$. **Preference model** $\mathbb{P}\left(a^1 \succ a^2 | x, a^1, a^2\right)$ provides a relative preference signal for multiple generated designs under a specified PPA optimization strategy. In PPA-RTL, we sample k designs from the base model for each functional description x in the set $\mathcal{X}$. RTL synthesis tool then performs logic synthesis and simulation, resulting in reports about power, timing, and area. By specifying an optimization strategy, comparing PPA values, and selecting better & worse designs, the model's policy is updated using an RL algorithm to align outputs more closely with the desired PPA strategy.

To accommodate various applications, we implement a flexible approach that adjusts the weights $w_P, w_{P'}$, and w_A for each objective to simulate the priority of PPA metrics across different applications. The preference dataset is then created by selecting better or worse designs based on the scoring function $s(x, a)$.

$$s(x,a) = -\left(w_P \times v'_P + w_{P'} \times v'_{P'} + w_A \times v'_A\right)$$

where v'_P (resp. $v'_{P'}$, v'_A) represents the normalized value of power (resp. performance, area) extracted from the post-synthesis reports. The non-negative hyperparameters $w_P, w_{P'}$, and w_A can be adjusted to satisfy different PPA requirements. The negative sign indicates that the objective is to minimize these metrics: lower power consumption, higher frequency (lower critical path delay), and smaller area.

Challenge 2 (C2) Acquiring PPA values of design requires using time-consuming and computationally intensive EDA tools for synthesis, simulation, and analysis. Each update requires performing these steps, resulting in the inability to obtain real-time feedback during model training. To bridge hardware design synthesis and model training, approximate evaluation and pre-generated offline PPA datasets provide promising methods. For the approximate method, training an ML-based approximation model to predict PPA values can quickly provide PPA estimates, but it has limitations in accuracy and reliability. The cumulative error of the approximate model combined with the randomness in LLM generation creates a dual source of error, which may cause the model to gradually deviate from the optimal solution during training, leading to designs that do not meet expectations. For offline PPA datasets, using pre-generated data directly during training eliminates the intermediate steps of generating PPA values each time, simplifying the training process. The offline approach also allows for more accurate model optimization, as the dataset is already precisely calculated through EDA tools, avoiding the prediction errors associated with approximate models.

To address **(C2)**, we construct an offline dataset with PPA values for the training process by employing an EDA tool to map hardware designs into gate-level netlists. The synthesis process generates three key reports—power, timing, and area—which collectively capture the physical and performance characteristics of the design.

1) ***Power Estimation****:* Power estimation includes dynamic power from switching activity and static power dissipated by transistors without activity, providing an essential view into the design's energy efficiency, expressed in milliwatts (mW).

2) ***Performance Estimation****:* Performance is measured by the number of operations executed per unit of time. Increasing the frequency reduces the clock period, allowing the design to perform more operations within the same time frame, thereby improving overall system performance. The maximum frequency (f_{max}) is inversely related to the critical path delay, as expressed by the formula: $f_{\text{max}} = 1/critical_path_delay$. Therefore, performance can be quantified by the critical path delay (measured in nanoseconds, ns): the smaller the critical path delay, the higher the frequency, and thus, the better the overall performance.

3) ***Area Estimation****:* The total cell area is the physical area required by the synthesized design, including contributions from combinational logic, sequential elements, buffers/inverters, and other components. It reflects the cumulative area of all logic cells mapped from the technology library, measured in square micrometers (μm^2), ensuring the design fits within layout constraints.

C. PPA-RTL Training

In Algorithm 1, training the PPA-RTL model begins with the preference dataset generation, followed by the training phase. We leverages the Direct Preference Optimization (DPO) [26] to update the policy π_θ.

In the preference dataset generation stage (lines 2-13), for each functional description $x \in \mathcal{X}$, the reference model π_{ref} generates k candidate code samples. The samples that pass the synthesis check are collected in the candidate code set A. From set A, the best and worst samples, denoted as a_w and a_l respectively, are identified based

TABLE I: **Optimization objectives setting:** seven common trade-offs in real-world hardware design. **Preference dataset generation:** the valid subsets generated by $3,000$ function descriptions on the base model, excluding unsynthesizable and non-differentiated outputs. **Model training:** loss and accuracy metrics during model training based on different training strategies (RL-only and SFT-RL).

Optimization objectives	Weights			Preference dataset size ($\|\mathcal{D}\|$)	RL-only		SFT-RL	
	w_P	$w_{P'}$	w_A		Accuracy	Loss	Accuracy	Loss
Power-Only (Power)	1	0	0	1966	0.8375	0.5352	1.0	0.0322
Performance-Only (Perf)	0	1	0	1827	0.8375	0.5356	1.0	0.0540
Area-Only (Area)	0	0	1	1936	0.8250	0.6148	0.9875	0.0408
Power and Performance (PP)	1/2	1/2	0	1808	0.7625	0.5595	1.0	0.0610
Power and Area (PowerA)	1/2	0	1/2	1927	0.8625	0.5454	1.0	0.0410
Performance and Area (PerfA)	0	1/2	1/2	1813	0.7875	0.5750	1.0	0.0683
Power, Performance, Area (PPA)	1/3	1/3	1/3	1892	0.7875	0.5444	1.0	0.0360

Algorithm 1 PPA-RTL Training

Input: Functional descriptions $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, base model π_{ref}, hyperparameters w_P, $w_{P'}$,w_A, sample times k, training iterations T.

Output: Trained hardware generation model π^*_θ.

1: $\mathcal{D} \leftarrow \emptyset$; $\pi_\theta \leftarrow \pi_{ref}$ ▷ *Initialize*
2: **for** each $x \in \mathcal{X}$ **do** ▷ *Preference dataset generation*
3: $A \leftarrow \emptyset$ ▷ *Candidate code set*
4: **for** $i \leftarrow$1 to k **do**
5: $a_i \leftarrow \pi_{ref}(x)$
6: **if** a_i passes synthesis check **then**
7: $A \leftarrow A \cup \{a_i\}$; Record $v_P(a_i)$,$v_{P'}(a_i)$,$v_A(a_i)$
8: Obtain $v'_P(a_i)$,$v'_{P'}(a_i)$,$v'_A(a_i)$ with min-max normalization
9: **for** $a_i \in A$ **do**
10: $s(x, a_i) \leftarrow -(w_P \times v'_P(a_i) + w_{P'} \times v'_{P'}(a_i) + w_A \times v'_A(a_i))$
11: $a_w \leftarrow \arg\max_{a_i \in A} s(x, a_i)$; $a_l \leftarrow \arg\min_{a_i \in A} s(x, a_i)$
12: **if** $a_w \neq a_l$ **then**
13: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, a_w, a_l)\}$
14: **for** $t \leftarrow$1 to T **do** ▷ *Training*
15: **for** each $(x, a_w, a_l) \in D$ **do**
16: $\mathcal{L}_t(\pi_\theta) \leftarrow -\log \sigma \left(\beta \log \frac{\pi_\theta(a_w|x)}{\pi_{\text{ref}}(a_w|x)} - \beta \log \frac{\pi_\theta(a_l|x)}{\pi_{\text{ref}}(a_l|x)}\right)$
17: Update π_θ via gradient descent on $\mathcal{L}_t(\pi_\theta)$
18: $\pi^*_\theta \leftarrow \pi_\theta$
19: **return** π^*_θ

on the scoring function $s(x, a_i)$ and stored in the preference dataset $\mathcal{D}$ in the form of preference tuples (x, a_w, a_l).

In the training phase (lines 14-17), the model π_θ undergoes T iterations of optimization. The implicit reward function $r^*_\theta(x, a) = \beta \log \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)}$ represents the preference of the current model relative to the reference model for a given prompt-response pair. In each iteration, the loss function $\mathcal{L}_t(\pi_\theta)$ optimizes the model by maximizing the likelihood of preferred data a_w and minimizing the likelihood of non-preferred data a_l through the difference $r^*(x, a_w) - r^*(x, a_l)$ Gradient descent is used to update the model parameters iteratively. The resulting model π^*_θ can generate optimized hardware designs that align with learned preferences.

IV. Experiments

In this section, we validate the integration of RL with PPA metrics to enhance the capability of hardware code generation LLMs in meeting diverse real-world application physical demands.

A. Experimental Setup

PPA-RTL is implemented on a Linux machine with eight A100 80GB GPUs. We consider two code generation models as baselines, Deepseek-coder [27] and RTLCoder [21]. Deepseek-coder is an open-source code generation model closest to GPT-4-Turbo. RTLCoder is an open-source hardware code LLM based on Deepseek-coder with supervised fine-tuning. We use two versions of these models, Deepseek-coder-6.7b (Original) and RTLCoder-DeepSeek-6.7b (SFT-only), thereby exploring two distinct training strategies: 1) **RL-only**: Direct reinforcement training using the Deepseek-coder-6.7b model (Original). 2) **SFT-RL**: Reinforcement training based on supervised fine-tuning RTLCoder of the original model. Additionally, to demonstrate the applicability of our PPA-based RL approach to various PPA scenarios, Table I designs seven optimization objectives that reflect the common trade-offs of real-world hardware design.

B. Dataset Generation

To acquire PPA-oriented preference datasets, we leverage Synopsys Design Compiler (DC) [28] with the TSMC 90nm CMOS technology library to synthesize designs and adjust the weights of PPA based on different optimization objectives in Table I. Specifically, we sampled $3,000$ (*Function Description⇔Reference Code*) pairs from the RTLCoder dataset [29]. Function descriptions were inputs for two base models, Deepseek and RTLCoder, to generate multiple hardware designs. For each function description, we construct five hardware designs, including generated and reference code, and synthesize these designs. Samples that failed to generate synthesizable designs were filtered out. By analyzing the resulting Power, Timing, and Area reports, we extracted power (mW), critical path delay (ns), and area (μm^2). Based on different optimization objectives, calculate scores for each design, filter out samples with no score differences, and then select the maximum and minimum values to construct seven distinct chosen-rejected preference datasets.

C. Model Training

We employ DPO to train PPA-RTL hardware generation LLMs. DPO simplifies the traditional complexities of RLHF by leveraging the best and worst selected responses to directly optimize the model's predictive capabilities. Table I shows the training loss and reward accuracy for models trained with two strategies: RL-only and SFT-RL. $Loss$ quantifies the discrepancy between the model's predictions and the preferred outcomes, with lower values indicating closer alignment with preferences. $Accuracy$ shows the proportion of predictions that match the preferred outcomes, with higher values indicating better alignment with the target preferences. The SFT-RL exhibits lower loss and higher accuracy across all optimization objectives, indicating that SFT-based RL leads to more stable performance compared to RL-only.

D. Experimental Evaluation

To evaluate the effectiveness of PPA-RTL models in improving various optimization objectives, we utilize the RTLLM-v1.1 benchmark [30] to generate designs for each example using the advanced

TABLE II: PPA evaluation results. The **Original** refers to the DeepSeek-Coder-6.7B model, while the **SFT-only** is a supervised fine-tuned model based on DeepSeek-Coder-6.7B specifically for hardware code generation. **Metrics Power**, **Perf**, and **A** represent power (mW), performance (ns), and area (μm^2), respectively. Each row corresponds to a hardware design from the RTLLM benchmark. **Green** squares show better metric than the original model, while **red** squares show worse metric. The SFT-RL models are significantly better than RL-only models in PPA improvement compared to the base model.

Model	Orignal			vs Orignal																						
				GPT-4o			RL-Power			RL-Perf			RL-Area			RL-PP			RL-PowerA			RL-PerfA			RL-PPA		
Metric	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A
accu	0.055	1250.888	0.169	36.59%	-0.85%	46.81%	10.39%	-0.22%	7.04%	36.84%	11.14%	-0.06%	-0.38%	-0.82%	3.49%	2.59%	-0.17%	-0.20%	4.76%	5.21%	1.61%	0.88%	25.95%	-0.33%	4.13%	1.26%	44.78%
adder_16bit	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
adder_32bit	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
adder_8bit	0.002	16.934	0.240	-0.60%	-14.09%	-0.23%	-0.12%	-2.03%	-0.06%	-0.25%	0.09%	-0.09%	0.00%	-1.87%	0.00%	-0.12%	0.00%	-0.06%	-0.10%	-1.92%	-0.01%	-0.10%	-1.83%	-0.05%	3.17%	-1.87%	-0.05%
adder_pipe_64bit	0.141	2773.478	0.240	-12.80%	0.00%	-7.14%	0.00%	-2.51%	29.34%	-49.78%	5.21%	-100.00%	-2.05%	-3.80%	44.36%	21.76%	15.28%	68.85%	9.56%	-0.11%	7.78%	-14.23%	14.58%	55.83%	-2.27%	15.42%	42.66%
alu	0.672	6504.926	6.737	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	51.23%	96.29%	60.62%	30.18%	61.23%	24.40%	52.56%	85.57%	55.77%	N/A	N/A	N/A	51.23%	96.29%	60.62%
asyn_fifo	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%	0.00%	N/A	N/A	N/A	0.00%	0.00%	0.00%	N/A	N/A	N/A	0.00%	0.00%	0.00%
calendar	0.015	496.742	0.244	-0.02%	-0.37%	-0.05%	10.73%	-0.16%	-0.03%	-0.01%	2.13%	-0.03%	-0.04%	-0.39%	12.03%	-0.01%	5.16%	-0.02%	3.27%	-0.39%	12.03%	-0.01%	15.62%	-0.03%	-0.02%	7.38%	15.27%
counter_12	0.004	98.078	0.265	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
div_16bit	0.164	1414.728	5.695	-3.29%	-100.00%	-2.03%	97.59%	95.39%	93.12%	97.59%	95.39%	93.12%	97.59%	95.39%	93.12%	97.59%	95.39%	93.12%	97.59%	95.39%	93.12%	-6.01%	13.20%	34.09%	97.59%	95.39%	93.12%
edge_detect	0.003	53.484	0.228	0.00%	-0.13%	0.00%	3.60%	-0.06%	-0.01%	0.00%	2.63%	-0.01%	0.00%	0.00%	1.58%	0.51%	1.75%	1.32%	2.98%	-0.13%	0.00%	-0.01%	10.53%	9.20%	-0.01%	5.26%	1.58%
freq_div	0.023	765.012	0.252	44.17%	0.79%	55.82%	79.03%	0.79%	87.59%	54.78%	0.26%	65.06%	72.75%	0.79%	82.14%	75.33%	0.79%	83.60%	66.73%	0.79%	77.07%	75.53%	0.79%	84.89%	74.80%	0.79%	82.86%
fsm	0.005	100.195	0.300	N/A	N/A	N/A	0.00%	20.00%	-0.01%	-0.02%	18.67%	-0.03%	-0.03%	20.00%	0.54%	42.61%	19.33%	-0.03%	-0.01%	18.00%	4.61%	-0.03%	18.89%	2.82%	53.65%	18.67%	2.92%
JC_counter	0.031	706.541	0.247	N/A	N/A	N/A	26.23%	-0.04%	-0.57%	-0.43%	38.95%	-0.57%	-0.42%	-0.04%	15.58%	25.35%	-0.04%	-0.57%	0.74%	-0.01%	-0.14%	-0.42%	-0.04%	1.45%	5.65%	15.09%	-0.57%
multi_16bit	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
multi_booth_8bit	0.070	1281.793	0.259	-2.33%	-0.22%	-0.99%	0.25%	-0.11%	-1.78%	0.25%	25.62%	2.87%	0.05%	8.29%	2.87%	12.37%	25.33%	22.13%	10.41%	12.87%	11.51%	-0.35%	12.96%	13.85%	5.01%	23.78%	3.78%
multi_pipe_4bit	0.055	887.645	0.261	16.11%	7.87%	-0.04%	-1.01%	-0.22%	-0.87%	N/A	N/A	N/A	N/A	N/A	N/A	24.64%	37.50%	15.09%	N/A	N/A	N/A	-0.92%	3.74%	-0.24%	5.48%	7.87%	18.43%
multi_pipe_8bit	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%	0.00%	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
parallel2serial	0.008	211.398	0.241	22.83%	0.50%	20.23%	6.10%	-0.11%	-0.46%	6.10%	0.50%	2.20%	16.43%	-0.02%	11.22%	24.20%	12.00%	-0.25%	39.63%	0.50%	39.84%	16.39%	0.50%	21.90%	16.43%	0.50%	11.22%
pe	2.878	11053.224	0.271	0.00%	0.00%	0.00%	0.38%	0.00%	0.00%	0.00%	0.00%	0.00%	0.38%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
pulse_detect	0.003	65.903	0.246	N/A	N/A	N/A	9.57%	0.00%	4.07%	-0.01%	18.70%	-0.01%	-20.51%	-0.04%	4.07%	6.37%	0.00%	-0.01%	3.23%	0.00%	3.54%	0.00%	0.81%	1.75%	4.18%	0.81%	6.82%
radix2_div	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
RAM	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	0.00%	0.00%	0.00%
right_shifter	0.006	107.251	0.189	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
serial2parallel	0.013	356.469	0.223	20.39%	-0.36%	25.38%	22.52%	-0.22%	28.11%	14.72%	1.80%	18.33%	12.12%	-0.36%	15.12%	13.92%	-0.05%	17.22%	14.33%	-0.37%	18.29%	15.03%	-0.36%	18.61%	21.84%	-0.23%	27.87%
signal_generator	0.006	203.777	0.287	-0.01%	0.42%	-0.02%	4.79%	1.11%	4.09%	3.02%	1.11%	-0.02%	0.68%	1.67%	1.38%	4.05%	2.37%	-0.01%	1.03%	1.25%	1.58%	0.00%	1.53%	0.48%	2.67%	20.61%	1.18%
synchronizer	0.010	196.157	0.250	-0.05%	0.00%	-0.05%	0.00%	0.00%	0.00%	-0.03%	5.60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	9.65%	-0.07%	21.71%	0.00%	0.00%	0.00%	0.00%	0.00%	0.79%
traffic_light	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
width_8to16	0.030	615.812	0.250	0.00%	0.00%	5.93%	11.40%	0.00%	8.56%	7.13%	1.33%	10.21%	11.11%	0.00%	8.79%	0.00%	0.00%	5.36%	27.69%	-0.04%	20.18%	11.40%	0.00%	8.56%	10.97%	0.00%	12.37%
Average Improvement Rates				5.26%	-4.63%	6.24%	10.82%	4.29%	9.93%	7.72%	10.42%	4.13%	8.85%	7.97%	13.22%	13.62%	9.85%	11.78%	12.74%	8.02%	13.65%	3.89%	4.68%	10.11%	12.66%	10.97%	15.20%

Model	SFT-only			vs Orignal			vs SFT-only																			
				SFT-only			SFT-RL-Power			SFT-RL-Perf			SFT-RL-Area			SFT-RL-PP			SFT-RL-PowerA			SFT-RL-PerfA			SFT-RL-PPA		
Metric	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A	Power	Perf	A
accu	0.035	738.058	0.250	36.18%	-0.85%	41.00%	65.83%	-0.07%	72.47%	-0.89%	21.40%	-2.34%	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
adder_16bit	0.013	174.989	0.074	0.00%	0.00%	0.00%	31.37%	-4.59%	52.69%	-0.98%	-10.03%	-1.02%	-0.06%	-5.73%	30.11%	-0.73%	-6.59%	-0.79%	57.72%	-15.69%	54.78%	0.65%	33.78%	46.24%	30.44%	4.05%	66.26%
adder_32bit	0.083	550.368	4.761	0.00%	0.00%	0.00%	93.59%	98.36%	86.67%	N/A	N/A	N/A	N/A	N/A	N/A	56.71%	59.62%	18.65%	57.91%	58.47%	60.83%	72.47%	78.16%	63.26%	57.36%	62.63%	45.59%
adder_8bit	0.017	135.475	1.147	-0.51%	-9.59%	-0.23%	-0.08%	-4.44%	0.00%	-1.19%	65.12%	-0.19%	28.25%	83.65%	90.23%	79.85%	12.60%	-0.63%	47.75%	45.64%	46.09%	13.45%	27.79%	7.40%	22.27%	37.25%	17.50%
adder_pipe_64bit	0.831	20015.049	0.240	-25.20%	0.00%	-35.49%	74.96%	-100.00%	93.13%	-4.31%	45.83%	93.13%	97.50%	86.25%	98.78%	64.67%	-0.93%	92.74%	60.02%	-13.52%	95.28%	5.69%	0.00%	99.92%	51.05%	-1.91%	96.91%
alu	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
asyn_fifo	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%	0.00%	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%	0.00%	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
calendar	0.016	536.962	0.281	-0.04%	-0.39%	-0.08%	7.64%	13.52%	8.15%	0.28%	64.41%	0.39%	4.89%	10.32%	5.75%	7.42%	13.52%	7.84%	5.25%	9.11%	7.99%	24.59%	7.20%	28.69%	6.38%	13.17%	7.49%
counter_12	0.004	98.078	0.265	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-0.40%	9.43%	0.00%	-0.40%	-0.10%	5.06%	13.81%	8.30%	-3.63%	68.29%	6.94%	4.88%	-0.02%	5.66%	1.08%	-0.25%	14.86%	10.79%
div_16bit	0.004	97.373	0.263	97.59%	95.39%	93.12%	0.00%	0.00%	0.00%	-0.40%	38.10%	-0.08%	-0.40%	-0.12%	4.40%	19.55%	7.43%	-3.64%	0.00%	0.00%	0.00%	-0.37%	7.43%	-3.64%	0.00%	0.00%	0.00%
edge_detect	0.003	53.626	0.240	0.00%	-0.13%	0.00%	2.75%	-0.21%	-0.40%	-0.06%	58.33%	-0.40%	23.68%	0.00%	35.53%	8.47%	-0.03%	-4.02%	-0.02%	-0.21%	11.66%	0.00%	0.00%	0.00%	-0.02%	-0.21%	-0.40%
freq_div	0.014	364.442	0.233	41.46%	7.41%	52.36%	70.82%	-0.28%	-1.23%	17.56%	57.14%	27.59%	32.24%	-0.10%	46.76%	53.99%	8.29%	-1.23%	50.07%	-0.27%	51.44%	-0.04%	1.43%	29.77%	53.13%	-0.16%	58.76%
fsm	0.004	80.791	0.248	10.17%	17.50%	19.37%	54.01%	-0.02%	68.56%	-0.02%	23.23%	17.03%	54.85%	3.03%	61.57%	0.00%	3.03%	-0.01%	-0.47%	-0.12%	-3.75%	-0.36%	1.82%	-3.78%	28.17%	3.03%	30.13%
JC_counter	0.043	995.602	0.251	-0.43%	-0.04%	-0.57%	98.27%	4.38%	98.44%	0.96%	0.27%	0.00%	0.27%	0.40%	0.07%	0.47%	0.30%	0.09%	0.28%	0.40%	0.07%	1.30%	0.20%	-0.01%	0.00%	0.00%	0.00%
multi_16bit	0.043	995.602	0.251	0.00%	0.00%	0.00%	N/A	N/A	N/A	96.19%	16.33%	98.30%	-0.20%	4.38%	96.88%	1.85%	2.05%	0.00%	0.28%	0.40%	0.07%	1.26%	0.00%	0.42%	0.28%	0.40%	0.07%
multi_booth_8bit	0.089	1446.762	0.269	-0.69%	-0.11%	-0.33%	74.39%	0.19%	66.96%	41.37%	13.20%	27.62%	30.27%	-0.17%	25.62%	37.39%	5.35%	30.16%	47.29%	3.62%	31.05%	37.39%	6.69%	30.16%	47.29%	3.62%	31.05%
multi_pipe_4bit	0.093	1438.718	0.281	-1.34%	-0.22%	-1.09%	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
multi_pipe_8bit	0.093	1438.718	0.281	0.00%	0.00%	0.00%	N/A	N/A	N/A	46.64%	23.20%	33.08%	-0.20%	14.59%	97.84%	95.59%	14.59%	-1.29%	99.83%	75.09%	99.75%	40.58%	-0.16%	-1.29%	99.83%	75.09%	99.75%
parallel2serial	0.006	159.466	0.240	25.03%	0.50%	24.57%	1.04%	-0.05%	-0.06%	-1.49%	10.08%	-7.37%	-2.53%	-1.06%	45.72%	34.45%	0.00%	37.17%	83.01%	38.33%	86.55%	-0.15%	6.29%	37.17%	61.04%	38.33%	86.55%
pe	2.878	11053.224	0.271	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-42.51%	14.76%	0.20%	-33.33%	0.00%	0.00%	0.00%	0.00%	0.00%	-100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-41.67%	0.00%	0.00%
pulse_detect	0.003	66.326	0.245	7.52%	0.41%	0.00%	-0.01%	-0.05%	2.13%	-0.03%	0.68%	-0.04%	-0.05%	0.00%	-0.21%	14.21%	2.04%	28.99%	2.74%	-0.25%	19.75%	-0.05%	25.67%	-0.17%	16.05%	-0.55%	30.59%
radix2_div	0.035	622.574	0.186	0.00%	0.00%	0.00%	90.09%	-0.61%	-1.36%	-0.18%	2.16%	-1.44%	-5.43%	-0.51%	96.37%	-5.43%	-18.84%	24.63%	7.55%	-0.57%	28.74%	96.60%	3.24%	97.17%	-0.60%	-0.57%	28.74%
RAM	0.002	16.934	0.210	0.00%	0.00%	0.00%	-0.06%	-0.38%	-0.43%	-1.64%	0.00%	-8.57%	0.00%	-0.29%	0.00%	N/A	N/A	N/A	-0.07%	-0.29%	36.77%	-2.07%	4.76%	20.16%	2.42%	3.17%	8.87%
right_shifter	0.006	107.251	0.189	0.00%	0.00%	0.00%	43.46%	-0.58%	0.00%	-1.49%	39.50%	0.00%	0.00%	-0.12%	21.05%	79.80%	87.70%	0.00%	37.18%	-0.49%	15.43%	-1.10%	26.99%	0.00%	-0.39%	34.30%	15.43%
serial2parallel	0.013	317.661	0.238	0.07%	-0.16%	10.89%	13.53%	-0.12%	4.49%	-0.25%	26.22%	-0.65%	-0.28%	20.54%	66.24%	52.38%	0.00%	-0.64%	25.39%	-5.14%	4.34%	-0.11%	1.20%	8.78%	8.92%	36.86%	70.01%
signal_generator	0.006	201.096	0.284	6.33%	1.11%	1.32%	-0.19%	11.97%	-0.39%	-0.01%	2.11%	-0.04%	-0.01%	33.54%	46.67%	-0.01%	2.25%	6.18%	11.07%	-6.01%	1.94%	-1.08%	27.78%	6.18%	80.72%	36.97%	3.13%
synchronizer	0.010	196.157	0.250	0.00%	0.00%	0.00%	5.18%	0.00%	-0.02%	0.00%	0.00%	0.00%	0.00%	0.00%	45.32%	5.18%	0.00%	6.33%	0.00%	0.00%	0.00%	12.13%	7.20%	14.32%	0.00%	0.00%	0.00%
traffic_light	0.010	196.157	0.250	0.00%	0.00%	0.00%	-0.04%	0.00%	0.00%	-0.56%	2.60%	-3.76%	20.98%	4.00%	26.98%	80.55%	2.81%	2.91%	0.00%	0.00%	0.00%	0.00%	0.00%	7.91%	0.00%	0.00%	0.00%
width_8to16	0.030	579.062	0.250	1.01%	0.00%	5.97%	88.01%	1.33%	89.76%	11.30%	0.71%	26.94%	45.41%	0.44%	33.59%	35.75%	2.81%	53.98%	61.49%	0.00%	66.13%	15.91%	1.78%	35.35%	1.78%	6.67%	19.05%
Average Improvement Rates				7.30%	4.11%	7.81%	32.58%	0.73%	25.58%	6.31%	20.99%	11.94%	12.31%	10.54%	40.85%	29.44%	8.25%	11.75%	24.90%	7.82%	28.79%	12.67%	11.00%	21.00%	20.97%	14.68%	29.05%

general-purpose language model GPT-4o [31] and 14 models trained through RL-only and SFT-RL methods. These generated designs were synthesized with DC and evaluated based on power consumption, performance, and area. Table II shows evaluation results.

- **Lack of Sensitivity to PPA in Existing Models:** Compared to the original model, the current state-of-the-art model GPT-4o lacks effective optimization for these critical post-synthesis metrics, and the existing supervised fine-tuning hardware generation model SFT-only has limitations in achieving comprehensive performance improvements.
- **Trade-offs Between Metrics:** Optimizing a single metric can negatively impact others. Focusing on power metric, RL-only and SFT-RL models optimized for power (Power, PP, PowerA, and PPA) show significant improvements compared to models that prioritize only area or performance (Area, Perf, and PerfA) at the expense of power.
- **Balanced Optimization Across Metrics:** By comprehensively considering all metrics, the (SFT-)RL-PPA models demonstrate superior performance in both training strategies. The SFT-RL-PPA model improves Power by 20.97%, Performance by 14.68%, and Area by 29.05%, on average compared to the SFT-only model.
- **Design Quality Improvements with RL-based Training:** RL-based training using PPA metrics effectively improves hardware code generation quality. Moreover, the SFT-RL models show more significant performance improvements than RL-only models, with more green cells across a broader range of objectives.

Fig. 2 provides a detailed comparison of the Area and Power changes between the RL-Power model and its original model,

TABLE III: The PPA values for the calendar task based on SFT-only model and SFT-RL-PPA model. Performance is represented by the critical path delay.

Model	Power (mW)	Performance (ns)	Area (μm^2)
SFT-only	1.5884	0.29	536.9616
SFT-RL-PPA	1.4871	0.26	496.0368

Deepseek, for the designs generated on the RTLLM benchmark. The RL-Power model reduces power consumption without significantly increasing area, showcasing its ability to balance power and area trade-offs effectively.

E. Case Study

In this case study, we provide a detailed analysis of the designs generated by the PPA-RTL models before and after optimization, using the design of a `calendar` module as an example. The `calendar` module has two inputs: `CLK` (clock) and `RST` (reset). It includes three variables: `Secs`, `Mins`, and `Hours`, all initialized to zero. Each clock cycle, `Secs` increments by one. When `Secs == 59`, `Mins` increments, and `Secs` resets. When both `Secs == 59` and `Mins == 59`, `Hours` increments, and both `Secs` and `Mins` reset. This cycle continues until `Hours == 23`, `Mins == 59`, and `Secs == 59`, after which all values reset.

Table III compares the PPA values of the `calendar` designs generated by the SFT-RL-PPA model and its base model SFT-only model. Through RL targeted at Power & Performance & Area, the SFT-RL-PPA model effectively balances power, performance, and area requirements. To further analyze model differences, Table IV provides the designs generated by both models.

TABLE IV: The calendar module designs generated by SFT-only model and SFT-RL-PPA model.

SFT-only model

```
always @(posedge CLK or posedge RST)
       begin
   if (RST) begin Secs <= 6'b0; end
   else if (Secs == 6'd59) begin Secs
          <= 6'b0;end
   else begin Secs <= Secs + 1; end
end
always @(posedge CLK or posedge RST)
       begin
   if (RST) begin Mins <= 6'b0; end
   else if (Mins == 6'd59 && Secs ==
         6'd59) begin Mins <= 6'b0;
          end
   else if (Secs == 6'd59) begin Mins
          <= Mins + 1; end
end
always @(posedge CLK or posedge RST)
       begin
   if (RST) begin Hours <= 6'b0; end
   else if (Hours == 6'd23 && Mins ==
         6'd59 && Secs == 6'd59)
         begin Hours <= 6'b0; end
   else if (Mins == 6'd59 && Secs ==
         6'd59) begin Hours <= Hours
         + 1; end
end
```

SFT-RL-PPA model

```
always @(posedge CLK or posedge RST)
       begin
   if (RST) begin Secs <= 0; Mins <=
         0; Hours <= 0; end
   else begin
      if (Secs == 59) begin
      Secs <= 0;
         if (Mins == 59) begin
            Mins <= 0;
            if (Hours == 23) begin
                  Hours <= 0; end
            else begin Hours <= Hours
                  + 1; end end
         else begin Mins <= Mins + 1;
               end
      end else begin Secs <= Secs +
            1; end
   end
end
```

The SFT-only model uses three separate `always` blocks to manage updates for seconds, minutes, and hours, respectively. The processing logic for different time units is separated, enabling the logic clearer and the code easier to manage and expand. However, since the time units are scattered in different `always` blocks, potential race conditions may arise due to execution order. For example, updating `Secs` to zero might occur slightly before updating `Mins`. This brief delay could result in the minute and hour logic not detecting `Secs` as 59 in a given instant, preventing the intended increment. During logical synthesis, we observe this change that `Hours[0]` and `Mins[0]` are at $0.29ns$ on critical path delay, while `Secs[0]` is at $0.28ns$. The SFT-RL-PPA model updates of `Secs`, `Mins`, and `Hours` are completed in the same logical condition, and their updates are synchronized with each other. When `Secs` reaches 59 and resets to 0, `Mins` and `Hours` are updated within the same clock cycle. Concentrating the update logic within a single block reduces potential jitter caused by signal propagation between different `always` blocks, ensuring the atomicity of the update operations and avoiding potential race conditions.

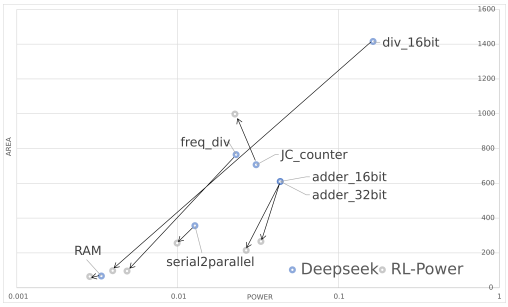


Fig. 2: The comparison of power and area between **Deepseek** (original model) and **RL-Power** (RL model based on Deepseek, optimized for Power) in RTLLM benchmark. The **x-axis** represents power consumption, and the **y-axis** represents area. Arrows indicate the performance shift from the original model (blue) to the RL-Power optimized model (gray) for the same design.

TABLE V: The Syntax and Function accuracy on the RTLLM-v1.1 (pass@5) between the SFT-only model and SFT-RL models.

		SFT-only	SFT-RL					
			Power	Perf	Area	PP	PowerA	PerfA	PPA
RTLLM-v1.1	Syn.(%)	93.1	93.1	96.6	89.6	93.1	86.2	96.6	93.1
pass@5	Func.(%)	48.3	48.3	44.8	41.4	44.8	44.8	48.3	41.4

V. Discussion

Although the PPA-RTL framework effectively achieves optimization goals tailored to different application scenarios, it is essential to evaluate whether integrating PPA into the hardware generation model affects the original model's code generation accuracy. We assess the syntactic and functional accuracy of the SFT-RL-based models on the RTLLM benchmark. Table V compares the SFT-RL models with its base model SFT-only. For syntactic accuracy, the SFT-RL models maintain the same or higher accuracy across five optimization objectives, as syntactically incorrect designs were filtered when selecting preference data based on PPA values, which benefits syntactic correctness. Two models retain their original accuracy levels for functional accuracy, while the other five models show a slight decrease of less than 7%. Overall, PPA-RTL can maintain the original model's accuracy within a limited error.

VI. Conclusion

In this paper, we propose PPA-RTL, a novel approach to align hardware code generation LLMs with variable power, performance, and area requirements. In experimental evaluation, PPA-RTL demonstrates significant improvements compared to traditional LLM-based hardware design, providing a flexible framework for balancing performance, efficiency, and resource usage. Our work paves the way for more adaptive, efficient, cost-effective IC design automated processes.

Acknowledgment

Portions of this work were supported by the National Science Foundation (2340949, 2419880, and the EPSCoR ARISE 2148878 REI Award).

References

[1] J. Kilby, "Invention of the integrated circuit," *IEEE Transactions on Electron Devices*, vol. 23, no. 7, pp. 648–654, 1976.

[2] L. G. Amarù, P. Vuillod *et al.*, "Logic optimization and synthesis: Trends and directions in industry," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, D. Atienza and G. D. Natale, Eds. IEEE, 2017, pp. 1303–1305. [Online]. Available: https://doi.org/10.23919/DATE.2017.7927194

[3] J. Hu, T. Li, and S. Li, "Formal equivalence checking between SLM and RTL descriptions," in *28th IEEE International System-on-Chip Conference, SOCC 2015, Beijing, China, September 8-11, 2015*. IEEE, 2015, pp. 131–136. [Online]. Available: https://doi.org/10.1109/SOCC.2015.7406927

[4] P. Wei and B. Murmann, "Analog and mixed-signal layout automation using digital place-and-route tools," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 29, no. 11, pp. 1838–1849, 2021. [Online]. Available: https://doi.org/10.1109/TVLSI.2021.3105028

[5] G. J. Milne, "The formal description and verification of hardware timing," *IEEE Trans. Comput.*, vol. 40, no. 7, p. 811–826, Jul. 1991. [Online]. Available: https://doi.org/10.1109/12.83619

[6] OpenAI, "Gpt-4 Technical Report," ArXiv, Tech. Rep. abs/2303.08774, 2023.

[7] W. Fu, S. Li *et al.*, "Hardware phi-1.5b: A large language model encodes hardware domain specific knowledge," in *Proceedings of the 29th Asia and South Pacific Design Automation Conference, ASPDAC 2024, Incheon, Korea, January 22-25, 2024*. IEEE, 2024, pp. 349–354. [Online]. Available: https://doi.org/10.1109/ASP-DAC58780.2024.10473927

[8] M. Liu, N. R. Pinckney *et al.*, "Invited paper: Verilogeval: Evaluating large language models for verilog code generation," in *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*. IEEE, 2023, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICCAD57390.2023.10323812

[9] Z. Pei, H. Zhen *et al.*, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=jKnW7r7de1

[10] S. Thakur, J. Blocklove *et al.*, "Autochip: Automating HDL generation using LLM feedback," *CoRR*, vol. abs/2311.04887, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.04887

[11] J. Schulman, "Reinforcement learning from human feedback: Progress and challenges," in *Berkeley EECS Colloquium. YouTube www. youtube. com/watch*, 2023.

[12] H. Mehrafarin, S. Rajaee, and M. T. Pilehvar, "On the importance of data size in probing fine-tuned models," in *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Association for Computational Linguistics, 2022, pp. 228–238. [Online]. Available: https://doi.org/10.18653/v1/2022.findings-acl.20

[13] J. Li, J. Chen *et al.*, "The dawn after the dark: An empirical study on factuality hallucination in large language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 10 879–10 899. [Online]. Available: https://doi.org/10.18653/v1/2024.acl-long.586

[14] H. Pearce, B. Tan, and R. Karri, "DAVE: deriving automatically verilog from english," in *MLCAD '20: 2020 ACM/IEEE Workshop on Machine Learning for CAD, Virtual Event, Iceland, November 16-20, 2020*, U. Schlichtmann, R. Gal *et al.*, Eds. ACM, 2020, pp. 27–32. [Online]. Available: https://doi.org/10.1145/3380446.3430634

[15] J. Blocklove, S. Garg *et al.*, "Chip-chat: Challenges and opportunities in conversational hardware design," in *5th ACM/IEEE Workshop on Machine Learning for CAD, MLCAD 2023, Snowbird, UT, USA, September 10-13, 2023*. IEEE, 2023, pp. 1–6. [Online]. Available: https://doi.org/10.1109/MLCAD58807.2023.10299874

[16] K. Chang, Y. Wang *et al.*, "Chipgpt: How far are we from natural language hardware design," *CoRR*, vol. abs/2305.14019, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.14019

[17] M. Nair, R. Sadhukhan, and D. Mukhopadhyay, "Generating secure hardware using chatgpt resistant to cwes," *IACR Cryptol. ePrint Arch.*, p. 212, 2023. [Online]. Available: https://eprint.iacr.org/2023/212

[18] Anthropic. (2023) Meet claude. [Online]. Available: https://www.anthropic.com/claude

[19] W. Fu, S. Li *et al.*, "A generalize hardware debugging approach for large language models semi-syntectic datasets," *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2024. [Online]. Available: https://doi.org/10.1109/TCSI.2024.3487486

[20] W. Fu, K. Yang *et al.*, "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2023, Tianjin, China, December 13-15, 2023*. IEEE, 2023, pp. 1–6. [Online]. Available: https://doi.org/10.1109/AsianHOST59942.2023.10409307

[21] S. Liu, W. Fang *et al.*, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," 2024.

[22] M. Liu, T.-D. Ene *et al.*, "Chipnemo: Domain-adapted llms for chip design," 2024.

[23] H. Dong, W. Xiong *et al.*, "RLHF workflow: From reward modeling to online RLHF," *CoRR*, vol. abs/2405.07863, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.07863

[24] P. F. Christiano, J. Leike *et al.*, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg *et al.*, Eds., 2017, pp. 4299–4307. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html

[25] L. Ouyang, J. Wu *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed *et al.*, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

[26] R. Rafailov, A. Sharma *et al.*, "Direct preference optimization: Your language model is secretly a reward model," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann *et al.*, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html

[27] D. Guo, Q. Zhu *et al.*, "Deepseek-coder: When the large language model meets programming – the rise of code intelligence," 2024.

[28] S. D. Compiler, "Synopsys design compiler," *Pages/default. aspx*, 2016.

[29] "Rtlcoder dataset." [Online]. Available: https://github.com/hkust-zhiyao/RTL-Coder

[30] Y. Lu, S. Liu *et al.*, "Rtllm: An open-source benchmark for design rtl generation with large language model," 2023.

[31] OpenAI, "Hello gpt-4o," 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

BS-PDN-Last: Towards Optimal Power Delivery Network Design With Multifunctional Backside Metal Layers

Min Gyu Park, Amaan Rahman, Sung Kyu Lim
School of ECE, Georgia Institute of Technology, Atlanta, GA
mgpark@gatech.edu

***Abstract*—The increasing demand to maximize PPA gains with backside metal layers has driven their function beyond power delivery. This study introduces a novel BS-PDN-last flow, crucial for leveraging multifunctional backside, by deferring power routing to the post-signal routing stage. This approach addresses the IR-drop and performance trade-offs inherent in conventional PDN-first flows. Experimental results show that the BS-PDN-last flow achieves a 90% reduction in Total Negative Slack and a 12% performance gain with BS-CDN. Additionally, our work presents the first comprehensive comparison of FS-PDN, BS-PDN, and multifunctional backside designs, evaluated on both physical design and workload metrics leveraging accurate vector-based analysis.**

I. Introduction

As CMOS technology scales down, increasing resistance in the frontside metal layers has led to a substantial IR-drop. The Backside Power Delivery Network (BS-PDN) has emerged as a promising solution, incorporating thicker and less resistive backside metals beneath the silicon substrate. Furthermore, backside power delivery is motivated to decouple the PDN from frontside signal and clock routes, improving power integrity and system performance due to improved frontside routability.

However, backside process incurs additional costs due to advanced processing such as wafer thinning and nano-Through Silicon Vias (nTSVs) formation [4]. To offset these costs, there is an increasing demand to leverage backside metal resources for maximizing Power, Performance, and Area (PPA) gains. For instance, recent research efforts push towards the integration of Backside Clock Delivery Network (BS-CDN) [2] and Backside Signal Routing (BSS) [5]. These studies focus on maximizing the potential of backside metals by extending their function beyond power delivery. However, they adhere to the conventional PDN-first flow, wherein BS-PDN is implemented initially, leaving BS-CDN and BSS to be deployed in the remaining regions. This approach induces substantial disruption of cell placement due to design rule violations encountered during the conversion of frontside cells to the backside, shown in Fig. 1-(a).

For the first time, we investigate the optimal sequence of design steps to maximize the benefit of multifunctional use of backside metal layers. We introduce a novel BS-PDN-last flow, which postpones P/G routing after the completion of frontside clock and signal routing as well as backside clock routing. By positioning P/G routing in the later stages, the BS-PDN-last breaks down the inherent trade-off between IR-drop and performance gain on the conventional PDN-first flow, demonstrating up to an 88% reduction in Total Negative Slack. Furthermore, we offer a comprehensive comparison of frontside only, backside with PDN only, and multifunctional backside designs, assessed through both physical design and workload metrics using precise vector-based analysis. The BS-PDN-last flow with BS-CDN achieves a 16.1% improvement in performance and an 18.9% increase in energy efficiency on the ResNet-50 workload compared to the frontside-only design.

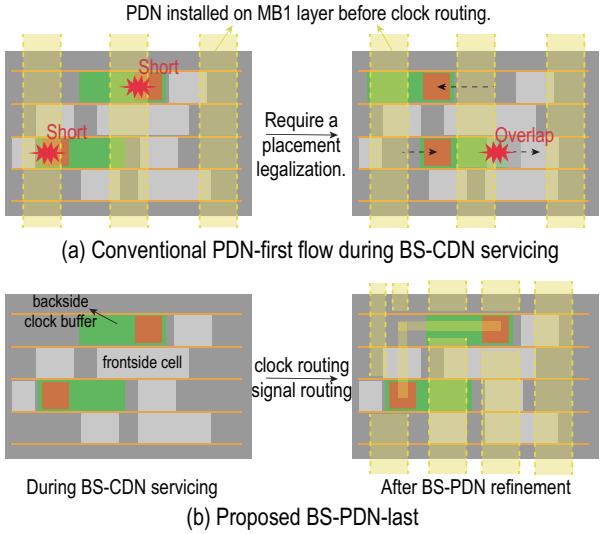


Fig. 1: Comparison between PDN-first and BS-PDN-last flows: The conventional PDN-first flow triggers substantial placement legalization to resolve short between PDN and nTSVs as well as cell overlap. However, BS-PDN-last maintains optimal cell positions with maximization of PDN.

II. Related Work and Shortcoming

Backside Clock Delivery Network was initially presented in [2], [8]. The approach involves building the clock tree on the frontside first, then converting selected clock buffers into backside buffers. This method employs a paired-buffer strategy, as illustrated in Fig. 2-(a): backside-out buffers transfer the clock signal from the frontside to the backside, while backside-in buffers route the clock from the backside to the frontside. To achieve this, nano-TSVs are employed on the output and input pins, respectively, within each cell. They revealed that combining BS-CDN with BS-PDN provides performance improvements of 8.7% in one benchmark. However, the performance gains were marginal in other benchmarks, achieving only 1.3% and 2.8% improvements, respectively. This failed to deliver improvements compelling enough to offset the increased costs of backside integration, raising concerns about whether successful improvements can be achieved in industrial-scale designs. We also face similar challenges when using the PDN-first flow with BS-CDN as shown in Table II.

III. Problem Formulation and Benchmark

A. Problem Formulation

The conventional PDN-first flow employed in prior work [2] follows the sequence of backside power routing, frontside clock construction, backside clock servicing, clock routing, and frontside

signal routing. In scenarios where backside metal layers are utilized for multiple purposes, our goal is to determine the optimal sequence of each step to maximize the benefits of backside integration for both performance and IR drop.

B. Benchmark Architectures

We evaluate the impact of BS-PDN-last flows using the Rocket Chip, an open-source processor, and the Gemmini SoC, a DNN accelerator system, along with a simple pure logic AES benchmark.

The Rocket Chip [1], a RISC-V CPU generator, provides a complete SoC RTL with support for CPU cores, a memory system interfacing with L2 cache, MMIO peripherals, and a DMA device. The specific configuration used in this study includes a dual-core setup, with each Rocket Core featuring a 5-stage in-order scalar processor, along with 16kB L1 data cache and 16kB L1 instruction cache.

The Gemmini project [3] offers a reconfigurable full-stack platform that integrates the DNN accelerator Gemmini, a CPU, and a L2 cache, enabling comprehensive system-level analysis. This platform not only generates hardware RTL for given configurations but also enables workload simulation, providing instance-level toggle rates and total cycle counts. For this study, we utilized a Gemmini accelerator configured with a 16x16 systolic array comprising a 4x4 tile and 4x4 mesh, a 4-bank 128kB scratchpad, a 2-bank 64kB accumulator, a Rocket single core with 16kB L1 data cache and 16kB L1 instruction cache, and a 128kB L2 cache.

IV. Backside PDN-last Flow

A. Motivation

With the conventional PDN-first flow, the conversion of frontside buffers to backside requires careful consideration of Power/Ground (P/G) straps on Backside Metal layer 1 (MB1), as shown in Fig. 2-(b). They can cause a P/G-to-clock short if any overlap happens between MB1 P/G straps and nano-TSV in the backside buffer. Additionally, backside buffers have a longer width than frontside buffers due to the presence of the nano-TSV. This can potentially lead to overlaps with adjacent cells if the back buffer is placed in its original front buffer position, as illustrated in Fig. 2-(c). To address potential design rule check (DRC) violations within a PDN-first flow, Algorithm 1 legalizes cell placement with minimal displacement. Recognizing that relocating adjacent cells facilitates DRC resolution, the algorithm allows the movement of both BS cells and surrounding cells. This is achieved by employing an advanced legalizer option with the *place* status set for all cells, excluding those on timing-critical paths.

However, such an approach can significantly disrupt the original clock tree structure. The impact becomes more pronounced with higher BS-PDN utilization, as shown in Table I: with a 70% BS-PDN utilization in a PDN-first flow, backside clock buffer displacement reaches up to 4.13 μm, equivalent to a vertical shift of approximately 35 rows. Such substantial alterations to the clock tree structure degrades the performance compared to the design without BS-CDN. Furthermore, even with 50% PDN utilization, the effectiveness of backside clock routing is reduced compared to a BS-CDN implementation without PDN. This result reveals the following insights: (1) The PDN-first flow obstructs backside buffer conversion, potentially leading to suboptimal performance or performance degradation. (2) The PDN-first flow introduces a trade-off between IR-drop reduction and performance gain, depending on PDN utilization. This trade-off arises because higher PDN utilization increases interference but reduces IR-drop.

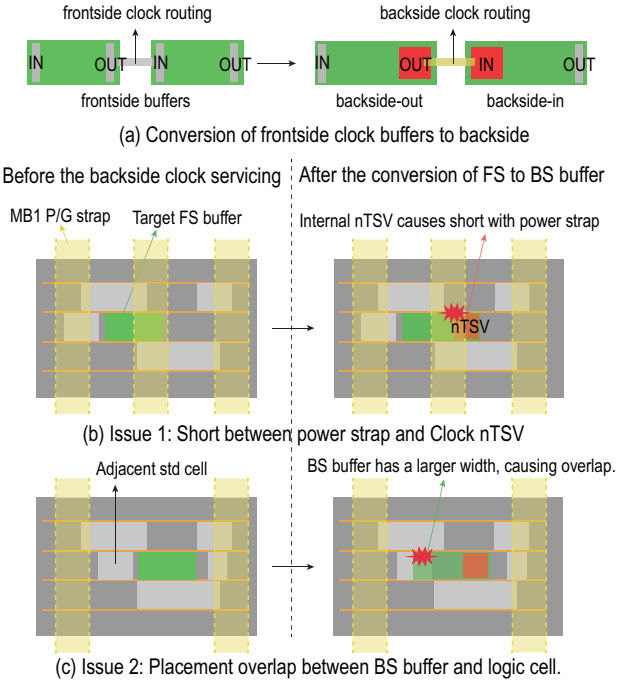


Fig. 2: Backside clock buffer overlap issues in [2]. (b) between a nTSV and a backside wire, (c) between a BS buffer and a frontside logic cell.

TABLE I: Quantification of the overlap issues with PDN-first flow described in Fig. 2. In this experiment, Gemmini SoC services 218 backside clock buffers for BS-CDN.

BS-CDN	disabled	enabled		
PDN util.	90%	None	50%	70%
Displacement of clock buf. (μm)	- -	- -	Mean: 0.27 Max: 0.49	Mean: 0.39 Max: 4.13
WNS (ps)	192.6	114.1 (-40.8%)	177.3 (-7.96%)	206.6 (+7.27%)
D. IR-drop (mV)	54.6	-	89.9	79.3

B. Overview of the Approach

To break down the inherent trade-off in the PDN-first flow, we proposed the BS-PDN-last flow, shown in Fig. 3. It is designed to fully leverage backside metal layers, supporting BS-CDN to maximize the performance gain. This methodology postpones PDN routing to the final stage of physical design, introducing an initial routing blockage on backside metal layers to pre-allocate essential regions. It accommodates various process constraints, including metal density limits, regular P/G pad placement rules, and spacing requirements between backside power and clock routing for signal integrity, through parameter adjustments in the Algorithm 2. Additionally, if the design fails to meet IR-drop targets, the BS-PDN-last flow can re-build PDN with minimal impact on overall physical design time.

C. Step 1: BS-PDN Planning

To defer PDN routing after the signal routing stage, it is crucial to ensure that BS-CDN does not obstruct power delivery to each cell and macro. Therefore, the BS-PDN planning step is implemented before utilizing backside resources for other purposes. This step pre-allocates routing blockages in critical regions to ensure proper P/G routing in subsequent steps.

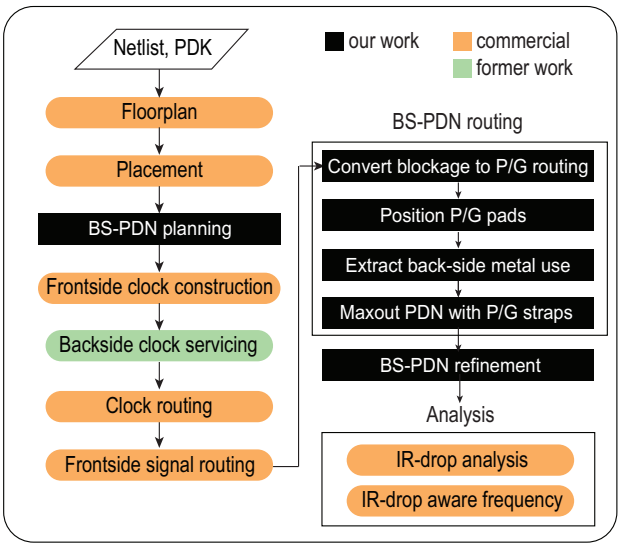


Fig. 3: Proposed BS-PDN-last flow: It preserves optimal backside clock buffer placement while maximizing PDN utilization to mitigate IR-drop.

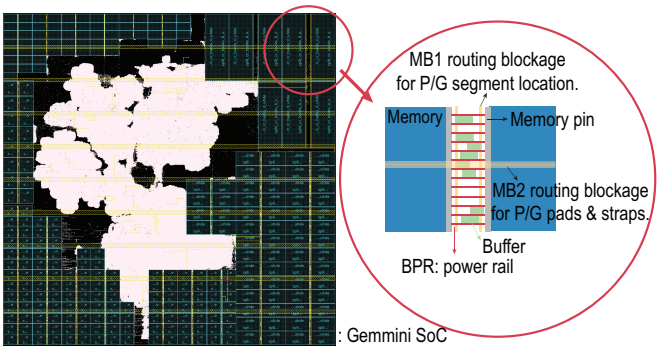


Fig. 4: BS-PDN planning: Routing blockages are added on MB1 and MB2 layers to ensure power delivery to every component.

In detail, for the design with memory macros, EDA tools often place buffers within the narrow spaces between memory macros to meet the timing. During multifunctional backside servicing, backside buffers can limit available space for power routing in these regions. To prevent such issues, Fig. 4 illustrates the placement of routing blockages on the vertical MB1 layer to reserve future P/G segment areas. Moreover, when the P/G pads need to follow a regular arrangement, routing blockages are established on the horizontal MB2 layer to facilitate subsequent assignment of MB2 P/G straps. These MB2 P/G straps will be aligned directly above the regular P/G pads. With these two strategies, essential power routing regions can be pre-allocated, facilitating seamless integration with BS-CDN while maintaining robust power delivery across all cell instances.

D. Step 2: BS-PDN-aware Backside Clock Servicing

Algorithm 1 is employed to identify candidates within the frontside clock tree and switch to backside buffers in both PDN-first and BS-PDN-last flows. Candidates are selected from buffers located on the path between the clock pin and the launch DFF of timing-critical paths. This effectively minimizes Worst Negative Slack (WNS) by reducing the delay from the clock pin to the launch DFF and providing additional timing margin for signal routing between the launch and capture DFFs. To ensure the paired scheme of backside

Algorithm 1 BS-PDN-aware Backside Clock Servicing

Input: $\mathbf{G_{clock}}$: Clock tree graph using only frontside buffers.
$\mathbf{Path_{crit}}$: List of timing critical paths in order of worst case.
$\mathbf{N}$: Target percentage.
Output: $\mathbf{G^*_{clock}}$: Clock tree graph with backside buffers.

```
Lock the location of logic cells (non-clock cells) on critical paths.
for path in {top N% of Path_crit} do
    ctp ← clock tree path from clock pin to launch DFF on path
    for buf in {reverse order of ctp} do
        if dests of buf are all frontside buffers then
            Convert buf to BS-OUT and dests to BS-IN.
            Legalize placement with minimum displacement.
```

buffers, the algorithm verifies that all destinations are frontside buffers, enabling their conversion to backside-in buffers. Furthermore, logic cells on timing-critical paths are fixed during legalization, ensuring no disruption to the placement quality. For the BS-PDN-last flow, as PDN is not installed in the initial phase, only a small amount of MB1 routing blockage and cell placement overlap causes legalization.

E. Step 3: BS-PDN Routing

Once the physical design is done alongside routing blockages on MB1 and MB2 layers, these blockages are converted into P/G routing, and P/G pads are positioned according to a specified pitch. Furthermore, backside metal regions used by BS-CDN and pre-installed BS-PDN are identified to ensure P/G routing compatibility in subsequent steps. For the MB1 layer, regions with existing MB1 metal usage, nano-TSV locations within backside clock buffers, and VB1 (via between MB1 and MB2) locations are extracted. On the other hand, for the MB2 layer, areas with existing MB2 metal usage and VB1 locations are extracted.

Unlike the conventional FS-PDN, which must balance routing resource allocation and IR-drop mitigation due to interference with frontside routing, BS-PDN focuses on maximizing the remaining backside metal resources. This strategy not only minimizes IR-drop but also enhances lateral thermal conduction [6]. By implementing the BS-PDN after BS-CDN servicing, it is possible to fully maximize PDN deployment without concerns of interference with BS-CDN.

Accordingly, first step of Algorithm 2 positions P/G straps between existing backside metal usage while avoiding overlap and spacing rule violations. For the maxout process with P/G straps, we can adjust the width and pitch of P/G straps to achieve the target PDN utilization, while also varying spacing to control the impact of P/G straps on backside clock routing. These adjustments help to manage key factors, including IR-drop, thermal conduction, coupling capacitance, and signal integrity.

F. Step 4: BS-PDN Refinement

In conventional power routing, P/G straps are used in a form that connects both ends of the core to ensure connectivity between different metal layers regardless of strap locations. However, when a sufficient number of P/G straps are installed, short P/G segments can also maintain reliable inter-layer connections. Additionally, as illustrated in Fig. 5-(a), in cases where backside clock routing is densely packed, broad areas without P/G straps happen, leading to significant IR-drop. Therefore, Algorithm 2 positions shorter P/G segments between existing P/G straps in step 2, avoiding any overlap with previously installed backside routing. By incorporating

Algorithm 2 BS-PDN Routing & Refinement.

Input: $\mathbf{GDSII_{in}}$: Layout with P/G planning and backside clock.
$\mathbf{MB1_x}, \mathbf{MB2_y}$: Lists of [*start*, *end*] intervals indicating regions where the respective metal layers are occupied.
$\mathbf{MB1_y}, \mathbf{MB2_x}$: Intervals in the y and x-direction representing the occupied ranges for each interval in $\mathbf{MB1_x}$ and $\mathbf{MB2_y}$.
$\mathbf{w1}, \mathbf{w2}, \mathbf{w3}, \mathbf{w4}$: Target widths of P/G.
$\mathbf{p1}, \mathbf{p2}, \mathbf{p3}, \mathbf{p4}$: Target pitch values.
$\mathbf{spacing}$: P/G spacing from backside clock routing.
Output: $\mathbf{GDSII_{out}}$: Layout with maximized P/G utilization.

Step 1: Maxout BS-PDN with P/G straps.
for i in $\{1, 2\}$ **do** ▷ loop over MB1 and MB2.
  $unit \leftarrow \mathbf{w}i + \mathbf{p}i$
  for j in $\{0, \ldots, \text{len}(\mathbf{MB}i_{\mathbf{x}|\mathbf{y}})\}$ **do**
    ▷ Insert P/G straps between consecutive occupied regions.
    $interval \leftarrow start_j - end_{j-1}$
    if $interval - 2 \times \mathbf{spacing} > unit$ **then**
      $N \leftarrow \text{CEIL}((interval - 2 \times \mathbf{spacing})/unit)$
      Integrates N P/G straps between end_{j-1} and $start_j$.
Step 2: Add P/G Segments
$unit \leftarrow \mathbf{w}(i+2) + \mathbf{p}(i+2)$ ▷ loop over MB1, MB2. ($i \in \{1, 2\}$)
for j in $\{0, \ldots, \text{len}(\mathbf{MB}i_{\mathbf{x}|\mathbf{y}})\}$ **do**
  $width \leftarrow end_j - start_j$
  if $width > unit$ **then**
    $M \leftarrow \text{CEIL}(width/unit)$
    for k in $\{0, \ldots, \text{len}(\mathbf{MB}i_{\mathbf{y}|\mathbf{x}}[\mathbf{j}])\}$ **do**
      $H \leftarrow start_k - end_{k-1}$
      Inserts M P/G segments of $\mathbf{w}i$ width and H height.

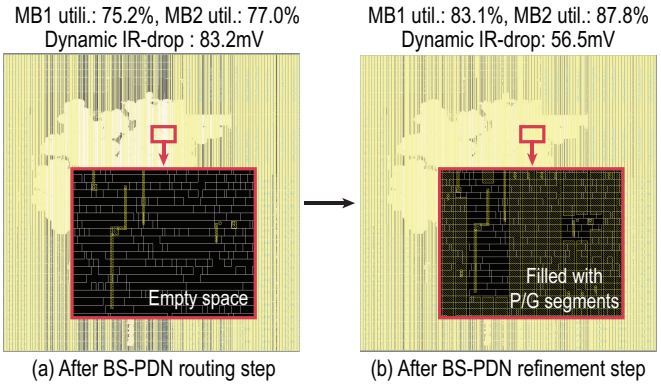


Fig. 5: Maximizing PDN utilization through the addition of P/G wire segments during the BS-PDN refinement step.

supplementary P/G segments, BS-PDN utilization is fully maximized, achieving reduced IR-drop, as demonstrated in Fig. 5-(b).

V. Results and Discussion

A. Experimental Setup

For the Gemmini SoC benchmark, we incorporate waveform-based analysis with the instance-wise switching activity throughout workload execution, whereas other benchmarks use common methods that assume uniform switching across all DFFs. Designs with backside integration leverage six frontside metal layers and two backside metal layers, along with buried power rails, across all three benchmarks.

The detailed methodology of waveform-based analysis is presented in Fig. 6. First, workload simulation using Synopsys VCS provides

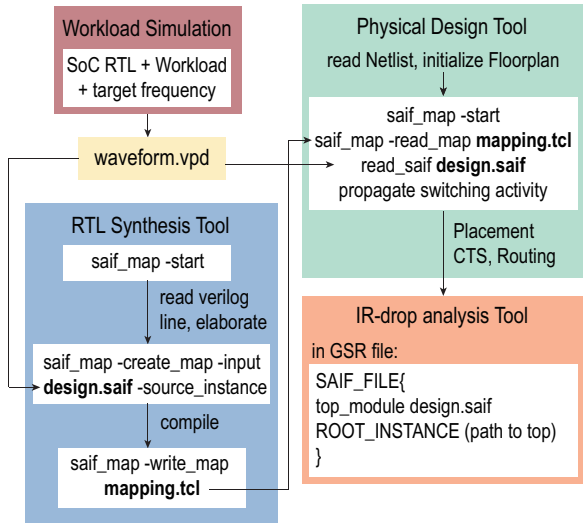


Fig. 6: Waveform-based analysis used in this study for Gemmini SoC benchmark. It enables a comprehensive evaluation of PDN-first and BS-PDN-last flows in real-world applications.

output in the Value Change Dump Plus (VPD) format, detailing value changes across each RTL component. However, component-wise VPD files can be extremely large; for instance, a 16x16 Gemmini SoC running ResNet-50 generates approximately 1TB of waveform data. To streamline power analysis, the waveform is converted into a Switching Activity Interchange Format (SAIF) file, summarizing average toggle counts across the duration of the workload.

Using the GTCAD 3nm PDK [9] at a supply voltage of 0.7V, Synopsys Design Compiler facilitates mapping between RTL instances in the SAIF file and gate-level cells in the synthesized netlist file by creating a naming correlation file. Using Synopsys ICC2, toggle counts from the SAIF file are then mapped into the corresponding cells, enabling precise vector-based power consumption estimates under workload conditions.

For IR-drop analysis, Ansys Redhawk incorporates SAIF files to yield a realistic IR-drop profile. This IR-drop information is then integrated into performance analysis through Synopsys Primetime, leveraging gate-level cell-specific voltage drops to improve the accuracy of timing assessments.

Using these comprehensive methodologies, workload runtime, energy consumption, and energy efficiency are calculated as shown below. Total cycles can be obtained from workload simulation. In this study, we use the inference of ResNet-50 workload with a batch size of 4 to analyze the impact of PDN-last flow on workload metrics.

$$\text{Runtime (ms)} = \frac{\text{Total Cycles}}{\text{Performance (GHz)} \times 10^6}$$

$$\text{Energy (mJ)} = \frac{\text{Vector-based Power (mW)} \times \text{Runtime (ms)}}{10^3}$$

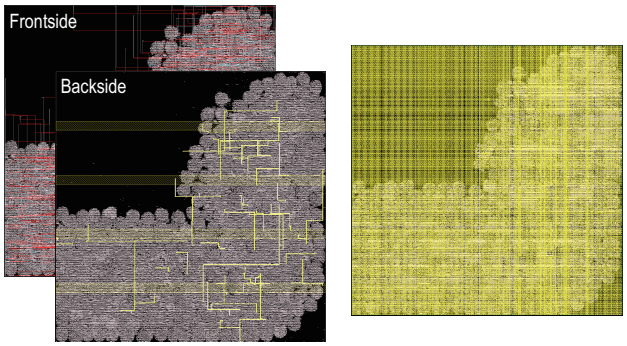
$$\text{Energy Eff. (TOPS/W)} = \frac{\text{Trillion Operations (TOP)}}{\text{Runtime (ms)} \times \text{V.-Power (mW)} \times 10^6}$$

B. Comparison between PDN-first and PDN-last Flow

Given the trade-off between IR-drop and performance in the PDN-first flow, it is important to select the PDN specifications carefully to enable a fair comparison between the PDN-first and PDN-last flows. In this work, we apply the same PDN utilization achieved from the PDN-last flow to ensure comparable IR-drop levels while demonstrating the performance advantages.

TABLE II: Iso-performance comparison between BS-PDN-first vs. PDN-last flow. BS-CDN is added in both cases (baseline: BS-PDN only.)

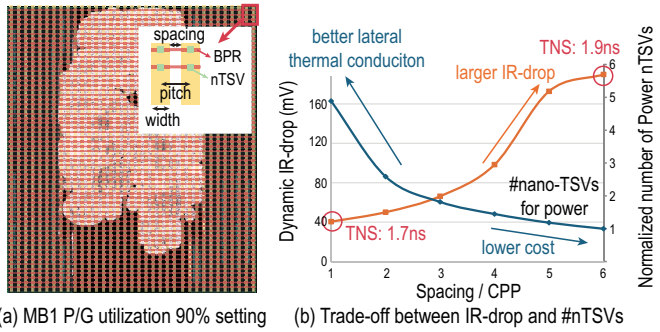
	AES (#cell=163K, 7.5GHz)			Rocket Chip (#cell=388K, 2.6GHz)			Gemmini SoC (#cell=1.17M, 1.5GHz)		
Backside PDN Backside CDN	PDN-first no	PDN-first yes	PDN-last yes	PDN-first no	PDN-first yes	PDN-last yes	PDN-first no	PDN-first yes	PDN-last yes
MB1 PDN util.	90%	80.6%	81.6%	90%	79.4%	79.1%	90%	82.7%	83.1%
MB2 PDN util.	90%	95.2%	95.8%	90%	85.6%	83.3%	90%	88.3%	87.8%
#Clock nTSV	-	127		-	306		-	217	
Max. disp. (μm)		2.94	0.41		3.61	0.42		37.4	0.43
WNS (ns)	26.0	13.2	9.2 (-65%)	46.8	36.7	21.9 (-53%)	192.6	235.85	135.3 (-30%)
TNS (ps)	5.1	0.66	0.47 (-90%)	7.5	6.7	1.7 (-77%)	35.3	203.96	24.4 (-31%)
Eff. freq (GHz)	6.27	6.83	7.02 (+12%)	2.32	2.37	2.46 (+6%)	1.16	1.11	1.25 (+8%)
Power (mW)	133.2	136.5	138.2	228.7	232.1	240.8	642.0	604.9	670.8
D. IR-drop (mV)	32.5	29.9	25.5	37.3	44.2	40.5	54.6	55.9	56.5



(a) Frontside and Backside clock routing with routing blockage on MB2 for BS-PDN planning

(b) Backside PDN structure with BS-PDN-last flow

Fig. 7: Final CDN and BS-PDN routing with BS-PDN-last flow on AES.



(a) MB1 P/G utilization 90% setting

(b) Trade-off between IR-drop and #nTSVs

Fig. 8: Impact of backside metal layer 1 (= MB1) P/G width & pitch on Rocket Chip.

Table II shows an iso-performance comparison across three configurations: BS-PDN only, BS-CDN with PDN-first flow, and BS-CDN with BS-PDN-last flow for each benchmark. As the BS-PDN-last flow defers PDN installation with minimal routing blockages, cell displacement during backside clock buffer conversion is significantly reduced compared to the PDN-first flow. Therefore, the BS-PDN-last flow successfully preserves the optimal logic cell and clock buffer positions, addressing the limitations of the PDN-first flow and achieving significant performance gains.

The results also indicate, as design size increases, legalizing placement for backside buffers within a PDN-first flow becomes significantly more challenging, leading to greater maximum displacements. This is because larger designs require stronger drive-strength buffers to meet clock specifications; however, these wider (=stronger) backside buffers are more difficult to legalize under a PDN-first approach. Consequently, the performance benefits gained from backside clock routing are further diminished. Given that the Gemmini SoC used in this study comprises 1.17 million cells—and recognizing that typical industrial designs are substantially larger—these observations highlight the critical importance of adopting a BS-PDN-last flow.

C. Impact of Backside Metal Width & Pitch

For a given PDN utilization, adjustments to P/G width and pitch are possible. For instance, a 90% target utilization can be maintained regardless of the absolute width and spacing values, as long as the width-to-spacing ratio remains 9:1. In this section, we examine the impact of varying the MB1 P/G spacing from 1x to 6x Contacted Poly Pitch (CPP) while maintaining a consistent 90% target utilization. Although Fig. 8-(a) illustrates a simplified layout with a regular BS-PDN arrangement, the actual design contains irregular pattern of P/G straps and segments similar to Fig. 7-(b). Under the current setting, MB1 utilization remained relatively stable, ranging from 79% to 81%, ensuring an equitable comparison across different spacing options.

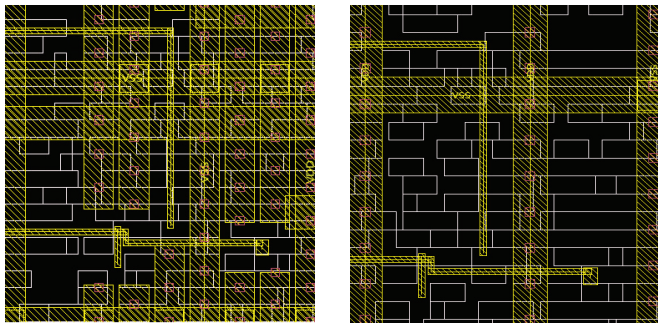
As shown in Fig. 8-(b), larger spacing increases the length of the high-resistance BPR paths, resulting in a larger dynamic IR-drop. As a result, timing metrics are negatively impacted, increasing the number of timing failure points from 428 to 480 and raising Total Negative Slack (TNS) from 1.7ns to 1.9ns. On the other hand, reducing the width and spacing of MB1 P/G straps increases the number of power nano-TSVs for MB1-BPR connection. Consequently, it improves lateral thermal conduction [6]; however, manufacturing costs increase. In summary, the MB1 width and pitch settings have a direct impact on the number of nano-TSVs and the power delivery path, creating a trade-off between IR-drop, TNS, thermal conduction, and cost.

D. Impact of Spacing between Backside Power and Clock Wires

The application of BS-CDN necessitates the PDN to share metal layers with clock routing, which introduces potential signal integrity issues due to coupling capacitance. To evaluate this, we compute the coupling capacitance by comparing the total capacitance of the backside net before and after PDN implementation. The spacing between P/G lines and clock/signal routing are varied from 1x-20x and 40x of the minimum pitch, respectively. According to Table III, increasing the spacing reduces total coupling capacitance by 2.9% and 3.1%, respectively. However, with increased spacing, as illustrated in Figure 9, the length and number of P/G straps decreased. Therefore, substantial spacing can hinder the installation of P/G straps, which

TABLE III: Effect of spacing between backside P/G and backside clock routing. The spacing in the x and y directions is set as a multiple of the minimum pitch for the MB1 and MB2 layers, respectively.

Spacing	1x	20x	40x
Total coupling cap. (fF)	49.4	47.9 (-2.9%)	47.8 (-3.1%)
Average PDN util. (%)	83.4	83.1	82.9
Dynamic IR-drop (mV)	37.3	38.2 (+2.4%)	39.9 (+7.0%)



(a) spacing = 1x min_pitch (b) spacing = 40x min_pitch

Fig. 9: Different spacing between power and clock net on Rocket Chip.

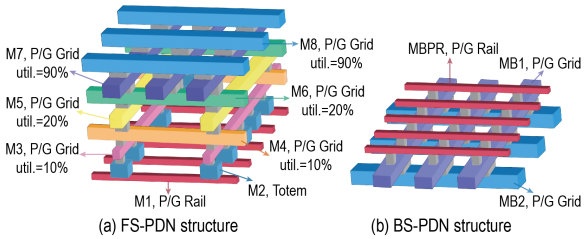
exacerbates IR-drop. Moreover, increasing P/G strap spacing beyond a specific threshold incurs minimal impact on coupling capacitance. Thus, determining an optimal spacing level is crucial to balance coupling capacitance minimization and efficient P/G planning.

E. Workload Simulation Results

This section presents a comparative analysis of FS-PDN, BS-PDN, and BS-CDN + BS-PDN-last flow on the Gemmini SoC, evaluating both physical design-related and workload-related metrics. Designs utilizing backside metal layers implement power grids on MB1 and MB2 with buried power rails, whereas FS-PDN designs construct power grids from M3 to M8 with distinct utilization, with power rails located on M1 as shown in Fig. 10-(a). The FS-PDN specification was configured to approach a worst-case dynamic IR-drop as close to 10% of the supply voltage as possible, while avoiding significant interference with frontside routing.

The GTCAD 3nm PDK [9] used in this study provides cells with consistent height across both BPR-enabled and BPR-free configurations, resulting in an equivalent footprint for all designs. Rather than achieving footprint savings as seen in conventional FS-PDN vs. BS-PDN analysis [7], our BS-PDN designs achieve power benefits due to reduced parasitic capacitance at cell pins as well as smaller load capacitance from better frontside routability.

As shown in Table IV, the design integrating BS-CDN with the BS-PDN-last flow achieves a 16.1% enhancement in performance, along with a 13.7% reduction in ResNet-50 runtime, a 17.4% improvement in energy consumption, and an 18.9% increase in energy efficiency. Additionally, BS-CDN with BS-PDN-last flow demonstrates substantial gains over the design with BS-PDN alone, especially in terms of performance and runtime, by 7.8% and 7.3% respectively. Given the relatively low cost of extending existing backside metal layers from single-function BS-PDN to multi-functional BS-PDN + CDN designs—compared to the higher cost of converting a frontside-only design to a backside-integrated —the importance of leveraging backside metal layers for diverse functionalities becomes evident.



(a) FS-PDN structure (b) BS-PDN structure

Fig. 10: PDN structure used in this work: the baseline FS-PDN uses P/G rails on M1 and grids on M3–M8, while the BS-PDN is implemented on the BPR and backside.

TABLE IV: Comparison between FS-PDN, BS-PDN, BS-CDN + PDN-last with 3nm PDK (supply voltage=0.7V). Workload metrics (runtime, energy, E-eff) are based on ResNet-50 with a batch size of 4.

Gemmini SoC	FS-PDN	BS-PDN	BS-PDN + CDN
PDN flow	PDN-first	PDN-first	BS-PDN-last
PDN pattern	regular	regular	irregular
PDN util. (%)	based on Fig. 10-(a)	MB1: 90 MB2: 90	MB1: 83.1 MB2: 87.8
# metals (f+b)	8+0	6+2	6+2
Footprint	$452\mu m \times 452\mu m$		
Vector-based Power (mW)	690.4 -	642.0 (-7.0%)	670.8 (-2.8%)
Perf. (GHz)	1.08	1.16 (+7.7%)	1.25 (+16.1%)
D. IR-drop (mV)	89.8	54.6	56.5
Runtime (ms)	115.5	107.5 (-6.9%)	99.7 (-13.7%)
Energy (mJ)	79.7	69.0 (-13.4%)	65.8 (-17.4%)
Energy-eff. (TOPS/W)	0.185 -	0.214 (+15.7%)	0.220 (+18.9%)

VI. Conclusion and Future Work

We present a BS-PDN-last flow that enhances performance and energy efficiency while maintaining IR-drop levels in multifunctional backside metal layer applications. By deferring PDN routing to the final design phase, this approach achieves up to a 90% reduction in Total Negative Slack and up to a 12% increase in performance when integrated with BS-CDN, effectively eliminating the trade-off between IR-drop and performance gain inherent to PDN-first flows. Notably, the benefits of the BS-PDN-last flow scale significantly with increasing design complexity, highlighting its relevance for industrial-scale chip implementations.

While our current focus is on backside clock and power routing, the proposed approach is readily extendable to backside signal routing (BSS). Since BSS involves relocating critical frontside signal nets to the backside, it necessitates the use of nano-TSVs and the MB1 layer, resulting in a competitive allocation with the BS-PDN—similar to the conflict observed with BS-CDN. Especially, the MB1 routing blockages introduced during our BS-PDN planning stage are effective in mitigating issues associated with memory-to-register frontside net conversions to backside.

Acknowledgment

This work was supported by the Semiconductor Research Corporation under the JUMP 2.0 Center Program (CHIMES), Samsung Advanced Institute of Technology (SAIT) under the AI for Semiconductors Program, Samsung Semiconductor Logic Path Finding Lab, and the Ministry of Trade, Industry & Energy of South Korea (1415187652, RS-2023-00234159).

REFERENCES

[1] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, et al. The Rocket Chip Generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 4:6–2, 2016.

[2] N. E. Bethur, P. Vanna-Iampikul, O. Zografos, L. Zhu, G. Sisto, D. Milojevic, A. García-Ortiz, G. Hellings, J. Ryckaert, F. Catthoor, et al. GNN-assisted Back-side Clock Routing Methodology for Advance Technologies. In *61st ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024.

[3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, et al. Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration. In *58th ACM/IEEE Design Automation Conference (DAC)*, pages 769–774. IEEE, 2021.

[4] A. Jourdain, M. Stucchi, G. Van der Plas, G. Beyer, and E. Beyne. Buried Power Rails and Nano-Scale TSV: Technology Boosters for Backside Power Delivery Network and 3D Heterogeneous Integration. In *IEEE 72nd Electronic Components and Technology Conference (ECTC)*, pages 1531–1538. IEEE, 2022.

[5] B.-S. Kim, S. Choi, J. H. Lee, K. Lee, J. Park, J. Kwon, S. Park, K. Chun, H. Simka, A. Kumar, et al. Expanding Design Technology Co-Optimization Potentials with Back-Side Interconnect Innovation. In *IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 1–2. IEEE, 2024.

[6] H. Oprins, J. L. R. Bohorquez, B. Vermeersch, G. Van der Plas, and E. Beyne. Package level thermal analysis of backside power delivery network (BS-PDN) configurations. In *21st IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (iTherm)*, pages 1–7. IEEE, 2022.

[7] D. Prasad, S. T. Nibhanupudi, S. Das, O. Zografos, B. Chehab, S. Sarkar, R. Baert, A. Robinson, A. Gupta, A. Spessot, et al. Buried Power Rails and Back-side Power Grids: Arm® CPU Power Delivery Network Design Beyond 5nm. In *IEEE International Electron Devices Meeting (IEDM)*, pages 19–1. IEEE, 2019.

[8] P. Vanna-iampikul, H. Yang, J. Kwak, J. X. Hu, A. Rahman, N. E. Bethur, C. Hao, S. Yu, and S. K. Lim. A Design Methodology for Back-side Power and Clock Routing Co-Optimization (Invited). In *IEEE Symposium on VLSI Technology Circuits (VLSI Technology and Circuits)*. IEEE, 2024.

[9] J.-S. Yoon, J. Jeong, S. Lee, J. Lee, S. Lee, R.-H. Baek, and S. K. Lim. Performance, Power, and Area of Standard Cells in Sub 3 nm Node Using Buried Power Rail. *IEEE Transactions on Electron Devices (T-ED)*, 69(3):894–899, 2022.

PASK: Cold Start Mitigation for Inference with Proactive and Selective Kernel Loading on GPUs

Xuanteng Huang*, Jiangsu Du†, Nong Xiao‡, Xianwei Zhang§
School of Computer Science and Engineering
Sun Yat-sen University
Guangzhou, China
*huangxt57@mail2.sysu.edu.cn, {†dujiangsu, ‡xiaon6, §zhangxw79}@mail.sysu.edu.cn

***Abstract*—Today, DNN inference is widely adopted, with numerous inference services being spawned from scratch across instances in scenarios such as spot serving, serverless scaling and edge computing, where frequent start-stops are required. In this work, we first delve into the inference workflow and uncover the origins of cold start when invoking a DNN model. Specifically, DNN execution is blocked by the kernel loading process to prepare the code object executing on GPU at the DL primitive library (e.g., cuDNN and MIOpen). To tackle this, we propose PASK, a kernel loading and reusing middleware to mitigate the widespread cold start issue. Unlike the reactive kernel scheduling policy used by existing frameworks, PASK adopts a proactive strategy to interleave code loading, kernel issuing and GPU computation to achieve higher hardware utilization. To further reduce the loading overhead, PASK recycles existing loaded kernels to accomplish the DNN operator, rather than introducing new kernels for every layer. Meanwhile, PASK categorically organizes the cached kernels to efficiently find the applicable kernel for reuse and thus minimize incurred runtime overhead. We implement and evaluate PASK atop of open source DNN inference engine and primitive library on off-the-shelf GPUs. Experiments demonstrate PASK is capable of alleviating the cold start overhead of popular DNN models with $5.62\times$ speedup on average.**

***Index Terms*—Cold Start, GPU, DNN, Inference**

I. Introduction

The last decade has witnessed the unprecedented prosperity of deep learning and its widespread application in many aspects like computer vision [1]–[9], recommendation [10], and natural language processing [11]. Serving model inference has become the dominating workload in modern data centers, which accounts for more than 90% infrastructure costs in AWS [12] and 200 trillion daily invocations in Meta [13].

This trend also sparks the research efforts devoted into the ML inference infrastructures like request batching [14]–[17], model selection [18], [19], memory optimizations [20], tensor pre-fetching [21] and resource sharing [22], [23]. As underlying support, the accelerators like GPUs are trending to be much more powerful and the DNN operators are continuously optimized by experts for efficient computation [24]–[27]. As a result, DNN inference tends to be more hindered by the non-computation aspects, with a significant portion being spent on the models' cold starting process [28]. Fig. 1(a) compares the cold and hot execution times (ratios between the first and successive iterations) of representative DL models on both data center and consumer-grade GPUs from different vendors, where significant cold start slowdowns are commonly observed on respect platforms ($23.7\times$, $19.5\times$ and $31.3\times$ for MI100, A100 and 6900XT, respectively). Fig. 1(b) further demonstrates the breakdowns of the cold start overhead, which is classified into four parts by their execution ordering phases. Before launching a kernel onto the GPU, the host runtime (i.e., CUDA or HIP) is responsible for checking whether the kernel code object (e.g., compiled SASS instructions binary stream) is present in the managed host memory. If not, the runtime will load it from either the shared library or compiled ELF executables, then set memory permissions and look up the target symbol positions. It can be captured from Fig. 1(b) that the majority (65.8%) of cold start overhead is attributed to loading absent kernel code objects (`code loading`) while the computation (`GPU execution`) constitutes only a small proportion (8.4%). Therefore, the code loading should be considered alongside data pre-fetching [21], keep alive [29] and pre-warming [30] techniques to mitigate DNN cold start overhead.

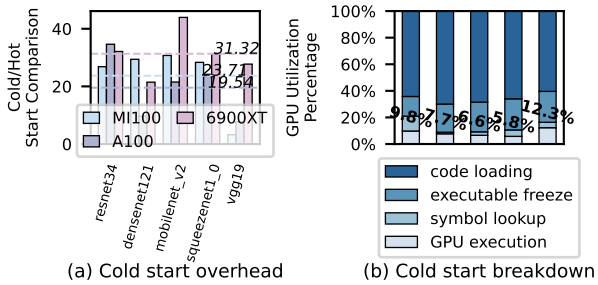


Fig. 1. DNN model cold start (a) overhead and (b) breakdown of typical models. The dashed lines in (a) represents averaged slowdowns and percentage numbers in (b) mark the GPU execution portion.

The cold start overhead originated from code loading is inevitable in scenarios like preemptive serving [31]–[36], serverless scaling [17], [20], [29], [37] or edge computing [30], [38], [39]. For example, the inference service has to be migrated across preemptive instances, or deployed in new instances to cope with request spikes. In edge or mobile devices, the inference service is forced to be suspended or swapped out due to limited hardware resources (e.g., memory capacity), and restarts later. Under these circumstances, code loading is unavoidable when starting a DNN model as there are no available loaded kernels in the system.

There are multiple proposed approaches to mitigate the non-computation overhead of DNN models [25]–[27], [40]. TASO [25], Rammer [26], Cocktailer [27] and MonoNN [40] apply some operator-level transformations on the models' computation graph for higher execution efficiency. Nevertheless, these methods focus on the *static* optimizations to produce a series of fast kernels running on GPUs, failing to consider the crucial runtime overhead of loading these kernels. There also exist designs sharing container [29], [41], runtime [30], [42] or tensor [20] to warm up the cold models, but

§ Xianwei is the corresponding author.

This research was supported by the National Key R&D Program of China under Grant NO. 2022ZD0115304, the National Natural Science Foundation of China #62472462, #62461146204, #62402534, #62332021, the GuangDong Basic and Applied Basic Research Foundation: 2023A1515110117, and was sponsored by CCF-Tencent Rhino-Bird Open Research Fund (CCF-Tencent RAGR20240102).

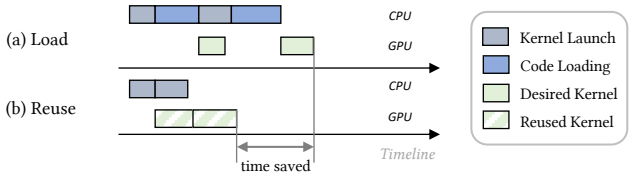


Fig. 2. Instead of waiting for the fresh loading (a), PASK reuses existing loaded kernels (b) to implement the DNN layers.

the sharing granularity is generally too coarse as DNN models are monolithic and cannot be efficiently transformed via dependency analysis for traditional services [43]. As such, extra steps are needed to transform the shared resources for accommodating with the cold start model. NNV12 [38] optimizes the DNN cold start process via selecting different kernels to implement the NN operator to reduce the tensor layout interchange overhead before execution. Nonetheless, the above existing methods ignore the kernel code object loading procedure, which actually constitutes the most significant portion of the cold start overhead for DNN models (Fig. 1).

In this work, we propose PASK, a kernel loading and reusing approach to mitigate the cold start latency of DNN inference. We first analyze the DNN inference serving procedure and attribute the cold start overhead to the mismatch between static model materialization and runtime kernel loading: inference framework determines the finalized kernel sequences from abstract model computation graph without considering the incurred loading overhead at runtime. To cope with this, PASK seeks the opportunities to *reuse* existing potentially less performant but already loaded kernels to materialize NN operator, thereby skipping the more substantial kernel loading procedure, as depicted in Fig. 2.

However, it's non-trivial to efficiently select the substitute from all cached kernels as checking whether the kernel is applicable for current problem can be pretty time-consuming. To this end, we design a categorical cache which stores the cached kernels based on their patterns to minimize the applicable checking overhead. We integrate PASK into the open source inference stack with AMD GPUs [1], and conduct evaluations on various DNN models which show remarkable performance improvements.

To summarize, this work makes the following contributions:

- We investigate the origins of DNN model cold start overhead, and uncover it is primarily due to the mismatch between static model materialization and runtime kernel loading.
- With the insights, we propose PASK, a kernel loading and reusing approach leveraging the already loaded kernels to finalize the NN operator, thus circumventing the substantial kernel loading process. Equipped with interleaved execution and categorical cache, PASK achieves higher GPU utilization meanwhile minimizing the incurred runtime overhead.
- We implement PASK atop of open source inference software stack, and evaluate using representative DNN models to demonstrate its ability to mitigate the cold start overhead.

II. Background and Motivation

A. DNN Model Serving with GPU

GPU has become the *de-facto* accelerator to speed up DNN execution. A DNN model is composed of multiple layers with each defining a specific operation like convolution or pooling. To harness the hardware resources, DNN operators are offloaded onto GPUs for faster execution through various frameworks like TensorRT [44] for NVIDIA and MIGraphX [45] for AMD GPUs. These frameworks *materialize* the DNN layers with the computation graph optimizations [46]–[48] and tuned kernels [24], [49] for efficient execution.

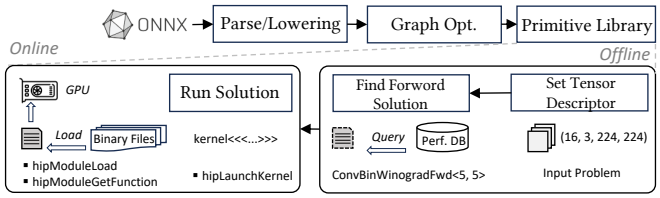


Fig. 3. DNN inference serving workflow. The serving framework firstly lowers the model and applies multiple optimizations on the requested model, then invokes operators from the primitive library to materialize the model.

Offline preparation. Fig. 3 depicts a typical workflow of DNN serving with GPU, where the requested model is submitted in the ONNX format containing multiple canonical operators [50] for the ease of model description and exchange. After receiving the model, the serving framework first converts the ONNX structure into lower descriptions with regard to the backend hardware, including GPU context creation, host-device synchronization setup, and tensor memory layout generation. Then the serving framework applies a series of hardware-independent graph-level optimization passes like *dead code elimination* and *common subexpression elimination* [45], [51] to reduce unnecessary computations. Finally, the serving framework exploits the highly-optimized kernels from vendor-provided primitive libraries (e.g., MIOpen and cuDNN) to accomplish the operators defined in the computation graph.

For each DNN layer (e.g., convolution), the serving framework sets the tensor descriptors needed by the primitive library with input *problem* (image and filter sizes, number of filters, data types etc.). Then the library finds the optimal *solution* [2] for the given problem from all applicable ones by querying an integrated database [52] which records the anticipated performance of each solution on current problem. After this, the lower operations and the determined optimal solutions for each layer are saved in framework-specific formats (e.g., `.trt` files for TensorRT and `.mgx` files for MIGraphX) for online inference request serving. The framework usually maintains a model registry [53] to store the lowered model and directly loads them when the request comes to avoid redundant lowering.

Online inference serving. When the inference request arrives, the framework invokes the corresponding optimized model and runs the determined solutions sequentially by launching the underlying kernels onto the GPU. Nevertheless, there are often cases where the code object of the desired kernel is absent due to the lazy loading behavior [54] which delays the loading until launch. Therefore, the primitive library has to *load* the code objects for the missing kernels from compiled binary files (e.g., dynamic linked ELF), resulting in unbearable cold start latency for model inference. Only after the code objects are loaded into memory will the kernels be issued to the GPU for computation. Such a reactive launch and lazy loading behavior leads to substantial latency at model cold start. Note that the loading delay comes from the mismatch between the offline operator lowering and the runtime solution loading: the primitive library determines solutions from the GPU performance perspective, meanwhile introducing extra loading overhead when launching missing kernels. In

[1]This can be implemented on NVIDIA platforms as well if more flexible interfaces are accessible.

[2]One solution may contain multiple kernels to transform input/output tensor layout/precision, conduct primary computation and epilogue reduction.

principle, the gap can be effectively bridged if one can proactively interleave the code loading and kernel launch executions, and reuse already loaded kernels to accomplish the computation rather than loading missing kernels from scratch each time.

B. Generalized Solution for Wider Applicability



Fig. 4. Generality-performance trade-offs for *Winograd* solutions. Specialized solutions are more performant but applicable for specific problems, while generic solutions can be applied to more problems but are less efficient.

Solution specialization. Fig. 4 demonstrates some exemplified convolution solutions embedded in MIOpen. The solution usually follows certain *patterns* to conduct the computation. For example, solutions using *Winograd* [55] accomplish the convolution via matrix multiplication, and apply some transformations on both the input image and filters to lessen the number of multiply-and-add instructions. Other commonly used patterns include *GEMM*, *DirectConv* and *ImplicitGEMM*. Solutions with the same pattern typically follow similar workflows to accomplish the computation. Besides, there are further optimization spaces to exploit if certain problem fields are deterministic before the model starts (i.e., compile time). In Fig. 4, instead of taking the `ConvWinogradNaiveFwd` solution which accepts inputs with any dimension, one can alter to invoke `ConvBinWinogradRxSFwd` if the input is deemed to be a 2D image. By avoiding unnecessary loops and boundary checks, the latter hence brings in higher GPU performance, but at the expense of generality. Furthermore, the library opts for leveraging more specialized solution `ConvBinWinogradFwd<5,5>` if the filter size is also a specific value (5×5), which helps organize the shared memory layout and better overlap the memory access and computation within the kernel. The highly specialized solutions produce better GPU performance tailored for the specific problem, but they are not applicable to other problems as there are more constraints. Currently, the primitive libraries (cuDNN, MIOpen) tend to harness the specialized solutions from the performance perspective, and fallback to the more general solutions only if no specialized solution is located. Inevitably, this leads to more solution loads at the model cold start, as the library may select an uniquely specialized solution for each individual layer with varying problem description.
Solution applicability. To ensure successful executions and correct results of DNN inference, the primitive library should only return *applicable* solutions for a given problem. The applicability checking process is time-consuming as it includes complicated logic such as workspace size checking, input format match, environment variable and hardware capability validation. In MIOpen, each solution implements the `IsApplicable` interface for the library to check whether it can be used to accomplish the computation for a given problem without any constraint violations (e.g., data type mismatch, out-of-bound memory access). Apparently, the incurred runtime overhead can be significantly alleviated if the applicable checks can be reduced to quickly identify a valid substitutive solution.

III. Design

In this section, we present the components and workflow of PASK. Fig. 5 illustrates the overview of our design, where PASK receives the lowered model as its input. During model parsing, PASK chooses to proactively load the solutions once they are parsed and starts the execution immediately once they are loaded (section III-A). For an absent solution, PASK checks the applicability of loaded solutions to seek the opportunities of reusing existing solution to accomplish the computation, thereby skipping to load the missing one (section III-B). Upon reusing, PASK integrates a categorical solution cache (section III-C) to organize existing loaded solutions and provides an interface to efficiently select an applicable solution for the given problem. Finally, we elaborate the implementation of PASK atop of open source inference engine and DL primitive library (section III-D).

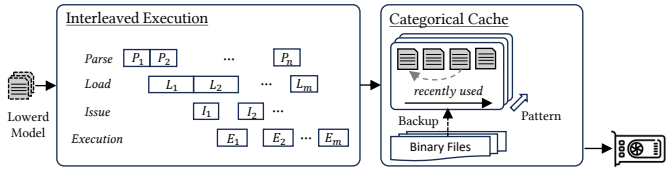


Fig. 5. Design overview. PASK receives the lowered model (a sequence of n solutions) as input, and then proactively parses, loads, issues the DNN layer to improve the GPU utilization via interleaved execution. Upon absent solution, PASK seeks the opportunities to reuse the loaded solutions and organizes them based on their patterns for efficient applicability checking.

A. Proactively Interleaved Execution

We introduce the interleaved execution to cope with the the inefficient execution hindered by the reactive launch and lazy loading pattern mentioned in section II-A. During parsing layers in DNN computation graph, PASK spawns two extra host threads to handle code loading and kernel issuing, respectively (Fig. 5). For a layer i, once it gets parsed (P_i, de-serialize the framework-specific model file to obtain the determined solution), the loading thread immediately starts to load the code objects of the desired solution (L_i). Once finish, the loading thread notifies the issuing thread to issue the kernels of the loaded solution (I_i) and execute them on GPU (E_i). This proactive design loads the necessary code objects in advance (once the parse finishes), therefore launches of the corresponding kernels incur no loading overhead. Besides, PASK starts the kernels at the earliest feasible time right after the loading. For comparison, current reactive serving frameworks start to launch the solutions only after all layers are parsed, and load the absent code objects on demand during launch. PASK mitigates the inefficiency with proactive interleaved execution, thereby achieving higher GPU utilization than traditional reactive inference engines.

As the DNN layer parsing is much faster than the code loading and execution, there will be a layer m where all the n layers are parsed and all preceding layers, including m, have finished their executions on GPUs (Fig. 5). We denote layer m as the *milestone* where PASK unconditionally loads the missing solutions into cache set $\mathcal{S}_m$ before m and selectively reuses the cached solutions after m. Note the minimal duration of a model execution is no less than the parsing process, hence there are free time slots for PASK to load before m solutions, which act as cached alternatives for the selective reuse in subsequent layers.

B. Selective Solution Reuse

After the milestone layer m, PASK quickens the execution of the remaining layers by avoiding unnecessary solution loads with selective adoption of cached solution. *Algorithm* 1 lists the solution reusing procedure where the ExecutionPlan function describes the layer-wise reuse decision procedure. When the target solution s_i^* ($i > m$) in layer i is parsed, PASK checks whether the corresponding

code object has been already loaded. If so, s_i^* will be directly used to complete the computation defined in layer i without incurring any loading overhead. Otherwise, PASK firstly attempts to acquire the applicable alternative solution by querying the cached solution set $\mathcal{S}_m$ (section III-C). Once a valid substitutive solution $\hat{s_i}$ exists, PASK will reuse it to accomplish the computation in layer i rather than loading the original absent solution s_i^* from scratch.

If, unfortunately, no valid solution can be found in the cached set $\mathcal{S}_m$, PASK has no choice but to load the desired solution s_i^*.

Algorithm 1 DNN execution with solution reuse

Input:
n: number of layers in the DNN model
m: the milestone layer (section III-A)
$\mathcal{S}_m$: solution cache set at layer m
p_i: problem to be solved in layer i
s_i^*: statically optimal solution in layer i

function EXECUTIONPLAN($\mathcal{S}, L, i$)
  for layer i from $m+1$ to n **do**
    if $s_i^* \in \mathcal{S}_m$ **then** ▷ s_i^* already loaded
      compute layer i with s_i^*
    else
      $\hat{s_i} \leftarrow$ GETSUBSOLUTION($\mathcal{S}, s_i^*, p_i$)
      if $\hat{s_i} \neq null$ **then** ▷ reusable solution found
        run layer i with substitutive solution $\hat{s_i}$
      else ▷ no applicable solution
        wait for s_i^* loading
        add s_i^* to $\mathcal{S}_m$
function GETSUBSOLUTION($\mathcal{S}, s^*, p$)
  list $\leftarrow \mathcal{S}[s^*.\text{pattern}]$
  for $s \in$ list's most recently used solution **do**
    if s.IsApplicable(p) **then**
      lru.update(s)
      return s
  return $null$

C. Categorical Solution Cache

In *Algorithm* 1, PASK queries the solution cache for a loaded solution to be reused for implementing the operator. As discussed in section II-B, it is expensive to lookup the applicability with current problem for every loaded solution in the cache. PASK thus adopts a categorical solution cache design to effectively find an applicable solution from all existing loaded ones.

In PASK, the loaded solutions are categorically organized in separated lists based on their *patterns* (section II-B). When a solution is loaded/invoked, it will be inserted/moved to the head of list. The GETSUBSOLUTION function in *Algorithm* 1 details how PASK handles the query for a substitutive solution given the desired solution s^* and problem p. It seeks for $\hat{s}$ from the list with the same pattern as s^* in the the most recently accessed (load or lookup) order. If one applicable solution is found in the categorical list, it is returned and moved to the list head (Fig. 5). Otherwise, the cache returns $null$, indicating there are no applicable solutions for current problem.

PASK adopts this categorical design to minimize the number of lookups (i.e., solution applicability checking for a problem) when searching for $\hat{s}$. When a specialized solution for problem p is missing, it is more likely to find a more general one with the same pattern to solve p (Fig. 4). If there are no applicable solution in one categorical list, PASK regards the query failed and returns $null$ directly to skip more lookups for other categories. We search the solutions in the recent access order similar to the LRU cache, as the neighboring layers in DNN models usually possess similar problem descriptions. Therefore, recently used solutions are more likely to be applicable to current problem.

D. Implementation

We implement PASK on the basis of the open source inference engine MIGraphX [45] and DL primitive library MIOpen [56] for AMD GPUs. PASK acts as a transparent middleware for end users.

We enhance MIGraphX to support the interleaved execution (section III-A). MIGraphX abstracts the computation graph by a list of instructions, which define the DNN operators and their corresponding input/output tensors. For interleaved execution, PASK spawns 3 host threads responsible for parsing, loading and issuing kernels, respectively (Fig. 5). We leverage single-producer-single-consumer (SPSC) channels to coordinate the interactions among host threads in MIGraphX. Instructions are fed to the loading thread when finishing the parse and leaves for the interleaved execution in the FIFO order. For each callee kernel of the solution, PASK invokes `hipModuleLoad` and `hipModuleGetFunction` (Fig. 3) to proactively load the target ELF sections from the binary files and determine location of the desired symbol by the kernel name.

We further augment MIOpen to support the solution reuse and loading skip in PASK. When the inference framework invokes `miopenRunSolution`, PASK follows *Algorithm* 1 to reuse the potentially applicable loaded solution for the computation in current layer. PASK implements the reuse logic for the common convolution, pooling and activation primitives in MIOpen.

TABLE I
EVALUATED DNN MODELS.

Model	# Primitive Layers	Type	Abbr.
`AlexNet` [1]	5	Img. Rec.	`alex`
`VGG16` [2]	16	Img. Rec.	`vgg`
`ResNet34` [3]	14	Img. Rec.	`res`
`RegNet_Y_800MF` [58]	28	Img. Rec.	`reg`
`EfficientNet_B7` [4]	58	Img. Rec.	`eff`
`Faster_R-CNN` [5]	16	Obj. Det.	`rcnn`
`SSD300` [59]	27	Obj. Det.	`ssd`
`FCN` [6]	18	Sem. Seg.	`fcn`
`UNet` [60]	37	Sem. Seg.	`unet`
`VIT_B_16` [7]	1	ViT	`vit`
`Swin_B` [8]	1	ViT	`swin`
`Swin_V2_B` [9]	1	ViT	`swin2`

IV. EXPERIMENT SETUP

Testbed. We evaluate PASK in a sever equipped with AMD EPYC 7773X CPU and AMD Instinct MI100 GPU, which possesses 32 GB VRAM and 120 Compute Units. The DNN inference server is implemented atop of MIGraphX and MIOpen as the graph optimization engine and DL primitive library backended by ROCm 6.0.2.

Workloads. We collect representative DNN workloads (Table I) from multiple fields (e.g., image recognition, object detection, semantic segmentation and vision transformer) implemented the PyTorch model zoo as served models in inference requests, where the vision transformer models are also used to emulate the transformer-based language models. Models are exported as ONNX format from PyTorch under the default configuration and then fed into MIGraphX for compute graph optimizations, and finally lowered to the corresponding GPU operations. The number of MIGraphX layers for each model is also listed in Table I. We follow the common image settings used in the ImageNet [57] dataset.

Evaluated schemes. To faithfully validate the effectiveness of PASK, we evaluate the above models under the following schemes:

- *Baseline*: the default inference workflow used by MIGraphX inference framework which loads every absent solutions.
- *NNV12*: the cold start mitigation design in NNV12 [38] to avoid input/output tensor layout transform.
- *Ideal*: hot execution of the DNN model, with all the desired solutions already resident.
- *PaSK*: full design of PASK, including interleaved execution and categorical solution cache.
- *PaSK-I*: PASK with only interleaved execution module being enabled, no solution reuse mechanism is enabled.

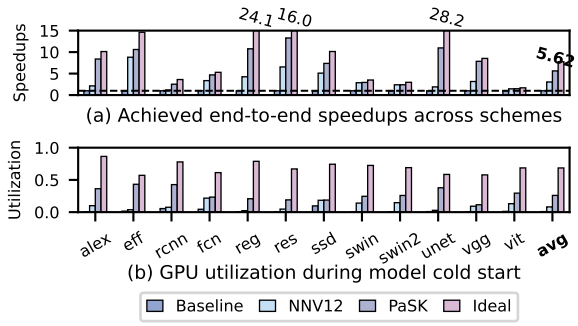


Fig. 6. (a) End-to-end cold start speedups for evaluated models under different schemes; (b) GPU utilization (fraction of time when GPU is active for computing) during model cold start.

- *PaSK-R*: PASK with solution reusing only, which exhaustively checks the applicability of every cached solution.

V. Results and Analysis

A. Cold Start Mitigation

We first investigate the achieved cold start mitigation effect under different schemes. Fig. 6(a) compares the end-to-end cold start speedups under *Baseline*, *NNV12*, *PaSK* and *Ideal* schemes, where *PaSK* achieves 5.62× speedups over *Baseline*. Compared to *Baseline*, *NNV12* possesses only moderate speedups (3.04×). *NNV12* mitigates the cold start by avoiding tensor layout transforms (e.g., NCHW → NHWC) between layers, limiting the optimize opportunities on specific layers. Furthermore, we can observe that *PaSK* performs better on models with more primitive layers (`eff`, `unet`, `reg` and `ssd`), as there are more opportunities for PASK to reuse the existing substitutive solution to replace the absent one to accomplish the computation. On the other hand, transformer models (`vit`, `swin` and `swin2`) perceive less end-to-end cold start latency improvement compared to other convolution-based models. As transformer models primarily comprise of GEMM operators from the BLAS library, there is only one primitive layer provided by MIOpen (Table I). Hence, there are less reuse chances as PASK could be invoked only when the primitive routine is called.

TABLE II
COLD START SPEEDUP WITH VARYING INFERENCE BATCH SIZES.

Batch Size	1	4	16	64	128
NNV12	3.04×	2.82×	2.49×	1.91×	1.74×
PaSK	5.62×	5.24×	4.55×	3.91×	3.10×
Ideal	7.75×	7.23×	6.94×	6.78×	6.41×

We further demonstrate the cold start improvements with varying batch size for each inference request in Table II. With enlarged batch size, GPU is better utilized and the proportion of duration consumed on code loading is thereby lower. Besides, hot execution is still far more efficient compared to cold start, while *PaSK* still outperforms *NNV12* with a large margin (210%).

B. Utilization and Breakdown

To understand the effectiveness of PASK, we adopt the hardware utilization (the proportion of time when GPU is active relative to the entire inference duration) as the metric to explain how PASK boosts the cold start performance of DNN models on GPU. Fig. 6(b) illustrates the achieved utilization under the various schemes, where we can capture that the GPU active time accounts for only a few portion (8.2%) of model execution for *NNV12*. As the majority of time is spent on kernel code loading, the cold start latency is several times higher than the one in hot start. *PaSK* avoids this substantial overhead by reusing the applicable solutions from rather than always loading the missing one, and effectively elevates the utilization to 25.9% on average. For *Ideal*, since all solutions are already cached, there is no loading overhead when launching the desired kernels onto the GPU thus the accomplished utilization is up-to 68.5% on average.

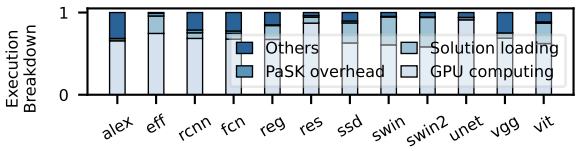


Fig. 7. Model cold start breakdown for *PaSK*.

Figure 7 further depicts the breakdowns of times consumed on different stages during model execution. We compare the ratios of times spent on GPU computing, solution loading, PASK overhead and others (e.g., host-device synchronization, model parse). The portion of time consumed on solution loading (11.2% on average) is acceptable compared to the cold start execution without dedicated management (Fig. 1). For transformer models (`vit`, `swin`, `swin2`), the solution loading proportion is relatively larger than other models, as the major operations are GEMMs. Moreover, the extra time incurred by PASK takes only 1.3% of the entire execution on average, indicating that PASK can efficiently retrieve the applicable alternative solution with negligible overhead.

C. Ablation Studies and Cache Statistics

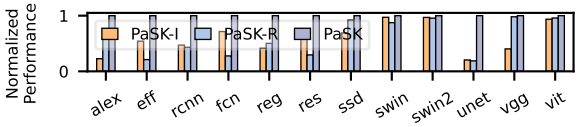


Fig. 8. Normalized performance of variants compared to *PaSK*.

In this section, we compare *PaSK* with its variants *PaSK-I* and *PaSK-R* to validate the effectiveness of solution reusing and categorical cache. For *PaSK-I*, the cold start latency reduction comes from the interleaved execution. Regarding *PaSK-R*, it adopts the naive cache which exhaustively searches all cached solution to find the optimal one, suffering from spending too much time in the applicability checking which is time-costly (section II-B). Fig. 8 illustrates the achieved proportional performance normalized to *PaSK*. For models perform worse in *PaSK-I* (`alex`, `vgg`, `unet`), we find the GPU execution durations before the milestone layer are short, thus there are less opportunities for *PaSK-I* to overlap the loading and execution process in these models. And we can only observe nuances of performance for transformer models, as they contain only one DL primitive operator (Table I) thereby possess few reuse opportunities.

To further investigate the impact of categorical cache, we collect the cache statistics during cold start. The cache hit rate describes the averaged possibility of successfully finding a applicable solution following the GETSUBSOLUTION routine in *Algorithm* 1. Fig. 9(a) shows the hit rate of categorical cache in PASK across models (69.7% on average) where transformer models are omitted as they contain only one primitive operator. We can capture that models comprising of more operators possess higher hit rates, as there are more chances for PASK to load the desired solution into the categorical cache following *Algorithm* 1. For subsequent queries, it is more likely to find a applicable solution in a cache with more entries.

Fig. 9(b) presents the averaged number of lookups per query (i.e., number of solutions whose applicability are checked), where the

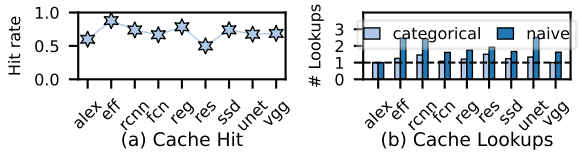


Fig. 9. Cache hit rate and averaged number of solution lookups for each hit.

categorical organization only uses 1.22 lookups per query while the naive searching requires 1.89 on average. As introduced in section II-B, checking the solution's applicability for a given problem is time-costly. Therefore, the naive solution searching strategy may result in severe performance downgrade due to enormous applicable checking overhead. It can be drawn from Fig. 9(b) that models with more lookups in the naive cache (`eff`, `unet`, `res`, `rcnn`) suffer from more pronounced performance downgrade in *PaSK-R* in Fig. 8. And the categorical organization in PASK can effectively reduce the number of required lookups per query, thus further boost the achieved performance.

VI. Discussions

Library supporting. As discussed earlier, the vision transformer models (`vit`, `swin`, `swin2`) possess less performance boosts from PASK because the major computation is matrix multiplication in the attention module [61]. Currently PASK is implemented atop of MIOpen, which contains the primitive operations used in DNN models (e.g., convolution, pooling), while the GEMM kernels are provided by the BLAS library (e.g., hipBLAS). Therefore, PASK can only identify and reuse a small portion of layers in the transformer models and offers limited cold start latency reduction for them. Note that it is trivial to extend PASK to support GEMM operators and accommodate with transformer models if more engineering efforts can be devoted to apply the similar modifications to hipBLAS. As hipBLAS and other vendor libraries (e.g. hipSPARSE [62] for sparse operators in GPU) following the same find-execution pattern (Fig. 3).

More factors for kernel specialization. Furthermore, PASK could be combined with other mechanisms by considering more factors. For instance, recent researches [63], [64] leverage the mix-precision components integrated in accelerators [65] to speedup the inference since DNN models usually have better tolerance on significance loss and numeric error. As the DNN operators are also specialized for tensor data types (`fp16`, `int8`), one may choose to still use high-precision data types if the corresponding kernels are already loaded while the low-precision ones are not. This offers the opportunities to accelerate the inference with high-precision computation by avoiding the overhead to load the absent kernels tailored for low-precision data types, even though the latter provide better performance on GPU. PASK can be generalized and applied to scenarios where the GPU kernels are specialized for some factors like input/output layout, data type, tensor tile size.

Loading desired solutions. Although PASK selectively skips the originally desired solutions to avoid the loading overhead, one can still load the skipped solutions during the intervals between two consecutive inference requests. Based on cloud workload traces [66], there are several seconds on average between the interval of two requests scheduled into the same inference instance. The interval duration is sufficient for PASK to proactively load the previously skipped solutions into the managed cache. Synergistically, they can act the candidates for reuse, or the desired solution which are already loaded for subsequent requests.

VII. Related Works

A. Cold Start Mitigation of DNN Models

Mitigating the cold start of DNN model has become the research appetite in recent years. Optimus [29] tackles the ML serverless container cold start problem by proposing the inter-function model transform approach which reuses operators from models of idle containers to compose the target model. Tetris [20] improves the serverless DNN inference latency and system container density via reusing duplicated tensors across instances. Pagurus [41] designs the algorithm to calculate the software dependency similarities between containers and then harness the idle container to build the intermediate image ready for helping to start the container rather than building from scratch. NNV12 [38] selectively loads the transformed weights in advance to avoid the tensor layout interchange overhead, and distributes the load and computation to different hardware units on edge/mobile devices.

These existing works share the container [29], [37], [41], service runtime [30], or model weights [20] among models. However, DNN models monolithic and cannot be transformed separately. Hence, some extra steps are needed to transform the shared resources to accommodate with the target model (e.g., modify the structure of warm models [29]). There are also general cold start mitigation approaches designed for cold microservice or serverless environments [37], [41], [67]–[70], but they fail to investigate the inference workflow and propose dedicated optimizations for DNN model invocations.

B. DNN Inference Optimizations

As the new computing paradigms thrive in recent years, cold start mitigation approaches have been proposed for preemptive spot [31], [33], [35], [41] computing and serverless computing [37]. For DNN model inference services under these two circumstances, the cold start process is inevitable as the serving instance has to be migrated across instances, or spawned in a new environment to satisfy the resource demands. PASK can be further integrated into the spot serving and serverless frameworks to mitigate the cold start issues with dedicated kernel loading management.

To harness the GPU capability for accelerating DNN computation, performance experts attempt to optimize operator graphs and model workflows to alleviate the memory bandwidth bottlenecks. TASO [25], Rammer [26], Cocktailer [27] and MonoNN [40] are graph transformation methods applying the intra- and inter-operator optimizations for the better hardware utilization. However, these approaches solely consider the kernel efficiency on GPU, and ignore the overhead of loading the multifarious generated kernels. PASK can be used to enhance these methods by assisting with some runtime information (e.g., the loaded kernels) to generate a more efficient execution plan for cold models.

VIII. Conclusion

In this work, we investigate the DNN model inference procedure and attribute cold start latency to the reactive scheduling strategy and kernel loading overhead. In order to mitigate the cold start latency, we propose PASK, a transparent middleware to schedule model kernels in a proactive strategy, and seek the opportunities to reuse existing loaded solutions organized categorically. We implement PASK on basis of the open source inference framework and primitive library and evaluate PASK with several popular DNN models. Detailed experiments demonstrate that PASK can effectively alleviate the cold start overhead for DNN model inference with 5.62× speedups compared to the default workflow used by the inference frameworks.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[3] Kaiming He et al. Deep residual learning for image recognition. In *CVPR*, 2016.

[4] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[5] Shaoqing Ren et al. Faster R-CNN: towards real-time object detection with region proposal networks. *PAMI*, 2017.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[7] Alexey Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[8] Ze Liu et al. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pages 9992–10002, 2021.

[9] Ze Liu et al. Swin transformer V2: scaling up capacity and resolution. In *CVPR*, 2022.

[10] Maxim Naumov et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv*, 2019.

[11] Jacob Devlin et al. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[12] Deliver high performance ML inference with AWS Inferentia. https://d1.awsstatic.com/events/reinvent/2019/REPEAT_1_Deliver_high_performance_ML_inference_with_AWS_Inferentia_CMP324-R1.pdf.

[13] Michael J. Anderson et al. First-generation inference accelerator deployment at facebook. *arXiv*, 2021.

[14] Ahsan Ali et al. Batch: machine learning inference serving on serverless platforms with adaptive batching. In *SC*, 2020.

[15] Pin Gao et al. Low latency RNN inference with cellular batching. In *EuroSys*, 2018.

[16] Gyeong-In Yu et al. Orca: A distributed serving system for transformer-based generative models. In *OSDI*, 2022.

[17] Chengliang Zhang et al. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *ATC*, 2019.

[18] Sohaib Ahmad et al. Proteus: A high-throughput inference-serving system with accuracy scaling. In *ASPLOS*, 2024.

[19] Daniel Mendoza, Francisco Romero, and Caroline Trippel. Model selection for latency-critical inference serving. In *EuroSys*, 2024.

[20] Jie Li et al. Tetris: Memory-efficient serverless inference through tensor sharing. In *ATC*, 2022.

[21] Yangyang Feng et al. Mobius: Fine tuning large-scale models on commodity GPU servers. In *ASPLOS*, 2023.

[22] Foteini Strati, Xianzhe Ma, and Ana Klimovic. Orion: Interference-aware, fine-grained GPU sharing for ML applications. In *EuroSys*, 2024.

[23] Kelvin K. W. Ng, Henri Maxime Demoulin, and Vincent Liu. Paella: Low-latency model serving with software-defined GPU scheduling. In *SOSP*, 2023.

[24] Tianqi Chen et al. TVM: an automated end-to-end optimizing compiler for deep learning. In *OSDI*, 2018.

[25] Zhihao Jia et .al. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *SOSP*, 2019.

[26] Lingxiao Ma et al. Rammer: Enabling holistic deep learning compiler optimizations with rtasks. In *OSDI*, 2020.

[27] Chen Zhang et al. Cocktailer: Analyzing and optimizing dynamic control flow in deep learning. In *OSDI*, 2023.

[28] Zhihao Bai et al. Pipeswitch: Fast pipelined context switching for deep learning applications. In *OSDI*, 2020.

[29] Zicong Hong et al. Optimus: Warming serverless ML inference via inter-function model transformation. In *EuroSys*, 2024.

[30] Misun Park, Ketan Bhardwaj, and Ada Gavrilovska. Pocket: ML serving from the edge. In *EuroSys*. ACM, 2023.

[31] Jiangfei Duan et al. Parcae: Proactive, liveput-optimized DNN training on preemptible instances. In *NSDI*, 2024.

[32] Xupeng Miao et al. Spotserve: Serving generative large language models on preemptible instances. In *ASPLOS*, 2024.

[33] Sanjith Athlur et al. Varuna: scalable, low-cost training of massive deep learning models. In *EuroSys*, 2022.

[34] Luo Mai et al. Kungfu: Making training in distributed machine learning adaptive. In *OSDI*, 2020.

[35] John Thorpe et al. Bamboo: Making preemptible instances resilient for affordable training of large dnns. In *NSDI*, 2023.

[36] Insu Jang et al. Oobleck: Resilient distributed training of large models using pipeline templates. In *SOSP*, 2023.

[37] Yanan Yang et al. Infless: a native serverless system for low-latency, high-throughput inference. In *ASPLOS*, 2022.

[38] Rongjie Yi et al. Boosting DNN cold inference on edge devices. In *MobiSys*, 2023.

[39] Kaihua Fu et al. Qos-aware irregular collaborative inference for improving throughput of DNN services. In *SC*, 2022.

[40] Donglin Zhuang et al. Mononn: Enabling a new monolithic optimization space for neural network inference tasks on modern gpu-centric architectures. In *OSDI*, 2024.

[41] Zijun Li et al. Help rather than recycle: Alleviating cold startup in serverless computing through inter-function container sharing. In *ATC*, 2022.

[42] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. Daydream: Executing dynamic scientific workflows on serverless platforms with hot starts. In *SC*, 2022.

[43] Hanfei Yu et al. Rainbowcake: Mitigating cold-starts in serverless with layer-wise container caching and sharing. In *ASPLOS*, 2024.

[44] NVIDIA TensorRT. https://developer.nvidia.com/tensorrt.

[45] AMD MIGraphX Documentation. https://rocm.docs.amd.com/projects/AMDMIGraphX/en/latest/.

[46] Size Zheng et al. Chimera: An analytical optimizing framework for effective compute-intensive operators fusion. In *HPCA*, 2023.

[47] Ganesh Bikshandi and Jay Shah. A case study in CUDA kernel fusion: Implementing flashattention-2 on NVIDIA hopper architecture using the CUTLASS library. *arXiv*, 2023.

[48] Ao Li et al. Automatic horizontal fusion for GPU kernels. In *CGO*, 2022.

[49] Introduction to torch.compile — PyTorch Tutorials. https://pytorch.org/tutorials/intermediate/torch_compile_tutorial.html.

[50] ONNX Operators - ONNX documentation. https://onnx.ai/onnx/operators/.

[51] Optimization — NVIDIA Triton Inference Server documentation. https://docs.nvidia.com/deeplearning/triton-inference-server/archives/triton_inference_server_1150/user-guide/docs/optimization.html.

[52] Using the performance database — MIOpen 3.2.0 Documentation. https://rocm.docs.amd.com/projects/MIOpen/en/latest/conceptual/perfdb.html.

[53] Model Repository — NVIDIA Triton Inference Server. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/model_repository.html.

[54] Nathan Otterness and James H. Anderson. Exploring AMD GPU scheduling details by experimenting with "worst practices". *Real Time Syst.*, 2022.

[55] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *CVPR*, 2016.

[56] MIOpen documentation. https://rocm.docs.amd.com/projects/MIOpen/en/latest/.

[57] Jia Deng et al. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[58] Ilija Radosavovic et al. Designing network design spaces. In *CVPR*, pages 10425–10433. Computer Vision Foundation / IEEE, 2020.

[59] Wei Liu et al. SSD: single shot multibox detector. In *ECCV*, 2016.

[60] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[61] Ashish Vaswani et al. Attention is all you need. In *NeurIPS*, 2017.

[62] hipSPARSE documentation. https://rocm.docs.amd.com/projects/hipSPARSE/en/latest/.

[63] Ji Lin et al. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In *MLSys*, 2024.

[64] Yilong Zhao et al. Atom: Low-bit quantization for efficient and accurate LLM serving. In *MLSys*, 2024.

[65] NVIDIA Tensor Cores: Versatility for HPC & AI. https://www.nvidia.com/en-us/data-center/tensor-cores/.

[66] Qizhen Weng. Beware of fragmentation: Scheduling gpu-sharing workloads with fragmentation gradient descent. In *ATC*, 2023.

[67] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. Icebreaker: warming serverless functions better with heterogeneity. In *ASPLOS*, 2022.

[68] Ashraf Mahgoub et al. ORION and the three rights: Sizing, bundling, and prewarming for serverless dags. In *OSDI*, 2022.

[69] Zijun Li, Quan Chen, and Minyi Guo. Pagurus: Eliminating cold startup in serverless computing with inter-action container sharing. *arXiv*, 2021.

[70] Dong Du et al. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *ASPLOS*, 2020.

Query-Based Black-Box Stealthy Sensor Attacks on Cyber-Physical Systems

Shixiong Jiang*
University of Notre Dame
sjiang5@nd.edu

Weizhe Xu*
University of Notre Dame
wxu3@nd.edu

Mengyu Liu
University of Notre Dame
mliu9@nd.edu

Fanxin Kong
University of Notre Dame
fkong@nd.edu

***Abstract*—We study the vulnerability of Cyber-physical systems (CPS) under stealthy sensor attacks in black-box scenarios. "Black-box" refers to scenarios where the attacker has minimal knowledge of the target system. Designing a stealthy sensor attack sequence under this scenario has two main challenges. The first one lies in ensuring the stealthiness of the sensor attack, meaning does not trigger an alert when applying the generated sensor attack sequence to the CPS. The second one is maintaining stealthiness throughout the attack generation process, indicating the limitation on the alarm frequency when generating the attack sequence. To address the above challenges, we develop a query-based black-box stealthy attack framework to violate the safety of the CPS. To maintain stealthiness during training, an active learning method has been introduced to extract the detector's information to a time series model. The stealthy attack sequence is then generated from that model. Experiments on four numerical simulations and a high-fidelity simulator demonstrate the effectiveness of the proposed framework.**

***Index Terms*—cyber-physical systems, vulnerability, stealthy, sensor attack, query, safety**

I. Introduction

Cyber-physical systems (CPS) integrate computing, networking, and physical systems via sensors and actuators. Widely used in applications like autonomous vehicles, drones, and power grids, CPS face significant security and safety challenges due to their complex environments, high connectivity, and long working periods. Sensor attacks are particularly severe, allowing attackers to manipulate sensor data and mislead controllers into issuing dangerous commands, such as manipulating the LiDAR sensor to steer a vehicle into obstacles [1] or crushing a drone by corrupting the values of an altitude sensor [2]. These emerging threats have spurred innovative research into sensor attack detectors [3]–[5]. The detectors trigger an alert by analyzing the difference between the predicted system states and system states from sensor readings.

Despite the presence of these detectors, some well-designed stealthy sensor attack methods can still achieve their objectives without triggering alarms. A stealthy sensor attack refers to the successful steering of a system into an unsafe zone while simultaneously avoiding the activation of detector alarms. These methods [6]–[8] often require significant knowledge of the system, such as system dynamics. However, having a thorough knowledge of the system is often unrealistic in practical applications. For example, attackers may find it hard to know the system model of a specific chemical plant. To fill this gap, we introduce the query-based method outlined in this paper for generating effective stealthy sensor attack sequences aimed at black-box CPS. In this context, "black-box" refers to scenarios where the attacker has minimal knowledge of the target system.

Designing a successful black-box stealthy sensor attack is promising yet challenging. The first challenge lies in ensuring the stealthiness of the sensor attack. Maintaining stealthiness of attack resides in devising an entire attack sequence that avoids triggering any alerts. Due to security and privacy considerations, CPS applications often obscure domain knowledge, including system dynamics and detector design, from external entities. Consequently, existing stealthy attack approaches that assume access to such knowledge are not applicable in this black-box scenario.

The second challenge is maintaining stealthiness throughout the attack generation process. This challenge implies that the process of obtaining the attack generator should not excessively trigger alarms, as system administrators may notice these additional alarms and modify the system or detector parameters, rendering the attack ineffective or exposing the attacker. Query is a widely used technique in black-box vulnerability analysis and attack synthesis. However, generating the attack through queries will inevitably trigger alarms. Reducing the number of alarms during the query process to ensure stealthiness during training poses a significant challenge.

To address these challenges, we propose a query-based black-box stealthy sensor attack framework to generate an efficient stealthy attack sequence while adhering to these constraints. We employ an active learning approach to generate attack sequences offline and utilize online queries to collect data. This enables the training of a neural network to replicate the system's detector, ultimately producing effective and stealthy attack sequences.

Our contributions are as follows. 1) We develop a framework to train a time series model that mimics a black-box system's detector behavior, capable of creating stealthy sensor attacks that significantly alter the system state. 2) Our method, based on active learning, efficiently gathers training data without triggering frequent alerts, maintaining training stealth. 3) We implement and evaluate our framework on various numerical and high-fidelity simulators, demonstrating its efficiency and effectiveness compared to existing methods.

II. Background and Related Works

In this section, we introduce some existing works on stealthy sensor attacks. The first group is conventional model-based approaches. The replay attack [9], a basic stealthy sensor attack, replaces real-time sensor data with previously recorded data. It simulates disturbance factors such as environmental noise, achieving a stealthy effect. However, it fails to effectively guide the system into the unsafe region. Covert attack [6], zero dynamic attack [7], and false data injection attacks [8] are all capable of performing stealthy sensor attacks. These methods typically generate a stealthy attack sequence offline, relying on complete knowledge of the system dynamics. However, such approaches necessitate an in-depth understanding of the system model, which significantly limits their applicability in real-world scenarios.

The second group uses data-driven methods to attack CPS. Studies like [10] and [11] target linear time-invariant (LTI) systems by learning the open-loop gain from control inputs, systems states, and

* denotes equal contribution.

some other information to generate attacks. Khazraei et al. [12] go further by designing a neural network-based attack generator to produce effective and covert sensor attacks. This is achieved by incorporating the detector's residuals and system deviation values as loss functions.

Although these learning-based methods can generate effective stealthy sensor attacks, they are ineffective in black-box scenarios, as they require access to internal system information such as control inputs and detector residuals. In contrast, our approach is designed for black-box settings, relying solely on sensor measurements and detector alarm status. These can be obtained through external observation: for sensor measurements, attackers can leverage their own additional sensors; for detector alarm status, the system's abnormal responses, such as recovery or reboot, often indicate an alarm has been triggered.

III. Preliminaries

A. System Model

The CPS model considered in the paper is a physical process governed by a discrete-time feedback controller. The controller operates at every time step. At the beginning of the time step t, the controller reads the state estimation $\hat{x}_{t|t-1}$ from the state estimator and uses a control algorithm to calculate a control input u_t. The control input u_t is then sent to the plant and drives the plant's evolution. Then the state estimator reads the sensor measurement y_{t+1} and gives the new state estimation $\hat{x}_{t+1|t}$. The system can be modeled as below:

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) + w_t \\ u_t &= \Omega(\hat{x}_{t|t-1}),\ y_t = h(x_t) + v_t \end{aligned}$$

where $x_i \in \mathbb{R}^n$, $u_t \in \mathbb{R}^m$ and $y_t \in \mathbb{R}^p$ are the real system state, control input, and sensor measurements. n, m and p denote the corresponding dimensions. $f : \mathbb{R}^m \to \mathbb{R}^n$ is the system dynamics, $\Omega : \mathbb{R}^n \to \mathbb{R}^m$ is the control algorithm, and $h : \mathbb{R}^n \to \mathbb{R}^p$ is the observation function.

B. Threat Model

We investigate the scenario of a black-box sensor attack, where an attacker can only access the sensor measurement and the alarm notifications. This assumption is valid, as attackers can measure accurate sensor values externally. Additionally, many systems visibly respond to detected attacks, such as by initiating recovery [13] or restarting [14].

In real-world scenarios, attacking multiple sensors simultaneously increases complexity and detection risk, especially in the black-box context of this paper. Thus, our approach assumes the attacker targets only one sensor at a time. Shown in the equation: $supp(a_t) = \{i \,|a_{ti} \neq 0\} \subseteq \{0, 1, ..., p\}$, where the support of attack vector $a_t \in \mathbb{R}^p$ at time step t is equal to 1, meaning only one sensor is under attack.

Under these sensor attacks, the state estimator will receive y_t^c as the sensor reading instead, as shown below:

$$y_t^c = y_t + a_t = h(x_t) + v_t + a_t$$

where a_t indicates the sensor attack at time step t. As a result, the estimated system state $\hat{x}_{t|t-1}$ may significantly deviate from the actual state x_t, leading to $\hat{x}_{t|t-1} \neq x_t$. Such discrepancies can prevent the controller from generating accurate control inputs, expressed as $\Omega(\hat{x}_{t|t-1}) \neq \Omega(x_t)$. Consequently, this may lead to the system state deviating to the unsafe area.

C. Ordinary Detector

We assume that the system is equipped with a residual-based detector, such as window-based [3], cumulative sum (CUSUM) [5], and χ^2 [4], which are three most commonly used detectors.

Regardless of the type of state estimator used in the system, the residual r_t, as shown in Equation (1), is calculated by subtracting the real sensor readings from the inferred sensor readings calculated by the state estimator.

$$r_t = \| \ y_t - h(\hat{x}_{t|t-1}) \ \| \tag{1}$$

1) Window-based detector: It is a short-memory detector that focuses on accumulating residuals within an observation window time. If the accumulated residuals r_t^{sum} exceed a predefined threshold τ, meaning $r_t^{sum} > \tau$, an alarm is triggered.

2) Cumulative sum (CUSUM): This approach enhances long-term memory for a window-based detector. It is predicated on the observation that the residual value, typically a small quantity associated with noise, remains low in the absence of an attack on the system. Unlike window-based detector, CUSUM lacks a fixed observation window w and updates s_t using the equation: $s_{t+1} = (s_t + |r_t| - b)^+$, where r_t is the current residual, drift b counters noise to prevent s_t from growing infinitely in the absence of attacks. An alert is triggered when $s_t > \tau$.

3) χ^2: This detector is also a long memory detector. χ^2 is based on the distribution of cardinality. It is depicted in equation: $g_t = r_t^T S_t^{-1} r_t$, where S_t is the covariance matrix of r, thus g_t is the weighted norm of r_t, with χ^2 distribution. The detector will raise an alert if g_t exceeds the threshold τ, indicating that the residuals' distribution has altered, possibly as a result of attacks.

D. Problem Statement

The attacker aims to perform effective and stealthy sensor attacks in black-box scenarios. Specifically, the goal is to generate an attack sequence targeting one sensor over a duration T, causing significant system deviation from reference values while avoiding detection and alarm triggers.

1) Effectiveness: The attacker seeks to generate an attack sequence that maximizes the impact on the target sensor, measured by state estimation errors, defined as:

$$\Delta x_t = x_t - \hat{x}_{t|t-1} \tag{2}$$

Before the sensor attack starts, the value of Δx_t is constrained by noise, and the real system state x_t is close to the reference value. Sensor attacks corrupt the data received by the estimator, causing significant deviations from the system's actual state and resulting in estimation errors. Furthermore, this leads the system to deviate from the reference.

2) Stealthiness: Regarding stealthiness, we require not only that the final attack sequence generated avoids triggering system alarms, but also that the process of creating the attack sequence through the algorithm minimizes excessive or frequent alarm triggers.

Specifically, the problem can be formulated as the following: Consider a CPS where the attacker has access only to sensor measurements and the detector's alarm status. The attacker generates an attack sequence targeting a single sensor. During the attack generation process, the attacker ensures that the detector alarm is not triggered frequently. Once the attack is initiated, the goal is to maximize state estimation errors while avoiding triggering the detector's alarm.

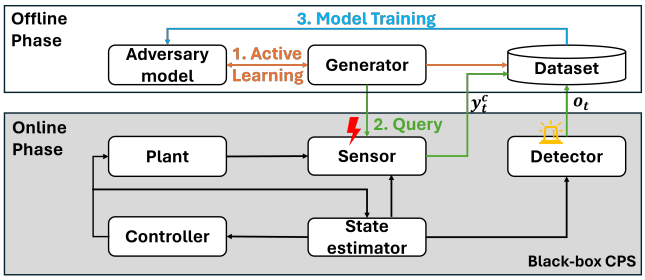


Fig. 1: The overview of the framework.

IV. Overview of the Framework

This section provides an overview of our query-based black-box stealth sensor attack framework, as shown in Fig. 1.

The grey box at the bottom represents the black-box CPS, whose internal information is inaccessible to us. We can only obtain sensor data y_t^c and the detector's result o_t. The white box at the top illustrates our query-based black-box stealth sensor attack framework. The adversary model is a time series model. It takes the sensor measurement y_t^c as the input and output alert or not o_t, allowing it to simulate the behavior of the detector. The generator is responsible for creating attack sequences based on our specifically designed active learning algorithm. The dataset is tasked with collecting data that is used to train the adversary model.

The orange, green, and blue arrows represent the three phases: active learning, querying, and model training. These phases are executed sequentially in a loop until a stealthy attack sequence meeting the requirements is generated. Active learning and model training occur offline, while the querying phase involves interaction with the CPS and operates online.

The process begins with the active learning phase, where the generator creates an initial attack sequence $a = \{a_0, a_1, ..., a_T\}$ of length T by interacting with the adversary model. In the query phase, this sequence is applied to the target system, producing sensor measurements y^c and detector alarm information o, which are collected into the dataset. A "one-step query" stops when the detector triggers an alarm, discontinuing any further attacks in the sequence. In the model training phase, the updated dataset, containing each attack sequence and corresponding alarm responses, is used to train the time-series model. The process then cycles back to active learning to generate a new attack sequence. This loop continues until a sequence of length T is produced that avoids triggering any alarms, resulting in the final stealthy attack sequence.

V. Query-based black-box stealthy sensor attack

This section outlines the process for generating a stealthy attack sequence targeting a black-box CPS. The first step is selecting a victim sensor, as the attacker in our threat model targets only one sensor at a time. The sensor with the highest potential to deviate the system from its reference should be chosen. A simple method is selecting the sensor most closely tied to the attack objective, such as targeting the distance sensor in a vehicular platooning system to cause collisions. Alternatively, a more refined approach involves testing the query-based attack method on each sensor and selecting the most effective one. For simplicity, we assume the appropriate sensor is pre-selected as the target.

A. Query

In this phase, a sensor attack sequence is applied to a stable system. The detector then identifies whether the sensors are maliciously perpetuated based on the ordinary detectors. The attacker collects the post-attack sensor measurements $y_t^c = y_i + a_i$ and alarm information o and compiles them into a set of training data.

When an alarm is triggered, the attacker halts the current attack and ends the one-step query. The query process concludes either when the detector raises an alarm or when the attack sequence reaches its end at T. For instance, if the query stops at step k, the training data includes $y^c = \{y_0^c, y_1^c, ..., y_k^c\}$, $y \in \mathbb{R}^{k*p}$ and the labels are $o = \{0, 0, ..., 1\}$, $o \in \mathbb{R}^k$. The attack sequence is then refined using an active learning algorithm, which is detailed later, and the refined sequence becomes the input for the next one-step query.

To maintain system stability, the next query is only initiated after the system returns to a stable state, allowing the attacker to control the interval between alarms and reduce detection risks. This flexibility improves stealth by spacing alarms and is especially useful during early training when initial attack sequences are more likely to trigger alerts.

B. Model Training

In this phase, we train the time series model to mimic the detector using the updated dataset.

Existing studies [10], [12] demonstrate that attackers can design stealthy attack sequences when they have access to the system model, detector parameters, and control signals. To reduce alarm triggers and minimize queries, having an approximate replica of the detector is crucial, as it allows for evaluating the likelihood of an alarm before launching a query.

The detector evaluates sensor readings across multiple time steps, forming correlated time series data, to decide whether to trigger an alarm. To replicate this behavior, we use a time series model, specifically a Long Short-Term Memory (LSTM) network in this paper. The LSTM model takes the attacked sensor readings y_t^c at time t as input and outputs the likelihood of the detector raising the alarm, shown in the equation below:

$$h_t = W_\theta(h_{t-1}, y_t^c), \ \hat{o}_t = \sigma(h_t)$$

where h_t denotes the hidden state calculated in the LSTM model at time t. The function σ is a mapping from h_t to $\hat{o}_t$, which is the alarm signal predicted by the current model.

Using the updated training dataset, the LSTM model learns the detector's behavior under various outputs and attack sequences. Since this is a binary classification task, the binary cross-entropy loss (BCEloss) [15] is used to calculate the loss.

At each training step, the LSTM model is trained on the previously collected dataset until it converges, ensuring $\hat{o}_t > 0.5$ if $o_t = 1$ and vice versa. This alignment ensures the model accurately reflects the detector's behavior in the real system. Consequently, the attack sequences generated using the LSTM model remain stealthy while achieving the maximum possible attack amplitude. Moreover, this approach enhances data utilization, which is especially valuable given the limited data available in this scenario.

C. Active Learning

In this phase, the generator generates the new attack sequence based on the LSTM model.

Attacking a system repeatedly to gather large amounts of data for training can significantly improve the LSTM model's performance. However, this is impractical under constraints on the number of alarms during training. To address this, we propose a generator based on an active learning algorithm to create new attack sequences using the trained LSTM model. By analyzing confidence scores, we identify

regions where the LSTM model is uncertain, guiding us to explore these areas to better simulate the detector's behavior. The details of

Algorithm 1 Active Learning

Input: y, a, β, k, a_{min}, a_{max}
Output: a_k
1: **while** $\beta > 0$ **do**
2: **for** $t = 0 : k$ **do**
3: $h_t = W_\theta(h_{t-1}, y_t^c)$
4: $\hat{o}_k = \sigma(h_k)$
5: **if** $ground(\hat{o}_k) == 0$ **then**
6: $a_{min} \leftarrow a_k$
7: **else**
8: $a_{max} \leftarrow a_k$
9: $a_k \leftarrow (a_{min} + a_{max})/2$
10: $\beta \leftarrow \beta - 1$

the active learning are summarized in Algorithm 1, which takes as inputs the time step k where the detector triggers an alert, sensor measurements, the partial attack sequence from step 0 to k, and the attacker-defined parameter a_{min}, a_{max}. The algorithm operates as follows: 1) Lines 2-4 are the LSTM inference process to get the output $\hat{o}_k$, the likelihood of an alert at step k. 2) Lines 5-9 update the a_k based on the output similar to binary search. If $\hat{o}_k < 0.5$, we change the a_{min} to a_k; If $\hat{o}_k > 0.5$, we set a_{max} to a_k. Then the new a_k is calculated from $(a_{min} + a_{max})/2$. 3) After β iterations, the updated a_k converges to a value near the boundary of the LSTM model's classification. This boundary value is used for subsequent queries as it represents data points where the model is most uncertain, making it highly valuable for refining the model.

The refined attack sequence a_t is generated based on the current LSTM model, minimizing the risk of triggering the detector during real-system queries. The active learning method further enables exploration near the LSTM model's decision boundary. Since the LSTM model is trained to approximate the system detector, the attack sequence performs similarly on both, ensuring stealthiness by avoiding alerts on the LSTM model. Additionally, the attack sequence is crafted using values near the detector's threshold at each time step, potentially maximizing Δx_t under the trained LSTM model, as per Equation (2). This aligns with our objective of pushing the system to deviate significantly without triggering alarms.

Although this method does not guarantee optimal attack performance, it achieves strong results in black-box scenarios, as the well-trained LSTM model closely mimics the real system detector. The active learning approach reduces the dataset size needed for LSTM model training, thereby minimizing alarm occurrences during data collection, and enhancing both the efficiency and stealth of the generator.

To summarise, the whole method is illustrated as algorithm 2. We first initialize a sequence of relatively large a and the LSTM model parameter θ. The T is the total length of the attack sequence which is defined by the attacker. The method works as follows: 1) Line 3-4: The attacker queries the system with the current attack sequence a; 2) Lines 5-8: Collect the data from time step 0 to k if the detector alerts at time step k; 3) Lines 9-10: Train the LSTM model until convergence and update the a_k using Algorithm 1; 4) Lines 11-13: Reset k to 0 if alarms, continue if not; 5) repeat the process after the attacker obtains a length of T attack sequence. By updating the attack duration steps, our LSTM model can effectively simulate the behavior of the detector in the actual system. The attack sequence

Algorithm 2 Query-based black-box stealthy attack algorithm

1: $Initialize\ \theta, T, a, \beta, a_{min}, a_{max}$
2: **for** $t = 0 : T$ **do**
3: **while** $k \leq t$ **do**
4: $a \rightarrow System$
5: **if** $alarm$ **then**
6: $y = \{y_0, y_1, ..., y_k\} \in \mathbb{R}^{p*k}$
7: $y^c = \{y_0 + a_0, y_1 + a_1, ..., y_k + a_k\} \in \mathbb{R}^{p*k}$
8: $o = \{o_0, o_1, ..., o_k\} \in \mathbb{R}^k$
9: $\theta \leftarrow LSTM\ update(\theta, y^c, o)$
10: $a_k \leftarrow Active\ Learn(y, a, \beta, k, a_{min}, a_{max})$
11: $k \leftarrow 0$
12: **else**
13: $k \leftarrow k + 1$

updated through this process can be utilized as a stealthy attack, causing the system to deviate from the reference without triggering an alert.

VI. Evaluation

We now compare the performance of our proposed stealthy attack strategy with two baselines. We test them on four numerical simulators and a high-fidelity simulator.

A. Numerical Simulation

To evaluate our algorithm and compare it with the baseline method, we consider several numerical CPS simulations: vehicle platoon, quadruple tank, DC motor speed, and aircraft pitch. We set $\beta = 25$ in all four benchmarks.

1) Simulator setting: **Vehicle platoon.** This system consists of four vehicles maintaining a stable 1-meter gap, forming a platoon, modeled after [16]. Each vehicle uses a lidar to measure the gap from the vehicle ahead. It also monitors the velocities of all vehicles. An attacker aims to reduce the gap between the front two vehicles and can only manipulate the distance sensor readings between them.

Quadrotor. The system model for controlling a four-rotor quadrotor is described in [17]. The input of the system is the total vertical thrust of the quadrotor, along with the three angular motions, The output of the system consists of six variables: The first three represent the relative position, while the last three correspond to the angles of pitch, yaw, and roll, respectively. In our scenario, we assume that the quadrotor is maintaining its altitude at 5 meters. An attacker manipulates the sensor measuring the altitude, causing it to drop.

DC Motor speed. The DC motor is an actuator to provide rotary motion and is usually coupled with wheels, drums and cables. The input of the system is the voltage source applied to the motor's armature, while the output is the rotational speed of the shaft. An attacker can manipulate the rotational speed sensor and the goal is to decrease the speed of the shaft.

Aircraft pitch. The aircraft pitch system [18] is an autopilot controlling the pitch angle of an aircraft. The system takes the elevator deflection angle as input and outputs the pitch angle. Assuming the yaw angle remains at 0 radians, the attacker aims to disrupt this stability by manipulating the pitch angle to deviate from its original value of zero.

2) Detector setting: For each benchmark, we employ both the CUSUM and χ^2 detectors. Process noise is modeled as zero-mean white noise. We tune the detectors to maintain a false alarm rate below 0.01 in the absence of attacks.

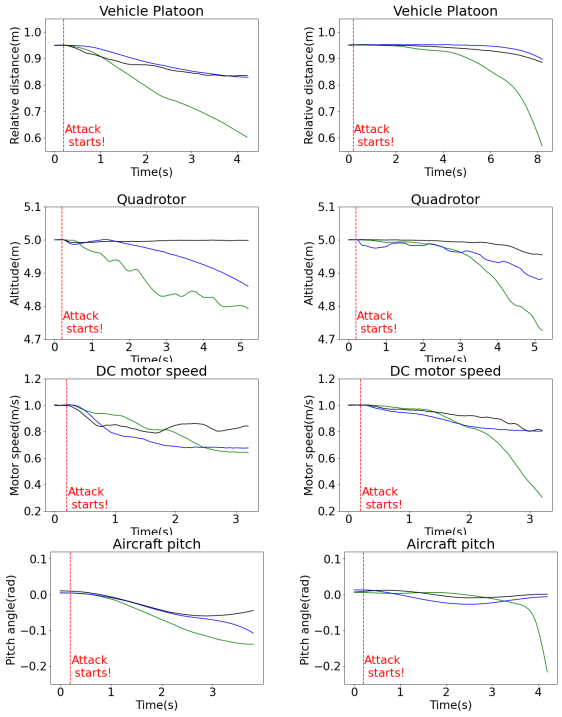


Fig. 2: Attack on each benchmark with CUSUM (left column) and χ^2 (right column) detector. The x-axis represents the time steps, while the y-axis represents the target system states. Green, blue, and black denote our query-based methods, the learning method, and random search methods, respectively.

3) Baseline setting: **Learning method.** We use the learning-based method [12] as a baseline. This method generates attack sequences using sensor measurements y_t and the detection function g_t during training. It also relies on N offline copies of the system, which is impractical in real-world scenarios. To adapt the baseline to our case, we set $N = 1$. **Random search method.** We introduce a second baseline method that uses a modified random search strategy to select the next attack sequence, as a purely random search is too inefficient. This method begins with the same initial value as our query-based approach and updates the attack sequence if it triggers an alarm, using the equation $a_t = random(a_t * 0.8, a_t)$, where the random function selects a value between $a_t * 0.8$ and a_t.

4) Attack effect analysis: We evaluated our query-based attack method against the two baseline methods in terms of attack effectiveness without imposing time constraints on the attacker. As shown in Fig. 2, our method drives all four benchmark systems away from their stable states more quickly and significantly than the baselines, demonstrating superior effectiveness. The results also reveal that with χ^2 detectors, the learning-based method often fails to consistently destabilize the system. Meanwhile, the random search method can generate attack sequences but performs the worst, as it lacks the capability to maximize the attack's impact. In contrast, our query-based black-box stealthy attack method significantly enhances the attack's impact compared to baselines.

5) Stealthiness analysis: In the experiments, all methods generate stealthy final attack sequences, making a comparison of final stealthiness irrelevant. Instead, we analyze stealthiness during the training process, focusing on two key aspects: 1) the number of queries and 2) the number of alarms triggered during training. Notably, the learning-based method leverages knowledge of the detector's residual, enabling it to avoid many alarm triggers during training. As a result, this method is excluded from the comparison in this subsection.

For the first aspect, we analyze how the number of queries varies with the length of the attack sequence. As shown in Table I, our method substantially reduces the number of queries compared to the random search method. This reduction highlights the effectiveness of the trained LSTM model in capturing the dynamic changes in the system's sensor values, enabling fewer queries and faster generation of the attack sequence within a shorter training period.

No. of queries: Query / Search	Detector	T = 50	T = 100	T = 150
Vehicle platoon	CUSUM	70 / 291	102 / 514	154 / 916
	Chi-square	111 / 533	190 / 988	293 / 1424
Quadrotor	CUSUM	95 / 373	298 / 671	526 / 998
	Chi-square	172 / 486	323 / 858	410 / 1257
DC motor speed	CUSUM	61 / 251	113 / 488	165 / 716
	Chi-square	103 / 356	262 / 581	489 / 752
Aircraft pitch	CUSUM	122 / 255	235 / 542	368 / 780
	Chi-square	116 / 405	216 / 865	269 / 1286

TABLE I: The number of queries to generate a certain length of attack sequence under different benchmarks and detectors using our query-based method and random search method.

For the second aspect, we ensure the system manager remains unaware of the attack by pausing the attack and loop processes when an alarm triggers during attack generation. The overall alarm rates for our method are shown in Table II. As can be observed, the attack remains highly stealthy during the training process.

B. CARLA

We implement the lane-keeping autonomous driving system [1] and on the high-fidelity simulator CARLA [19]. An attacker can manipulate GPS sensor readings, deviating the vehicle from its path and potentially crashing into other vehicles or obstacles. System variable e_d is used to measure the distance by which the vehicle deviates from the reference.

The results are shown in Fig. 3. The attacker generates an attack sequence for a duration of T=300 control steps. Prior to the start of the attack, the vehicle remains centered within the lane, with $e_d = 0$, and the residual is minimal. Once the attack commences, the vehicle progressively veers away from the lane without triggering any alarms. Ultimately, the vehicle collides with an off-road firetruck, but even then, the residual value fails to activate an alarm. This stealthy behavior indicates the success of the attack. Fig. 4 illustrates the process by which the attack impacts the vehicle. Note that the residual changes once the attack starts, this is inevitable because the χ^2 detector is sensitive to the noise changes. However, our attack does not trigger an alarm during the implementation phase.

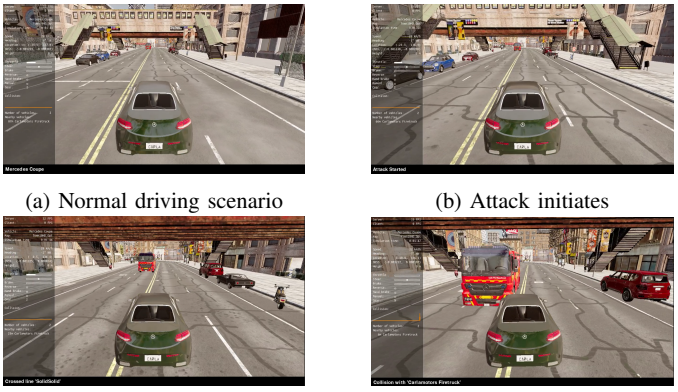
C. Discussion

One possible solution to defend against our proposed attacks is to block malicious users from gaining information about the detector by the query. As we mentioned before, the query process will inevitably generate alarms by the detector to get meaningful information. Defenders can utilize more sophisticated detectors, such as adaptive detector [20] and stealth attack detector [21].

The other direction to defend is protecting the system from being steered to non-secure areas under our proposed sensor attack

	Vehicle platoon	Quadrotor	DC motor speed	Aircraft pitch
Alarm rate	0.0036	0.0034	0.0035	0.0036

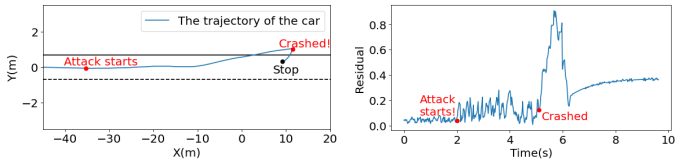
TABLE II: The alarm rate for each benchmark.



(a) Normal driving scenario (b) Attack initiates

(c) Vehicle deviates (d) Collision with firetruck

Fig. 3: The query-based attack on an autonomous vehicle.



(a) Normal state (b) Attack starts

Fig. 4: Illustrations of system states under normal and attack conditions in CARLA.

sequence. The simplest way to block sensor attacks is to leverage sensor redundancy. For example, in aircraft, there is usually more than one pitot tube to obtain airspeed. Some other methods, such as sensor fusion [22], well-designed state estimation [23], [24] can also be helpful.

VII. Conclusion

We propose a query-based black-box sensor attack method for CPS with a residual-based detector. Our approach ensures stealthiness both during the training process and throughout the attack execution. Extensive experiments have demonstrated the effectiveness of the proposed method.

Acknowledgment

This work was supported in part by NSF CNS-2333980. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation (NSF).

References

[1] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[2] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 881–896.

[3] I. Jovanov and M. Pajic, "Relaxing integrity requirements for attack-resilient cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 4843–4858, 2019.

[4] R. Tunga, C. Murguia, and J. Ruths, "Tuning windowed chi-squared detectors for sensor attacks," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 1752–1757.

[5] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[6] R. S. Smith, "Covert misappropriation of networked control systems: Presenting a feedback structure," *IEEE Control Systems Magazine*, vol. 35, no. 1, pp. 82–92, 2015.

[7] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson, "Revealing stealthy attacks in control systems," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 1806–1813.

[8] Y. Mo and B. Sinopoli, "False data injection attacks in control systems," in *Preprints of the 1st workshop on Secure Control Systems*, vol. 1, 2010.

[9] ——, "Secure control against replay attacks," in *2009 47th annual Allerton conference on communication, control, and computing (Allerton)*. IEEE, 2009, pp. 911–918.

[10] M. J. Khojasteh, A. Khina, M. Franceschetti, and T. Javidi, "Learning-based attacks in cyber-physical systems," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 1, pp. 437–449, 2021.

[11] A. Rangi, M. J. Khojasteh, and M. Franceschetti, "Learning based attacks in cyber physical systems: Exploration, detection, and control cost trade-offs," in *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, ser. Proceedings of Machine Learning Research, A. Jadbabaie, J. Lygeros, G. J. Pappas, P. A.nbsp;Parrilo, B. Recht, C. J. Tomlin, and M. N. Zeilinger, Eds., vol. 144. PMLR, 07 – 08 June 2021, pp. 879–892. [Online]. Available: https://proceedings.mlr.press/v144/rangi21a.html

[12] A. Khazraei, S. Hallyburton, Q. Gao, Y. Wang, and M. Pajic, "Learning-based vulnerability analysis of cyber-physical systems," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, 2022, pp. 259–269.

[13] L. Zhang, P. Lu, F. Kong, X. Chen, O. Sokolsky, and I. Lee, "Real-time attack-recovery for cyber-physical systems using linear-quadratic regulator," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[14] F. A. T. Abad, R. Mancuso, S. Bak, O. Dantsker, and M. Caccamo, "Reset-based recovery for real-time cyber-physical systems with temporal safety constraints," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–8.

[15] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on information theory*, vol. 42, no. 2, pp. 429–445, 1996.

[16] S. Dadras, R. M. Gerdes, and R. Sharma, "Vehicular platooning in an adversarial environment," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 167–178. [Online]. Available: https://doi.org/10.1145/2714576.2714619

[17] F. Sabatino, "Quadrotor control: modeling, nonlinearcontrol design, and simulation," 2015.

[18] D. Izci, S. Ekinci, A. Demirören, and J. Hedley, "Hho algorithm based pid controller design for aircraft pitch angle control system," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–6.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[20] F. Akowuah and F. Kong, "Real-time adaptive sensor attack detection in autonomous cyber-physical systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 237–250.

[21] M. Liu, L. Zhang, P. Lu, K. Sridhar, F. Kong, O. Sokolsky, and I. Lee, "Fail-safe: Securing cyber-physical systems against hidden sensor attacks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 240–252.

[22] J. Z. Sasiadek, "Sensor fusion," *Annual Reviews in Control*, vol. 26, no. 2, pp. 203–228, 2002.

[23] M. Pajic, J. Weimer, N. Bezzo, P. Tabuada, O. Sokolsky, I. Lee, and G. J. Pappas, "Robustness of attack-resilient state estimators," in *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2014, pp. 163–174.

[24] J. L. G. Bello and M. Kim, "A novel monocular disparity estimation network with domain transformation and ambiguity learning," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 474–478.

High-Performance Computing Architecture Exploration with Stage-Enhanced Bayesian Optimization

Vincent Fu[1,2], Mohamed Benazouz[1], Lilia Zaourar[1], Alix Munier-Kordon[2]
[1]*Université Paris-Saclay*, *CEA*, *List*, F-91120 Palaiseau, France
[2]*Sorbonne Université*, *CNRS*, *LIP6*, F-75005 Paris, France
{vincent.fu, mohamed.benazouz, lilia.zaourar}@cea.fr, alix.munier@lip6.fr

***Abstract*—The emergence of new applications in high-performance computing is driving the need for more efficient computing machines. As supercomputer architectures become increasingly complex, the combinatorial explosion of design spaces and the time-consuming nature of design simulations lead to challenging design space exploration problems. This work introduces an automated search framework to achieve power-performance-area efficient Arm Neoverse V1 processor designs. Based on multi-objective Bayesian optimization, we propose a new exploration algorithm named SEBO by enhancing the three main stages of the optimization. Experimental results show that SEBO can not only compete with the top state-of-the-art baseline algorithms, but also outperforms them in terms of the quality and diversity of the returned Pareto-optimal designs.**

***Index Terms*—High-performance computing, design space exploration, Bayesian optimization, ensemble learning**

I. Introduction

High-Performance Computing (HPC) is at the heart of recent advances in large-scale processing and modeling problems. From weather forecasting and physics simulations to machine learning and data analysis, these applications are becoming more and more computational intensive, leading to a sophistication of supercomputer architectures within the Multi-Processor System-on-Chip (MPSoC) design paradigm.

To ensure their efficiency before chip manufacturing, heavy simulation-based design is the state-of-the-art approach [5]. By running target applications on virtual prototypes, it gives designers access to key performance indicators (latency, power, etc.) of simulated architectures. The challenge is to figure out efficient architectural configurations by tuning through simulations, various hardware parameters such as core count, cache size, Network-on-Chip (NoC) topology, etc. However, due to the combinatorial explosion of design spaces and the time-consuming nature of design simulations, engineers must adopt a co-design approach by carefully selecting which architectural configurations to prioritize and for which target applications. Analytical formulations [32], [34] have been proposed as a simulation-substitute method for calculating performance metrics, but it requires confidential data from industry to correctly parameterize formulas and their coefficients. As a result, the choice of the final architectural configuration often relies on expert knowledge gained from engineering experience.

On the other hand, Design Space Exploration (DSE) [27] methodologies have shown impressive results for reaching more efficient designs [1], [22] than expert-made ones. Machine learning and operations research techniques provide powerful exploration algorithms to overcome high-dimensional multi-objective optimization difficulties. In this work, an automated design space exploration framework is proposed based on an advanced state-of-the-art Bayesian optimization algorithm for efficient 7nm Arm Neoverse V1 processor designs. Our contributions are as follows:

- The automated design space exploration framework uses VPSim [7], a SystemC transaction-level modeling (TLM) simulator, and a 7nm-supported McPAT [21], [38] modeler to compute Power-Performance-Area (PPA) estimations with a good simulation speed-accuracy trade-off.
- We propose a new exploration algorithm named SEBO, for Stage-Enhanced Bayesian Optimization, featuring a two-step Hammersley-transductive experimental design [37] model initialization, a trust region-based [12] ensemble surrogate modeling, and a batch acquisition function with parallel simulations. Each stage of the optimization is optimized to unveil an end-to-end efficient framework.
- The experimental results were validated for various standard HPC applications on large Arm Neoverse V1 architectural configurations with up to 128 cores. SEBO has been compared with the top state-of-the-art DSE baselines [1], [4], [25] and shows excellent performance in terms of the quality and diversity of the returned Pareto-optimal designs for a reasonable running time.

The paper is structured as follows. Section II discusses the state of the art of exploration algorithms. Section III depicts our proposed framework, followed by experimental results and observations in Section IV. Finally, Section V ends with conclusion and perspectives.

II. Related work

A. Model-free and model-based approaches

Exploration algorithms can be divided into two main classes: model-free and model-based approaches [27]. Understanding these is key to strengthening exploration by selecting the most adapted one to each workflow's specificities.

Early works focused on model-free approaches with algorithmic improvements. Platune [19], a platform-tuning environment, used Pareto local search with parameter clustering for efficient MIPS R3000 processor SoC design. By building a parameter dependency model, Platune enables exploration at

scale through design space reduction. The high-level synthesis DSE tool Cwbexplorer [29] has also adopted a clustering approach with simulated annealing. Despite the benefits of clustering, Palesi and Givargis [26] found that it may be insufficient to significantly reduce the design space and further improved Platune's scalability using the genetic algorithm SPEA2. However, since the model-free algorithms mentioned above heavily rely on heuristic functions, the exploration outcomes may be driven by some degree of intrinsic randomness. That means model-free algorithms are only efficient for high evaluation budgets. The greater the number of evaluations, the more quantitative information the randomness provides.

While evaluating a single configuration in Platune takes on average 1.5s, other frameworks like the VLSI flow of RISC-V Berkeley Out-of-Order Machine (BOOM) [1] can require 6 to 14 hours. In such cases, the randomness of model-free algorithms becomes undesirable, making sample-efficiency a critical feature to consider. In contrast, model-based approaches leverage machine learning models to capture the relationship between design parameters and objective functions. A typical example of that is reinforcement learning. Beltrame *et al.* [3] for instance, translated the domain knowledge of an MPSoC into a discrete-space Markov decision process and found the optimal exploration policy with value iteration. Lin *et al.* [22] went further using Monte-Carlo tree search to learn a deep policy network for efficient routerless NoC design. However, deriving optimal policies through reinforcement learning techniques is naturally sample-expensive, leading them to be more suitable for learning optimal control in applications like robotics or autonomous agents rather than for expensive simulation-based objective optimization.

B. Bayesian optimization: a sample-efficient methodology

As a similar learning-based alternative, Bayesian optimization [18] offers a more sample-efficient strategy. It sequentially employs two key components: the surrogate model and the acquisition function. After each design evaluation, the results are fed into the surrogate model to improve the objective function approximation. The acquisition function then leverages the surrogate model as a fully information-aware decision making tool to optimally suggest the next design to evaluate and so on. The sample-efficiency is therefore guaranteed by the surrogate model's ability to approximate the objective function precisely.

Among Bayesian optimization-based DSE frameworks, HyperMapper [25] proposed a random forest surrogate model for each objective function and combined with the expected improvement acquisition function using a randomized Chebyshev aggregation. Although random forests offer strong non-linear regression for discrete design spaces, their ability to model uncertainty as accurately as Gaussian processes, commonly used in Bayesian optimization, is debatable. In contrast, BOOM-Explorer [1] takes advantage of the latter and increases their modeling power through deep kernel learning. Moreover, since surrogate models need to be initialized, Bai *et al.* also implemented a clusterized transductive experimental design [37] as initial sampling for an informative warm start.

Despite the promising results of Bayesian optimization in DSE problems, most advanced Bayesian optimization techniques have been studied beyond hardware design. In the NeurIPS 2020 black-box optimization challenge [31], participants were asked to implement an efficient algorithm that outperforms the strongest baseline TuRBO [16], a new state-of-the-art Bayesian optimization algorithm, for various machine learning hyperparameter tuning tasks. The main conclusions of the challenge suggest that, first, Bayesian optimization was consistently effective. Not only did less-efficient Bayesian optimization-based baselines outperform genetic algorithms, but also all of the top proposed methods were Bayesian optimization techniques. Secondly, to outperform TuRBO, top participants naturally employed its main feature: using trust regions for local Bayesian optimization and incremented, on top of that, multiple surrogate and acquisition models as ensemble learning. Considering the no free lunch theorems [35], leveraging multiple surrogate models can further improve the approximation of objective functions.

To illustrate how the fit of surrogate models impacts exploration, Dreczkowski *et al.* [15] conducted a comparative study on discrete kernel-based Gaussian processes for combinatorial Bayesian optimization. They found that some kernels, such as the string subsequence [20], the overlap, and the Hamming embedding via dictionary [13], provide better model fitting than others across various benchmarking tasks. Furthermore, they also observed that trust region-based methods consistently outperform the non-trust region ones, confirming the efficiency of local modeling for any problem instances.

Assuming hardware design optimization deals with discrete parameters, leveraging trust regions and ensemble learning with efficient discrete space-tailored surrogate models is key to achieving a state-of-the-art Bayesian optimization-based DSE.

III. Automated Exploration framework

Fig. 1 shows our proposed framework and provides an overview of how an automated model-based DSE works. By alternating between design suggestions and simulation feedback, the exploration model progressively accumulates relevant information, which further enhances this alternating mechanism, driving the convergence toward efficient designs. Known as active learning, the way in which the model makes suggestions and updates itself with new information is central to exploration performance.

A. Multi-objective optimization problem

In hardware design, the quality of an architecture is measured through different key performance indicators, such as energy consumption, latency, and area. Since the latter are negatively correlated, the ideal design, which is the one that would have the best value for each metric at the same time, is in reality, not feasible. Trying to optimize one objective comes at the cost of degrading another. While in single-objective optimization, the optimality is trivially deduced from the total order of real numbers, searching optimal design in multi-

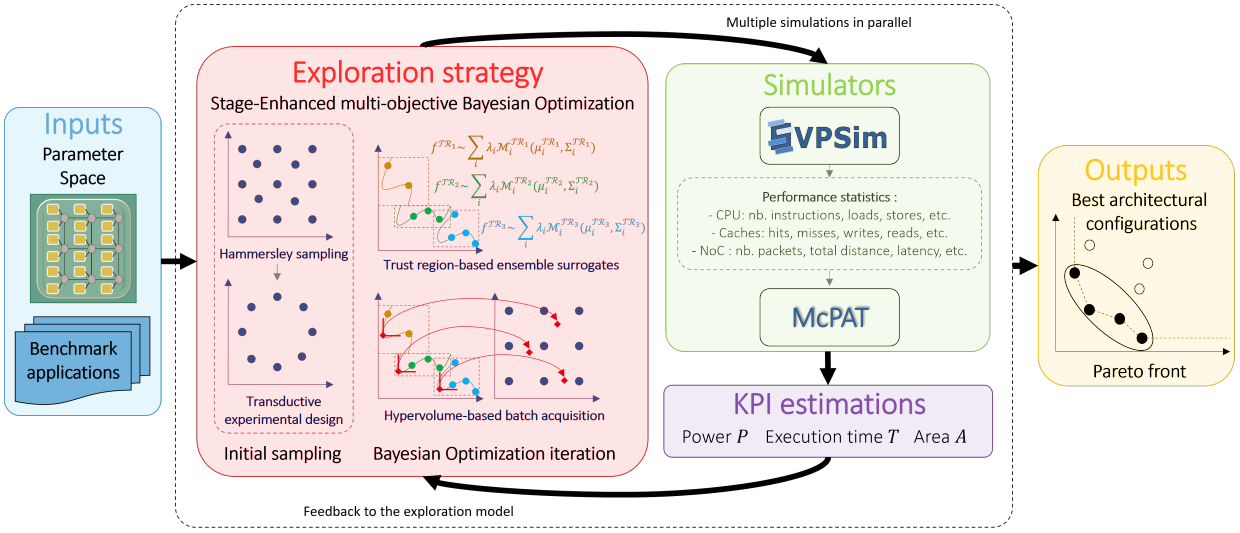


Fig. 1: Overview of proposed automated design space exploration framework

objective optimization comes down to making objective trade-offs defined by the Pareto optimality.

Definition 1 (Pareto optimality)**.** *Let $\mathcal{X}$ be the design space and $M \geq 2$. Given a M-dimensional multi-objective function $F(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})]^\top$, a solution $\mathbf{x} \in \mathcal{X}$ is said to be dominated by $\mathbf{y} \in \mathcal{X}$, denoted as $\mathbf{x} \preceq \mathbf{y}$, in minimization (resp. maximization) problem if and only if:*

$$\begin{aligned} &\forall\, p \in [\![1, M]\!], \quad f_p(\mathbf{y}) \leq f_p(\mathbf{x}) \quad (\textit{resp. } f_p(\mathbf{y}) \geq f_p(\mathbf{x})) \\ &\exists\, q \in [\![1, M]\!], \quad f_q(\mathbf{y}) < f_q(\mathbf{x}) \quad (\textit{resp. } f_q(\mathbf{y}) > f_q(\mathbf{x})) \end{aligned} \tag{1}$$

A solution $\mathbf{x}$ is Pareto-optimal if and only if no other solution dominates it (i.e. $\nexists\, \mathbf{y} \in \mathcal{X}, \mathbf{x} \preceq \mathbf{y}$).

Since different objective trade-offs are possible (*e.g.*, a design can be more latency efficient at the cost of higher energy consumption, or *vice versa*), the goal of multi-objective optimization is to return all feasible Pareto-optimal solutions.

Therefore, achieving PPA-efficient hardware designs involves multi-objective optimization. Given a design space $\mathcal{X}$, we want to retrieve the Pareto set $\mathcal{X}_P$ by simultaneously minimizing the energy consumption e, the execution time t and the area a given a target application γ:

$$\min_{\mathbf{x} \in \mathcal{X}} F(\mathbf{x}) = [e(\mathbf{x}, \gamma), t(\mathbf{x}, \gamma), a(\mathbf{x})]^\top \tag{2}$$

$$\mathcal{X}_P = \{\mathbf{x} \in \mathcal{X} \mid \nexists\, \mathbf{y} \in \mathcal{X}, \mathbf{x} \preceq \mathbf{y}\} \tag{3}$$

In our problem instance, a design $\mathbf{x}$ is in the design space $\mathcal{X}$ if its architectural parameters are among candidate values defined in TABLE I and satisfy hardware constraints.

As analytical formulations are not exploited here, the multi-objective values $F(\mathbf{x})$ are obtained through design simulations using various electronic design automation tools. In particular, we leverage VPSim and an enhanced version of McPAT [38] to build our simulation flow for Arm processor design.

B. VPSim simulator and McPAT modeler

VPSim [7] is a SystemC TLM-based virtual prototyping tool dedicated to hardware-software co-design. Based on the emulation platform QEMU [2] augmented with fast memory hierarchy models, VPSim enables running software applications on simulated architectures for various processor models including Arm cores. With a good balance between simulation speed and accuracy, VPSim not only estimates application execution time but also provides performance statistics such as the number of instructions, cache reads/writes, hits/misses, etc. However, since it computes only one of the objectives, we use McPAT to handle the remaining objective computations.

TABLE I: System-level design space of Arm Neoverse V1

Architectural parameters	Candidate values
Number of cores n_c	2, 4, 8, 16, 32, 64, 128
Number of cores per cluster n_q	1, 2, 4, 8, 16
L1 data/instruction cache size (KB) s_1	16, 32, 64, 128
L2 cache size (KB) s_2	256, 512, 1024, 2048
L3 cache size (KB) s_3	512, 1024, 2048, 4096, 8192
Cache block size (B) b	64, 128
L1 cache associativity a_1	2, 4
L2 cache associativity a_2	4, 8, 16
L3 cache associativity a_3	4, 8, 16, 32
Mesh NoC XY dimensions d_X, d_Y	1, 2, 4, 8, 16

McPAT [21] is a power and area modeling framework for many-core architectures. Built on top of CACTI [24], McPAT follows the same analytical approach to extend power and area evaluations for between 90nm to 22nm technology node-based MPSoC. Since McPAT needs to be combined with a performance simulator to function, the coupling of McPAT and VPSim is perfectly matched to form a complete PPA evaluation flow.

Furthermore, to support 7nm design, we draw inspiration from McPAT-Calib [38] by integrating 7nm FinFET technology [8] scaling parameters into McPAT's analytical models. Whereas McPAT-Calib leveraged on top of that, already simulated architecture data to train McPAT's models for better accuracy, we assume here to be design-agnostic, and to not use any prior simulation information before the start of DSE.

To automate DSE efficiently, we propose our own exploration algorithm named SEBO, based on improvements at

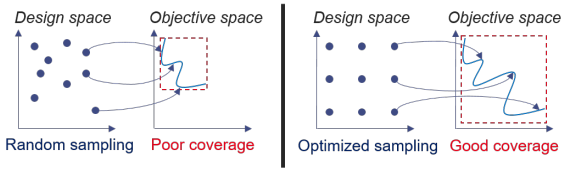


Fig. 2: Example of efficient sampling

three main stages of Bayesian optimization.

C. Stage-enhanced multi-objective Bayesian optimization

SEBO is a model-based algorithm based on Bayesian optimization principle [18] for multi-objective optimization. As SEBO employs surrogate models, they must first be initialized by training on an initial set of solutions with their corresponding labels (their objective values). In our problem, it means choosing initial architectural configurations and simulating them. This can be considered as the first stage of DSE.

A common method for selecting initial solutions is random sampling. Despite its simplicity, the stochasticity of random sampling can lead to a poor quality of initial solution set, ultimately biasing surrogate models by feeding them with undiversified information. Fig. 2 illustrates this impact.

In order to achieve an informative model initialization, we introduce the two-step Hammersley-transductive experimental design sampling as our first improvement. To understand the logic behind this, let's consider the ridge regression problem.

Definition 2 (Ridge regression)**.** *Given n training samples $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^\top$, $\mathbf{y} = [y_1, y_2, \dots, y_n]^\top$ where $(\mathbf{x}_i, y_i) \in \mathbb{R}^N \times \mathbb{R}$ for $i = 1, \dots, n$, small $\lambda > 0$ and $\|\cdot\|$ the Euclidean norm, Ridge regression consists of solving:*

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathbb{R}^N}{\arg\min} \|\mathbf{y} - X\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2 \tag{4}$$

Trivially, $\hat{\mathbf{w}} = \left(X^\top X + \lambda I\right)^{-1} X^\top \mathbf{y}$ with identity matrix I. Considering regression is not exact due to the potential influence of unseen factors, the labels $\mathbf{y}$ are commonly modeled as $y_i = \mathbf{w}^\top \mathbf{x}_i + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$ for $i = 1, \dots, n$. Then, the estimation error $\hat{\mathbf{w}} - \mathbf{w}$ has (with the zero vector denoted as $\mathbf{0}$ and $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \dots, \epsilon_n]^\top$):

$$\begin{aligned} \mathbb{E}\left[\hat{\mathbf{w}} - \mathbf{w}\right] &= \left(X^\top X + \lambda I\right)^{-1} X^\top \mathbb{E}\left[\boldsymbol{\epsilon}\right] = \mathbf{0} \\ \mathrm{Var}\left[\hat{\mathbf{w}} - \mathbf{w}\right] &= \mathrm{Var}\left[\left(X^\top X + \lambda I\right)^{-1} X^\top \boldsymbol{\epsilon}\right] \\ &= \sigma^2 \left(X^\top X + \lambda I\right)^{-1} X^\top X \left(X^\top X + \lambda I\right)^{-1} \end{aligned} \tag{5}$$

If the regression is exact, $\mathrm{Var}\left[\hat{\mathbf{w}} - \mathbf{w}\right]$ should be equal to $\mathbf{0}$. $\mathrm{Var}\left[\hat{\mathbf{w}} - \mathbf{w}\right]$ is therefore the metric which quantifies the amount of information that regression models cannot capture. Suppose we can now select our n training samples from the design space $\mathcal{X}$. Let V be the matrix composed of all designs $\mathbf{v} \in \mathcal{X}$. Selecting the most informative samples (for linear models) is equivalent to "minimize":

$$\mathrm{Var}\left[V\hat{\mathbf{w}} - V\mathbf{w}\right] = V\,\mathrm{Var}\left[\hat{\mathbf{w}} - \mathbf{w}\right] V^\top = \sigma^2 C C^\top \tag{6}$$

where $C = VX^\top \left(XX^\top + \lambda I\right)^{-1}$ with the push-through identity. As we can notice, "minimizing" this variance doesn't require any labels $\mathbf{y}$. This means we can select the most informative design solutions for our model initialization without simulating them. The kernel trick [30] is then applied within the regression framework to handle non-linear models, hence, defining the transductive experimental design problem (note that our formulation differs from the original paper [37]).

Definition 3 (Transductive experimental design)**.** *Let $\mathcal{H}$ be a reproducing kernel Hilbert space with a kernel function $k(\mathbf{u}, \mathbf{z}) = \langle \phi(\mathbf{u}), \phi(\mathbf{z}) \rangle_{\mathcal{H}}$ where $\mathbf{u}, \mathbf{z} \in \mathbb{R}^N$ and $\phi : \mathbb{R}^N \to \mathcal{H}$ is a feature mapping. Supposing $\|\cdot\|_F$ is the Frobenius norm and $f \in \mathcal{H}$ has the form $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$, the optimal transductive experimental design is solution of, with matrix notation $K_{VX} = \left(k(\mathbf{v}_i, \mathbf{x}_j)\right)_{1 \le i \le s,\, 1 \le j \le n}$:*

$$\begin{aligned} &\min_X \left\| K_{VX} \left(K_{XX} + \lambda I\right)^{-1} \right\|_F^2 \\ &\textit{s.t. } \{\mathbf{x}_i\}_{i=1}^n \subseteq \{\mathbf{v}_i\}_{i=1}^s \end{aligned} \tag{7}$$

Since solving transductive experimental design over the entire design space is NP-hard due to combinatorial explosion, we leverage Hammersley sampling [9], [36] to build a low-discrepancy representative subset of the design space for V. We then solve transductive experimental design on it, using a scalable genetic algorithm. While BOOM-Explorer [1] relies on architecture-specific prior knowledge for pre-sampling its transductive experimental design, Hammersley sampling, by contrast, doesn't require any prior information about interactions between design space variables to sample, making it adapted for all DSE problem instances. Thus, our two-step Hammerlsey-transductive experimental design sampling forms the initial set of designs to be explored.

Given this first stage improvement, our second involves proposing accurate and expressive surrogate models. An efficient approach to achieve that is to employ Gaussian processes (GP) [28] for each objective. GP are non-parametric models that can be viewed as multivariate normal distributions over functions. In other words, $f \sim \mathcal{GP}(m, k)$ if for any finite collection of function values $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2) \dots, f(\mathbf{x}_n)]^\top$:

$$\mathbf{f} \sim \mathcal{N}\left(\left[m(\mathbf{x}_1), \dots, m(\mathbf{x}_n)\right]^\top, K_{XX}\right) \tag{8}$$

where m and k are the mean and variance functions. This definition implies that any $f(\mathbf{x}_i)$ is modeled by a Gaussian distribution, where the density function represents the plausibility of $f(\mathbf{x}_i)$ to be equal to the corresponding value from the support. By setting m and k correctly, these plausibilities which characterize our beliefs about the objective space in relation to the design space, can be parametrized, enabling machine learning process. Assuming simulations may be noisy, let $m(\mathbf{x})$ be associated with its observed value y as the empirical mean and k be a kernel function with hyperparameters $\boldsymbol{\theta}$, the objective values of an unexplored design $\mathbf{x}_* \in \mathcal{X}$ can be predicted by computing the posterior probability distribution $f(\mathbf{x}_*) \,|\, \mathbf{x}_*, X, \mathbf{y}$ using Bayes' rule with the joint distribution:

$$\begin{bmatrix} f(\mathbf{x}_*) \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}_*) \\ \mathbf{y} \end{bmatrix}, \begin{bmatrix} k_{\boldsymbol{\theta}}(\mathbf{x}_*, \mathbf{x}_*) & K_{\mathbf{x}_* X | \boldsymbol{\theta}} \\ K_{X \mathbf{x}_* | \boldsymbol{\theta}} & K_{XX|\boldsymbol{\theta}} + \sigma^2 I \end{bmatrix}\right) \tag{9}$$

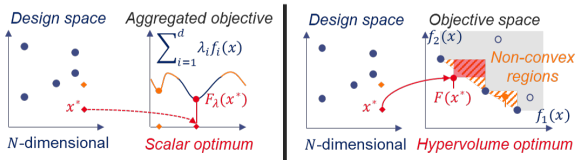


Fig. 3: Omission of non-supported Pareto-optimal solutions

where $\boldsymbol{\theta}$ is tuned by maximizing the marginal likelihood. As their expressiveness depends on the kernel function, the choice of the latter becomes critical.

To mitigate the risk of choosing an unsuitable kernel, we bring ensemble learning by selecting the most appropriate kernel from the power set of different linear combinations of discrete space-tailored kernels at each iteration of Bayesian optimization and more importantly, for each objective function. In particular, we leverage the overlap k^{O}, the transformed overlap k^{TO} [33] and the Hamming embedding via dictionary k^{HED} as our base kernels to construct our ensemble $\mathcal{K}$:

$$\mathcal{K} = \left\{ \sum_{k \in \mathcal{E}} k \;\middle|\; \mathcal{E} \in \mathcal{P}(\{k^{\text{O}}, k^{\text{TO}}, k^{\text{HED}}\}) \setminus \{\emptyset\} \right\} \tag{10}$$

$$k^{\text{O}}(\mathbf{u}, \mathbf{v}) = \frac{\sigma}{N} \sum_{i=1}^{N} \ell_i \delta(\mathbf{u}_i, \mathbf{v}_i) \quad k^{\text{TO}} = \exp\left(\frac{k^{\text{O}}}{\sigma}\right) \tag{11}$$

$$k^{\text{HED}}(\mathbf{u}, \mathbf{v}) = k_{\text{M}}^{5/2}\left(\phi_A(\mathbf{u}), \phi_A(\mathbf{v})\right) \tag{12}$$

where δ is the Kronecker delta, $k_{\text{M}}^{5/2}$ the Matérn 5/2 kernel, ϕ_A the embedding based on the dictionary A [13], ℓ_i are the lengthscales and the hyperparameters $\boldsymbol{\theta} = (\sigma, \ell_1, \ldots, \ell_N)$. Given a random test set $\mathcal{T}$ from already simulated designs and the density functions $\varphi_\cdot$, the optimal kernel $k_{\boldsymbol{\theta}} \in \mathcal{K}$ is selected by maximizing the sum of log predictive densities through leave-one-out cross validation:

$$k_{\boldsymbol{\theta}} = \underset{k \in \mathcal{K}}{\arg\max} \sum_{(\mathbf{x}_*, y_*) \in \mathcal{T}} \log\left(\varphi_{f(\mathbf{x}_*) \mid X, \mathbf{y} \setminus \{(\mathbf{x}_*, y_*)\}}(y_*)\right) \tag{13}$$

Although ensemble learning can improve model accuracy, it incurs a higher learning cost. To counter that, we leverage local Bayesian optimization by dividing the search space into trust regions [12], [16] to form several localized surrogate models trained on smaller datasets for faster multi-ensemble learning. Moreover, since each surrogate model is independent, we take advantage of training parallelization to accelerate our algorithm even further. For the trust region setup policy, we adopt the same policy from the original paper MORBO [12].

Let's now focus on our third stage improvement, which concerns sampling designs to be simulated using the acquisition function. Although there are popular ones [18] such as expected improvement and entropy search, all of them are single surrogate-based, making them unpractical for multi-objective Bayesian optimization. A common approach to break it down is to employ scalarization methods by aggregating all surrogate models into a single one. Nevertheless, optimizing an aggregated function such as a weighted linear aggregation can omit non-supported Pareto-optimal solutions in non-convex search space regions as illustrated in orange in Fig. 3.

To address this, we instead preserve the multidimensionality of multi-objective functions, and exploit hypervolume-based acquisition functions [10]. In particular, we leverage the noisy expected hypervolume improvement acquisition function [11] to even account the uncertainty in the Pareto front.

Definition 4 (Noisy expected hypervolume improvement)**.** *Given $\mathbf{r} \in \mathbb{R}^M$ a reference point, $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ the current set of observations, $\mathcal{Y}_P^{\mathcal{D}}$ the Pareto front of $\mathcal{D}$ in objective space, noisy expected hypervolume improvement is defined as:*

$$\alpha_{\text{NEHVI}}(\mathbf{x}) = \int \alpha_{\text{EHVI}}(\mathbf{x} | \mathcal{Y}_P^{\mathcal{D}}) p(f|\mathcal{D}) df \tag{14}$$

where α_{EHVI} is the expected hypervolume improvement [10] *and $p(f|\mathcal{D})$ the joint distribution of the Gaussian process f on $\mathcal{D}$.*

Since (14) is analytically untractable, Daulton *et al.* proposed Monte-Carlo integration to approximate it. Maximizing the acquisition function therefore provides the most promising design according to the joint surrogate model. As the acquisition function can only be computed pointwisely, we use multi-start local search to determine our next design to evaluate. We perform this for each trust region-based joint model, all in parallel, as a batch acquisition to support multi-simulations.

With all the aforementioned features, SEBO is set for efficient automated DSE and can be interfaced with any electronic design automation tools. Its algorithm is outlined below.

Algorithm 1 SEBO($\mathcal{X}$, s, n, n_{TR})

1 $V \leftarrow$ sample s Hammersley designs in $\mathcal{X}$ [36]
2 $X \leftarrow$ solve *transductiveExperimentalDesign*(V, n) *eq.* (7)
3 $Y \leftarrow$ *simulate*(X) **in parallel**
4 Initialize trust regions using procedure [12] Section 3.2
5 **while** simulation budget not exhausted **do**
6 **for** trust region $\mathcal{TR}_{i \in [\![1, n_{\text{TR}}]\!]}$ **do in parallel**
7 $\mathcal{T}_i \leftarrow$ build random test set in $\mathcal{TR}_i$
8 **for** objective $j \in [\![1, M]\!]$ **do in parallel**
9 $k_{\boldsymbol{\theta}} \leftarrow$ find best kernel with $\mathcal{T} = \mathcal{T}_i$ *eq.* (13)
10 Fit $f_j^{\mathcal{TR}_i} \sim \mathcal{GP}(m, k_{\boldsymbol{\theta}})$
11 $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X} \cap \mathcal{TR}_i} \alpha_{\text{NEHVI}}(\mathbf{x})$ *eq.* (14)
12 $\mathbf{y}_i \leftarrow$ *simulate*$(\mathbf{x}_i)$, $X \leftarrow X \cup \{\mathbf{x}_i\}$, $Y \leftarrow Y \cup \{\mathbf{y}_i\}$
13 Update trust regions with MORBO [12] lines **7-19**
14 **return** Pareto-optimal designs $\tilde{\mathcal{X}}_P$ and image values $\tilde{\mathcal{Y}}_P$

IV. Experimental results

To evaluate each aspect of candidate configuration efficiency, we chose `STREAM` [23], `DGEMM` [14], and `waLBerla` [17] as commonly used HPC benchmark applications. These benchmarks respectively target memory bandwidth, matrix computation, and numerical simulation performances.

We implemented SEBO, HyperMapper [25], and BOOM-Explorer [1] and interfaced them with our simulation flow to compare them on the same basis, on our benchmark applications. We also bring for the first time, a comparison with Hexaly Optimizer 13.0 (formerly known as LocalSolver) [4], a state-of-the-art commercial solver that provides simulation

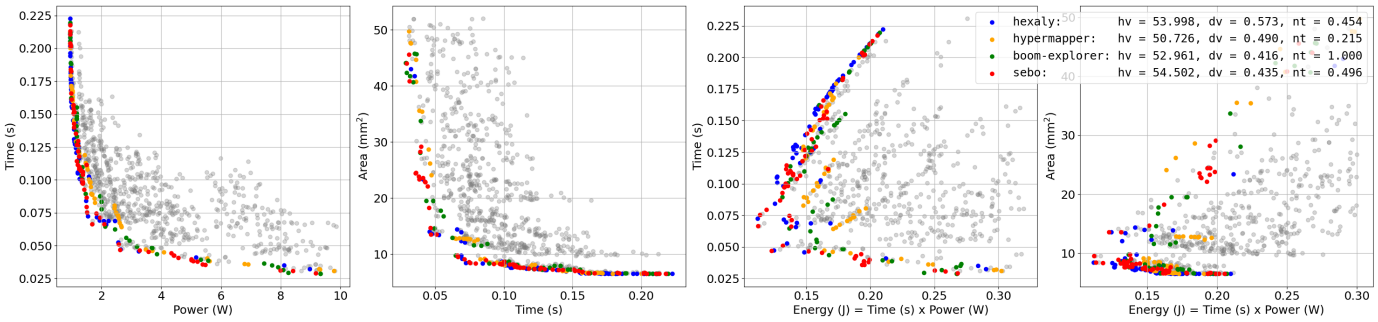


Fig. 4: Pareto fronts in bi-objective spaces with the hypervolume `hv`, diversity `dv` and normalized running time `nt` indicators

TABLE II: Key Pareto-optimal designs

	STREAM*			
<>-efficient	Design parameters†	Energy	Time	Area§
<Energy>	[16, 16, 128, 2048, 4096, 64, 2, 16, 32, 1, 1]	0.960	0.135	36.24
<Time>	[16, 1, 16, 2048, 8192, 128, 4, 4, 4, 4, 4]	1.246	0.094	78.46
<Area>	[16, 16, 16, 256, 512, 64, 2, 8, 16, 1, 1]	2.384	0.427	28.92
<Trade-off>‡	[16, 16, 128, 2048, 4096, 64, 2, 16, 32, 1, 1]	0.960	0.135	36.24
	DGEMM			
<Energy>	[2, 2, 64, 2048, 8192, 128, 2, 4, 8, 1, 1]	0.111	0.066	9.513
<Time>	[16, 16, 64, 2048, 8192, 128, 4, 16, 32, 1, 1]	0.268	0.029	44.02
<Area>	[2, 2, 16, 256, 512, 64, 2, 4, 16, 1, 1]	0.201	0.213	6.497
<Trade-off>	[4, 4, 128, 2048, 4096, 128, 2, 4, 4, 1, 1]	0.122	0.046	13.54
	waLBerla			
<Energy>	[16, 16, 128, 2048, 8192, 128, 2, 4, 4, 1, 1]	0.537	0.056	42.29
<Time>	[64, 1, 128, 2048, 8192, 128, 2, 16, 32, 8, 8]	2.491	0.054	270.1
<Area>	[2, 2, 16, 256, 512, 64, 2, 8, 16, 1, 1]	1.206	1.251	6.497
<Trade-off>	[8, 8, 16, 2048, 8192, 128, 4, 8, 32, 1, 1]	0.576	0.092	24.36

*For `STREAM`, the core count is fixed to 16, since its application size depends on it.
†The parameters follow the same order as TABLE I.
§The objectives use the same units as in Fig 4.
‡The trade-off design, which considers all objectives as equally important, is obtained by minimizing the equally-weighted Chebyshev aggregation evaluated on multi-objective values.

optimization solutions based on surrogate modeling, for more significant results. Since BOOM-Explorer and Hexaly do not support parallel exploration, the total simulation budget is set to 1000 for all algorithms, including the model initialization budget of $n = 64$ to ensure a fair comparison. For others, we allow $n_{\text{TR}} = 4$ parallel simulations (which is also the number of trust regions) at the same time during exploration. Assuming the core count is the most influential design parameter here, we also modify BOOM-Explorer's clustering parameter by the latter to enable it to be applied to our problem instances.

To compare exploration algorithms quantitatively, we propose three performance metrics: the hypervolume indicator, computed as the Lebesgue measure between the Pareto front and the nadir point of all Pareto fronts combined, the diversity indicator from Cai *et al.* [6] which evaluates the spreading and distribution of the front, and the normalized running time. In particular, the higher hypervolume, the better, and the lower diversity indicator and normalized running time, the better.

Fig. 4 provides an overview of the obtained Pareto fronts from SEBO and the baselines. The experimental results show that SEBO outperforms other baselines by around 1-7% in terms of hypervolume, indicating a higher overall quality of its Pareto front. Although the chosen baselines are top-performing DSE algorithms, SEBO still competes with them and demonstrates other strengths. For nearly the same hypervolume and normalized running time performances, SEBO provides a much higher diversity of its Pareto front than the commercial solution Hexaly by around 24% and than HyperMapper by 11%. Not only does SEBO return better Pareto-optimal designs related to their objectives values, but also provides a better diversification of solutions, which is essential for offering a broader range of efficient design choices to the architect. Furthermore, SEBO outscales BOOM-Explorer by running 2× faster and achieving a better Pareto front in the end. Interestingly, SEBO spent as much time as Hexaly to retrieve the Pareto front whereas the latter doesn't support parallel simulations. The increasing computational cost induced by ensemble learning is exactly counterbalanced by the use of trust regions and parallelization methods, thereby confirming the observations made in previous sections.

In TABLE II, we present some key Pareto-optimal designs that may interest designers. First, the results seems to be coherent. If we observe area-efficient designs, these are the ones that aim to minimize the values of design parameters, thus proving the consistency of the chosen simulators in correctly modeling design behavior. Secondly, we show the trade-off-efficient designs, which correspond to the best designs for architects who treats all objectives as equally important. The trade-off designs offer particular characteristics and are not necessarily the "average" of extreme designs. For instance, the trade-off design for `waLBerla` is around 2× slower than the time-efficient one, but about 4× and 11× more energy and area-efficient respectively. The gains are not proportional. Lastly, increasing the size of the architecture beyond a certain point no longer leads to faster execution times. If we observe time-efficient designs, None of them have the maximum core count. Whereas manual exploration may miss these observations, automated DSE procedures, in contrast, can capture non-trivial interactions between design configurations and performance metrics, highlighting DSE algorithms as game changers for designing efficient architectures.

V. Conclusion and Perspectives

In this work, we presented an automated framework based on our own DSE algorithm, SEBO, for efficient HPC processor design. With adapted features, SEBO returns qualitative and diversified Pareto designs with state-of-the-art performance. While we also provided key designs, selecting the optimal design according to the decision maker's preferences remains difficult and deserves to be studied as the final step of DSE.

REFERENCES

[1] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[2] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference (USENIX ATC)*, 2005.

[3] G. Beltrame, L. Fossati, and D. Sciuto, "Decision-Theoretic Design Space Exploration of Multiprocessor Platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2010.

[4] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua, "Local-Solver 1.x: a black-box local-search solver for 0-1 programming," *4OR*, 2011.

[5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, 2011.

[6] X. Cai, H. Sun, and Z. Fan, "A diversity indicator based on reference vectors for many-objective optimization," *Information Sciences*, 2018.

[7] A. Charif, G. Busnot, R. Mameesh, T. Sassolas, and N. Ventroux, "Fast Virtual Prototyping for Embedded Computing Systems Design and Exploration," in *Proceedings of the 11th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, 2019.

[8] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, 2016.

[9] S. Das and S. Tesfamariam, "State-of-the-Art Review of Design of Experiments for Physics-Informed Deep Learning," *arXiv preprint*, 2022.

[10] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Optimization," in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[11] S. Daulton, M. Balandat, and E. Bakshy, "Parallel Bayesian Optimization of Multiple Noisy Objectives with Expected Hypervolume Improvement," in *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[12] S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy, "Multi-objective Bayesian optimization over high-dimensional search spaces," in *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2022.

[13] A. Deshwal, S. Ament, M. Balandat, E. Bakshy, J. R. Doppa, and D. Eriksson, "Bayesian Optimization over High-Dimensional Combinatorial Spaces via Dictionary-based Embeddings," in *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023.

[14] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Transactions on Mathematical Software*, 1990.

[15] K. Dreczkowski, A. Grosnit, and H. Bou-Ammar, "Framework and benchmarks for combinatorial and mixed-variable bayesian optimization," in *Proceedings of the 37th International Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[16] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable Global Optimization via Local Bayesian Optimization," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[17] C. Feichtinger, S. Donath, H. Köstler, J. Götz, and U. Rüde, "WaLBerla: HPC software design for computational engineering simulations," *Journal of Computational Science*, 2011.

[18] P. I. Frazier, "Bayesian Optimization," in *INFORMS TutORials in Operations Research*, 2018.

[19] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip," in *IEEE/ACM International Conference on Computer Aided Design, IEEE/ACM Digest of Technical Papers*, 2001.

[20] A. Grosnit, C. Malherbe, R. Tutunov, X. Wan, J. Wang, and H. B. Ammar, "BOiLS: Bayesian Optimisation for Logic Synthesis," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022.

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[22] T.-R. Lin, D. Penney, M. Pedram, and L. Chen, "A Deep Reinforcement Learning Framework for Architectural Exploration: A Routerless NoC Case Study," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[23] J. McCalpin, "Memory bandwidth and machine balance in high performance computers," *IEEE Technical Committee on Computer Architecture Newsletter*, 1995.

[24] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007.

[25] L. Nardi, A. Souza, D. Koeplinger, and K. Olukotun, "HyperMapper: a Practical Design Space Exploration Framework," in *IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019.

[26] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES)*, 2002.

[27] J. Panerati, D. Sciuto, and G. Beltrame, "Optimization Strategies in Design Space Exploration," in *Handbook of Hardware/Software Codesign*, 2017.

[28] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[29] B. C. Schafer and K. Wakabayashi, "Design Space Exploration Acceleration Through Operation Clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2010.

[30] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2001.

[31] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020," in *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, 2021.

[32] S. Van den Steen, S. Eyerman, S. De Pestel, M. Mechri, T. E. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout, "Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics," *IEEE Transactions on Computers*, 2016.

[33] X. Wan, V. Nguyen, H. Ha, B. Ru, C. Lu, and M. A. Osborne, "Think Global and Act Local: Bayesian Optimisation over High-Dimensional Categorical and Mixed Search Spaces," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[34] T. Wolf and M. Franklin, "Performance models for network processor design," *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[35] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, 1997.

[36] T.-T. Wong, W.-S. Luk, and P.-A. Heng, "Sampling with Hammersley and Halton Points," *Journal of Graphics Tools*, 1997.

[37] K. Yu, J. Bi, and V. Tresp, "Active learning via transductive experimental design," in *Proceedings of the 23rd international conference on Machine learning (ICML)*, 2006.

[38] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "McPAT-Calib: A Microarchitecture Power Modeling Framework for Modern CPUs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

PIMDup: An Optimized Deduplication Design on a Real Processing-in-Memory System

Chun-Le Yeh
Department of CSIE
National Cheng Kung University
Tainan, Taiwan (R.O.C.)
p76101356@gs.ncku.edu.tw

Liang-Chi Chen
Department of CSIE
National Taiwan University
Taipei, Taiwan (R.O.C.)
d12922012@csie.ntu.edu.tw

Chien-Chung Ho
Department of CSIE
National Cheng Kung University
Tainan, Taiwan (R.O.C.)
ccho@gs.ncku.edu.tw

Yu-Ming Chang
Wolley Inc.
Hsinchu, Taiwan (R.O.C.)
san@wolleytech.com

Da-Wei Chang
Department of CSIE
National Cheng Kung University
Tainan, Taiwan (R.O.C.)
davidchang@csie.ncku.edu.tw

***Abstract*—Data deduplication enhances storage efficiency through non-destructive compression but is often hindered by the chunking process, which requires scanning the entire dataset. While traditional methods leveraging conventional architectures and hardware accelerators (e.g., GPUs and FPGAs) have been developed to address this issue, they continue to face challenges related to excessive data movement and associated performance degradation. These limitations stem from the von Neumann architecture, where computation and storage are separated in a processor-centric design, necessitating multiple memory hierarchy traversals and causing inefficiencies. To overcome these challenges, we explore UPMEM's DPU, a processing-in-memory (PIM) technology that reduces data movement by performing computations directly within memory. However, designing a deduplication system for DPUs presents unique obstacles, including restricted inter-DPU data sharing, the absence of native multiplication support, and significant DPU-CPU communication overhead. In response, we propose PIMDup, a DPU-optimized deduplication system that addresses these constraints through efficient parallelization, DPU-friendly chunking techniques, and reduced data transfer volumes. Experimental results demonstrate that PIMDup improves chunking performance without compromising deduplication accuracy, achieving a 1.67× speedup over CPU-based systems while maintaining 100% result consistency.**

***Index Terms*—Deduplication system, processing-in-memory, UPMEM DPU**

I. Introduction

In recent years, we have entered the era of the information explosion. This growing disparity places significant strain on storage systems, which must manage increasingly larger volumes within constrained capacities. To address these challenges, various solutions have been proposed, including distributed storage systems [1], [2], parallel architectures [3], [4], and hybrid systems leveraging solid-state drives (SSDs) [5], [6], which offer performance advantages over traditional hard disks. However, despite these advances, the fundamental issue persists: the sheer volume of data must be reduced. As a result, techniques for lowering storage costs are becoming increasingly critical.

Compression [7] and data deduplication [8] are two primary methods for enhancing storage efficiency. Compression reduces the size of individual files by identifying and re-encoding redundancy, with the degree of reduction depending on whether the process is lossy or lossless. In contrast, data deduplication identifies and eliminates duplicate data segments by referencing existing copies, conserving write bandwidth, and improving storage utilization. In data centers, deduplication typically achieves higher compression ratios than traditional compression while preserving non-destructive characteristics, allowing full restoration of original data. The deduplication process consists of four key steps [9]: (1) files are divided into chunks, the smallest units for deduplication; (2) each chunk is assigned a unique hash value (fingerprint) using a secure hash algorithm; (3) fingerprints are compared with existing ones to identify duplicates; and (4) duplicate chunks are replaced with references, while unique chunks and their fingerprints are stored for future reconstruction. Among these steps, *chunking—segmenting data into de-duplicatable fragments—is the most time-intensive, accounting for up to 53% of deduplication runtime [10]*. Accelerating chunking through processing-in-memory (PIM) technologies offers a promising approach to improving overall deduplication performance.

Processing-in-Memory (PIM) technology addresses data-intensive challenges by enabling computation directly within memory, reducing data movement overhead [11]. The UPMEM DPU, the first commercially available PIM solution [12], has demonstrated significant potential for tackling memory-bound workloads [13]–[19]. By integrating hundreds of processors within DRAM DIMMs, UPMEM DPUs allow parallelized, on-site data processing, making them well-suited for accelerating deduplication workflows. However, two major limitations constrain UPMEM DPU's effectiveness. First, data transfer remains dependent on the host CPU, as DPUs cannot directly access storage, limiting the offloading of the entire deduplication process. Second, UPMEM DPUs lack inter-processor communication [20], [21], complicating the generation of consistent chunking results across multiple processors. Despite these challenges, UPMEM DPUs offer substantial advantages, provided that data transfer costs are carefully managed and deduplication accuracy is maintained. Motivated by these opportunities and limitations, we propose PIMDup, a deduplication system explicitly optimized for DPU archi-

tectures. PIMDup addresses the unique challenges of DPU-based deduplication through a carefully designed system that balances hardware constraints with deduplication principles. This work aims to: (1) identify and address the technical challenges of implementing chunking on DPUs, (2) design a deduplication framework tailored for DPU hardware, and (3) evaluate the system's performance and deduplication accuracy through comprehensive experiments.

II. Background

A. Data Deduplication

Deduplication operates across files by identifying and eliminating redundant data segments, ensuring only unique data is stored. By referencing duplicate data instead of storing it repeatedly, deduplication significantly reduces storage volume, delivering high compression ratios while retaining lossless, non-destructive characteristics. Due to these advantages, deduplication has become a widely adopted technique in modern storage systems, offering both bandwidth savings and improved storage efficiency.

The deduplication process typically consists of four main steps: chunking, fingerprinting, indexing, and storage, as illustrated in Figure 1. (1) Chunking: Each file is divided into chunks, either of fixed or variable size, depending on the chosen chunking strategy. (2) Fingerprinting: A secure hash algorithm, such as SHA-1, MD5, or SHA-256 [22]–[24], is used to compute a unique fingerprint (hash value) for each chunk, serving as its identifier. (3) Indexing: The generated fingerprints are compared against an index to check for duplicates. If a matching fingerprint is found, the chunk is identified as redundant; otherwise, it is considered unique. (4) Storage: Unique chunks are stored, and their fingerprints are added to the index. For duplicate chunks, the system creates references to the already stored chunks, avoiding redundant storage. When reading data in a deduplication system, the original file is reconstructed by retrieving both unique chunks and references to duplicates. Using metadata to identify the correct chunk sequence, the system reassembles the data in its original form. This ensures that deduplicated data can be seamlessly restored when needed, maintaining usability and reliability.

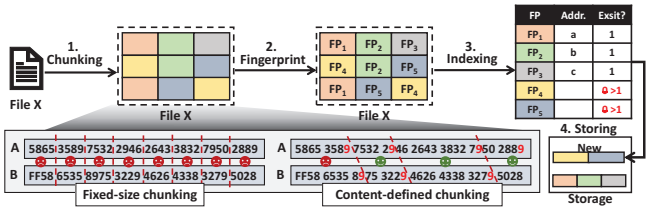


Fig. 1. The work-flow of deduplication process.

Among the four steps, chunking is particularly critical, as it determines how data is segmented into the smallest units for deduplication. An effective chunking strategy maximizes storage efficiency by accurately identifying redundant data. Conversely, a poorly designed chunking method may fail to detect similarities, resulting in suboptimal deduplication ratios and wasted computational effort. As shown in Figure 1, chunking methods are broadly classified into two categories: fixed-size chunking [25] and content-defined chunking [26]. Fixed-size chunking method has a well-known limitation, the boundary-shift problem [27], [28], which results in reduced deduplication ratios. To address this, content-defined chunking (CDC) has been developed. CDC uses specific patterns within the file's content as chunk boundary markers. Combining these markers with predefined maximum and minimum chunk sizes helps determine chunk boundaries, as seen in the example of number 9 in Figure1. The data between two chunk boundaries forms a chunk. Since the content itself determines these boundaries, CDC is capable of detecting redundant data even in files with slight variations, such as insertions or deletions. Therefore, this work adopts content-defined chunking to optimize the chunking process in deduplication systems.

B. UPMEM PIM DIMM

The UPMEM DRAM DIMM is a widely adopted commercially available Processing-in-Memory (PIM) product [12]. It integrates multiple in-DRAM processors, known as DRAM Processing Units (DPUs), directly into the memory. Unlike the traditional von Neumann architecture, which separates processing unit (CPU), memory, and storage, the UPMEM PIM DIMM enables computational capabilities at the memory tier, effectively breaking the conventional von Neumann bottleneck, i.e., data movement overheads. As shown in Figure 2, each UPMEM PIM DIMM comprises 16 PIM DRAM chips, and they are organized into two ranks of 8 chips each. Every chip contains 8 DPUs, resulting in 64 DPUs per rank. Each DPU operates as an independent processor and is equipped with a 64 MB DRAM bank, referred to as MRAM. In addition, each DPU has 64 KB of working memory (WRAM), functioning as a scratchpad, and 24 KB of instruction memory (IRAM) to store encoded instructions.

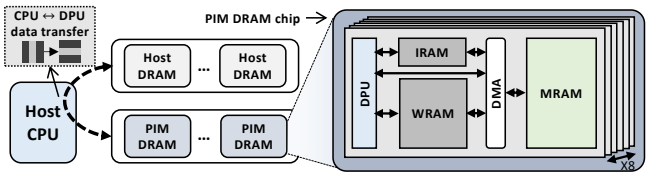


Fig. 2. The architecture of UPMEM DPU.

While DPUs enable memory-level computation and provide high external communication bandwidth via DDR slots, their architecture poses several challenges [20]. First, each DPU can only access its own MRAM, with no capability for direct data sharing or communication between DPUs, even within the same rank. Any inter-DPU communication must be mediated by the host CPU, introducing significant latency. Second, the limited capacity of WRAM necessitates careful memory management to minimize data transfers between WRAM and MRAM, as frequent transfers can negatively impact performance [15]. Third, DPUs cannot directly access data stored on disk; instead, data must first be transferred to the host CPU, which then forwards it to the DPU's MRAM. This process introduces additional overhead, such as matrix transpositions, which can further reduce system efficiency. Finally, the computational performance of DPUs is generally

lower than that of the host CPU, particularly for operations such as multiplication and division. These computational limitations can lead to increased execution latency during complex workloads, as further discussed in Section II-C.

C. Observation and Motivation

As outlined in earlier sections, DPUs cannot directly access data from storage nor share data among themselves. However, the deduplication process, particularly the indexing stage, relies on a global table for storing all fingerprints and involves storage I/O operations. Additionally, chunking—responsible for nearly half of the deduplication runtime—is a highly data-intensive task. Consequently, chunking presents a more suitable target for DPU acceleration. Based on this observation, large files can be divided into multiple segments and allocated to UPMEM PIM DIMMs for chunking within the DPUs, while the CPU handles the remaining tasks. Such a management procedure is illustrated in Figure 3: (❶) Each file segment undergoes chunking on the DPUs. (❷) The resulting chunking data is sent back to the host CPU. (❸) The host performs fingerprinting, indexing, and storage I/O while the DPUs concurrently process chunking tasks for subsequent file segments (looping back to ❶). This pipelined approach overlaps DPU and CPU operations, maximizing resource utilization.

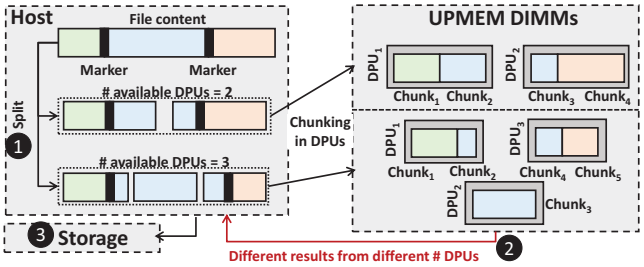


Fig. 3. A naïve design of a DPU-based deduplication.

This straightforward design encounters several critical challenges. First, the chunking process relies on a sliding window mechanism that scans each file segment, computing a hash at each position. When the hash matches a predefined value, a marker is placed to denote a potential chunk boundary. However, if a marker spans two DPUs, it cannot be accurately identified. Once markers are determined, each DPU uses them to define chunk boundaries, adhering to constraints on maximum and minimum chunk sizes. A key issue arises at the boundaries of file segments, where the DPU may fail to identify markers accurately, treating the remaining portion as a single chunk. This inconsistency leads to varying chunking results depending on the number of DPUs utilized. For instance, as illustrated in Figure 3, processing a file with 2 DPUs produces 4 chunks, whereas using 3 DPUs results in 5 chunks, undermining the stability and effectiveness of the deduplication process. Second, existing studies highlight a disparity in data transfer bandwidth between the CPU and DPUs. Specifically, the bandwidth for sending data from the host CPU to the DPUs exceeds that for returning processed results from DPUs to the CPU [20]. In the naïve design depicted in Figure 3, this imbalance results in smaller data volumes being transmitted over the faster channel (CPU to DPU), while larger processed data sets are sent over the slower channel (DPU to CPU). This inefficiency increases data transfer times and ultimately degrades overall system performance.

Moreover, the chunking process relies on hash functions to identify markers within file content. However, state-of-the-art hashing algorithms are not well-suited for execution on DPUs due to their reliance on polynomial equations that involve a significant number of multiplication operations. DPUs exhibit a substantial performance gap between addition and multiplication operations due to hardware limitations. As illustrated in Figure 4, the throughput for addition/subtraction operations on 32-bit integers reaches approximately 58.33 MOPS, whereas the throughput for multiplication operations is considerably lower. This disparity becomes even more pronounced for 64-bit integers, where addition/subtraction operations significantly outperform multiplication. These performance trends are consistent with benchmarks from prior research, which demonstrate that DPUs are poorly suited for tasks requiring extensive multiplication.

To address these challenges and the inherent hardware limitations of DPUs, we have formulated several key design objectives. First, all potential marker positions within a file must be identified accurately, even when markers span across multiple DPUs, to ensure precise chunking boundaries. Second, the chunking results must remain consistent, irrespective of the number of DPUs involved, to preserve the deduplication system's capacity for effective storage reduction. Third, considering the inefficiency of DPUs in performing multiplication-heavy operations, it is crucial to develop a chunking method tailored to the computational strengths of DPUs, thereby enhancing execution performance. Finally, the disparity in data transfer speeds between the DPUs' input and output channels must be addressed through optimized data transfer mechanisms to minimize performance bottlenecks. With these objectives in mind, this work introduces *PIMDup*, a DPU-based deduplication system specifically designed to capitalize on the in-memory processing capabilities and parallelism of DPUs. PIMDup is structured to address the outlined challenges while ensuring consistency and efficiency across a range of DPU configurations.

III. Proposed PIMDup Desing

A. Overview

Content-defined chunking is essential for efficient data deduplication but is also the most time-consuming step. To address this, we employ DPUs as parallel workers to accelerate the process. To ensure consistent chunking results across varying DPU counts, we propose an *anti-splitting file segmentation* (ASFS) method to identify all markers in the input file accurately. A *chunk boundary vector* mechanism records marker positions in DPUs, which are merged on the host CPU to define global chunk boundaries based on conditions like maximum chunk size (see Section III-B). Additionally, an *encoded boundary vector* compresses boundary data, reducing data transfer to one-eighth of the input file size (see Section III-C). To enhance performance, a *DPU-friendly chunking method* avoids inefficient computational patterns, improving speed without sacrificing accuracy (see Section III-D).

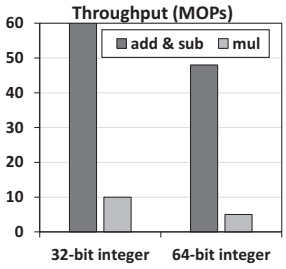


Fig. 4. A huge gap between arithmetic performances in the DPU.

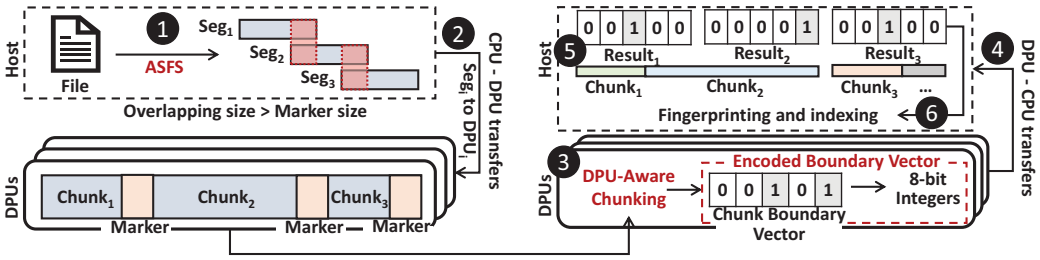


Fig. 5. Overview of proposed PIMDup.

Figure 5 outlines the workflow of the proposed *PIMDup* system. The host CPU first segments the input file using ASFS ❶ and distributes segments to the DPUs ❷. DPUs perform chunking using the DPU-friendly method, identifying markers and storing them in a chunk boundary vector ❸. These vectors are encoded into 8-bit integers and sent to the host CPU ❹, which reconstructs the vectors, aligns them with the input file, and applies constraints to finalize chunk boundaries ❺. The host CPU then completes fingerprinting and indexing ❻. While the CPU processes chunking results, the DPUs can immediately begin processing the next file segment, ensuring efficient parallelism and maximizing throughput. This pipeline reduces data transfer overhead and accelerates the deduplication process.

B. Anti-Splitting File Segmentation (ASFS)

DPUs lack a direct mechanism for sharing data among themselves. Consequently, if a file is split into mutually exclusive segments and distributed across multiple DPUs, a marker could fall on the boundary between segments, resulting in an incorrect marking of chunk boundary candidates. This error compromises the deduplication system's efficiency, reducing its bandwidth and space-saving benefits. To address this issue, we propose the anti-splitting file segmentation (ASFS) technique, which introduces overlapping regions between file segments during segmentation. The length of this overlap is set to exceed the size of the designated markers, as illustrated in Figure 5. In cases where a marker lies on the boundary between two DPUs, one of the DPUs will contain a complete copy of the marker within its overlapping segment. This ensures that the marker can be correctly identified during content-defined chunking, preventing errors in chunk boundaries. By using ASFS, we can guarantee that all markers are accurately detected, regardless of the number of active DPUs. In large-scale systems with thousands of DPUs, particularly in scenarios where the UPMEM PIM DIMM's yield varies, the number of DPUs assigned to a task may change dynamically. ASFS addresses this challenge by ensuring consistent chunking results across different DPU allocations, even when a varying number of DPUs processes the same file.

It is worth mentioning that the overlapping segment caused by applying ASFS does not incur significant memory duplicated overhead. Given that typical marker sizes are around 32 bytes, the extra memory required for overlapping segments is negligible. As these markers are often numerous and distributed randomly throughout the file, ASFS significantly enhances performance. By ensuring that each marker is contained fully within at least one segment, ASFS prevents costly chunking errors and preserves the deduplication system's bandwidth and storage efficiency. The benefits of this solution scale effectively, particularly in systems with high marker density, leading to improved overall performance with minimal trade-offs in space utilization.

C. Encoded Boundary Vector

The primary cause of inconsistency in chunking results is the inability of DPUs to communicate, which prevents them from having a "global view" when determining chunk boundaries. This limitation can result in some data segments being arbitrarily defined as chunks, even if they lack proper boundaries on both ends. To address this issue, we propose using a chunk boundary vector. As shown in Figure 5, after each DPU completes its chunking process, it generates a chunk boundary vector that records the positions of special markers, which are potential candidates for chunk boundaries. At this stage, these segments are not yet defined as chunks. Instead, each DPU transmits its chunk boundary vector to the host CPU. The host CPU then aggregates these individual vectors to create a "global vector" that corresponds to the entire file, capturing all potential chunk boundaries. Based on this global vector and predefined parameters, such as maximum and minimum chunk sizes, the host CPU determines the final chunk boundaries. Compared to the naïve design in Figure 3, where each DPU independently defines chunk boundaries, this approach allows the host CPU to make boundary decisions with a comprehensive, file-wide perspective, thereby ensuring consistency in the chunking results. Thus, it can eliminate the problem shown in Figure 3, that is, the inconsistent chunking outcomes for the same file when it is distributed across varying numbers of DPUs.

As discussed in Section II-C, the data transfer channels between the CPU and DPUs have differing speeds, with the output channel (DPU-to-CPU) being slower than the input channel (CPU-to-DPU). This difference makes it crucial to minimize the amount of data sent through the slower output channel. As shown in ❸ of Figure 5, to optimize data transfer efficiency, we encode the chunk boundary vector as a binary array, where each '1' represents the presence of a marker. By grouping every 8 bits of this array into a single binary integer, we can represent the chunk boundary information

much more compactly. Each element in the chunk boundary vector corresponds to the outcome of the sliding window hash operation, indicating whether the calculated hash value at each byte matches the designated marker. This setup enables the chunking results for every 8 bytes of the file to be encoded as a single 8-bit integer, which is then transferred from the DPUs back to the main memory. Compared to a non-encoded chunk boundary vector, where each byte in the vector represents whether a corresponding byte in the input file contains a marker, this binary encoding is much more efficient. By encoding each marker presence as a single bit, we reduce the data size that needs to be transferred. This approach ultimately reduces the transfer data size to one-eighth of the original, as it efficiently captures marker information within each byte.

D. DPU-Friendly Chunking Method

As illustrated in Figure 4, DPU performance is critical for determining processing speed. However, conventional chunking algorithms often rely on hashing with polynomial functions, which significantly hinder DPU execution efficiency. A straightforward solution might involve using a pre-defined table within each DPU to store all possible string values. However, this approach is impractical due to the limited 64 MB of MRAM available per DPU, which must simultaneously accommodate data storage and results for transfer back to the host, leaving insufficient memory for such a table.

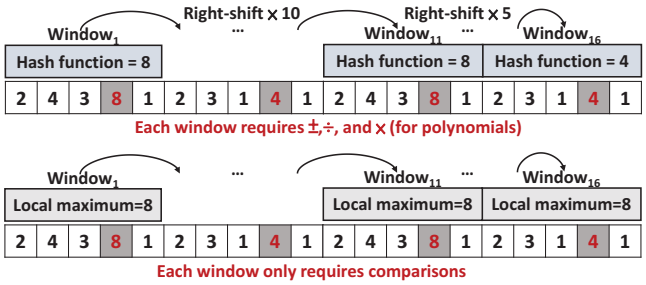


Fig. 6. Chunking by Rabin-based hashing v.s DPU-friendly chunking method

To overcome this limitation, we propose a non-hashing, DPU-friendly chunking method based on local-maximum-byte chunking [29]. This method employs a sliding window of predefined size to scan the data. As the window moves, the middle byte is checked to determine if it is the maximum within the window. If it is not, the window advances. If it is, the boundary between the middle byte and the preceding one is marked as a candidate chunk boundary. These boundary positions are then recorded using the encoded boundary vector method, ensuring efficient transfer back to the host. To align with the granularity of the encoded boundary vector, the proposed method uses a 32-byte sliding window to process the file. Unlike hashing-based approaches that rely on polynomial calculations and require multiplication and division operations, the local-maximum-byte chunking method only involves comparing byte values within the window. This comparison-based approach eliminates the computational overhead associated with polynomial functions. As shown in Figure 6, Rabin-based hashing methods involve numerous multiplications for polynomial functions, whereas the DPU-friendly chunking method relies solely on comparisons. These comparisons, implemented using subtraction, are highly efficient for DPUs due to their hardware design, further enhancing overall processing performance.

IV. Experiemt

A. Experiemtal Setup

In this section, we evaluate the performance of the proposed PIMDup design from multiple perspectives. The analysis of the experimental results is conducted based on the following metrics: (1) Internal DPU Performance: This metric assesses the benefits derived from the DPU-Friendly Chunking Method. (2)Data Transfer Bandwidth: To evaluate the effectiveness of the Encoded Boundary Vector, we measure the data transfer bandwidth between the CPU and DPUs. (3) Result Consistency: We verify that different numbers of DPUs performing chunking do not lead to inconsistent results. (4) Overall System Performance: We compare the performance of PIMDup against a traditional CPU-based deduplication system.

To demonstrate the feasibility of the PIMDup design, experiments were conducted on a real system equipped with UPMEM DPUs. Specifically, we utilized a UPMEM server with the following configuration: an Intel(R) Xeon(R) Silver 4110 CPU, 128 GB of host main memory, and 20 dual-rank PIM DIMMs, providing a total of 40 ranks and 2560 DPUs operating at @ 350 MHz. . The PIMDup system was implemented in C/C++ using the UPMEM SDK[1], and DPU-side chunking logic was programmed using the LLVM-based compiler provided by the SDK. This setup enabled direct performance comparison between PIMDup and traditional CPU-based deduplication systems.

The dataset used to evaluate PIMDup comprises 30 different release versions of virtual machine image files [9], [30]. The dataset includes 6 CentOS, 5 Fedora, and 19 Debian image files, collectively amounting to approximately 30 GB. These files exhibit deduplication ratios ranging from 38% to 55%, providing a comprehensive and diverse testbed for performance evaluation. The mixture of distributions and versions allows for a robust assessment of PIMDup's performance and consistency across various data types, thereby validating its effectiveness and reliability in real-world scenarios.

B. Experiemtal Result

To comprehensively evaluate the scalability of the proposed PIMDup design, we incrementally increased the number of DPU ranks from 4 to 36 (with each rank consisting of 64 DPUs) and measured the total execution time required for deduplication.

Figure 7 illustrates the results, detailing the time required for chunking, fingerprinting, and indexing across varying numbers of DPU ranks. The experimental results reveal a significant reduction in the total deduplication time as the number of DPU ranks increases. Notably, chunking remains the largest contributor to the overall deduplication time, though it decreases substantially with the addition of more DPU ranks. This improvement is attributable to the parallel processing

[1]https://sdk.upmem.com

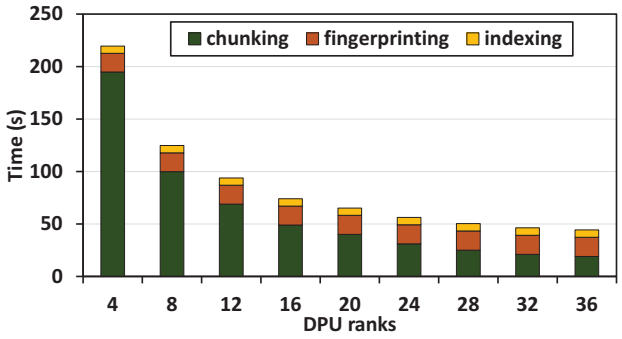


Fig. 7. The breakdown analysis for chunking, fingerprinting, and indexing time of PIMDup.

capabilities of the DPUs, which allow for more efficient handling of data chunks. In contrast, fingerprinting and indexing times show little to no improvement as the number of DPU ranks increases. This is because these tasks are executed by the CPU in PIMDup and are therefore unaffected by the scaling of DPUs. Despite this, the proposed PIMDup design ensures that the host CPU does not experience significant overhead during the integration of processed data. This demonstrates the scalability and efficiency of PIMDup in leveraging parallelism within DPUs while maintaining minimal impact on host-side operations.

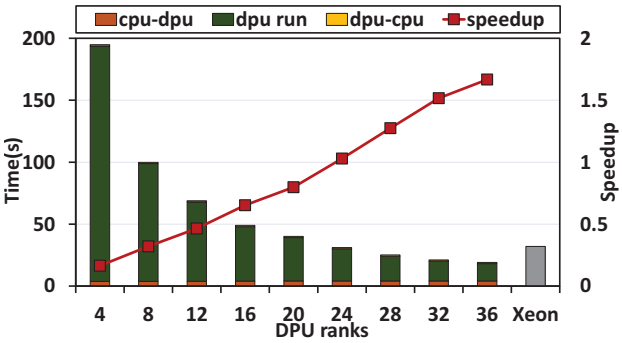


Fig. 8. The overall performance analysis and comparison for PIMDup.

Figure 8 provides several important insights into the performance of PIMDup. As the number of DPU ranks increases, the deduplication time decreases significantly, showcasing the scalability of the DPU-based implementation. Specifically, the processing time drops dramatically—from nearly 200 seconds with 4 DPU ranks to approximately 20 seconds with 36 DPU ranks. When compared to CPU-based deduplication (e.g., using Xeon processor), the advantages of our proposed PIMDup become even more apparent. At the maximum number of DPU ranks, PIMDup achieves a 1.67× reduction in processing time relative to the Xeon-based system. This improvement highlights how the DPU architecture excels in managing memory-bound workloads more efficiently than traditional CPU-based systems. Moreover, the data transfer overhead between the CPU and DPUs remains minimal, thanks to the adoption of the Encoded Boundary Vector design. This mechanism allows DPUs to efficiently notify the host of chunking results using a compact representation, reducing the volume of data that must be transferred. Consequently, the host is able to seamlessly integrate the chunking information while minimizing data transfer overhead, further enhancing the overall system efficiency.

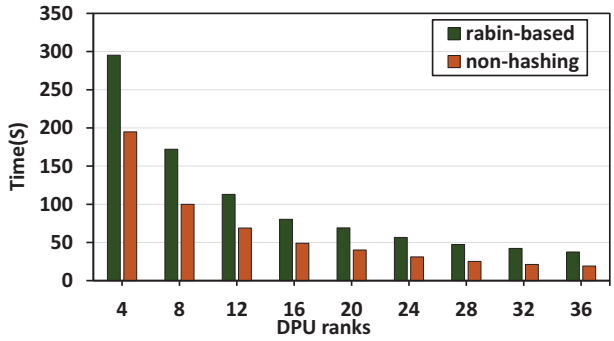


Fig. 9. The time required for rabin-DPU and non-hashing-DPU using different numbers of DPU ranks.

Figure 9 demonstrates that the DPU-friendly chunking method consistently outperforms the Rabin-based chunking method across all tested DPU configurations, regardless of the number of ranks employed. Notably, the DPU-friendly chunking method achieves an approximately 50% reduction in processing time compared to the Rabin-based approach. This substantial performance improvement stems from the inherent hardware constraints of the DPUs. As highlighted in our preliminary analysis, DPUs exhibit significantly higher throughput for addition and subtraction operations than for multiplication and division. The Rabin-based algorithm, which relies on polynomial multiplication to compute hash values, is computationally intensive and less suited to the architecture of the DPUs. In contrast, the DPU-friendly chunking method aligns better with the DPUs' operational strengths, enabling more efficient processing.

V. Conclusion

In this work, we present PIMDup, a deduplication system specifically optimized for DPUs, addressing critical challenges such as limited inter-DPU data sharing, the absence of native multiplication support, and the overhead associated with DPU-CPU data transfers. To overcome these issues, PIMDup introduces three key innovations: Anti-Splitting File Segmentation (ASFS) for efficient data allocation, an Encoded Boundary Vector to streamline data transfers between DPUs and the CPU, and a DPU-Friendly Chunking Method to enable high-performance execution on DPUs. Our experimental evaluation demonstrates that PIMDup achieves a 1.67× speedup compared to a CPU-based deduplication system while ensuring 100% result consistency. This highlights the effectiveness and reliability of our proposed design.

VI. Acknowledgments

This work was supported in part by National Science and Technology Council under grant nos. 112-2221-E-006-126-MY3 and 113-2221-E-006-217-MY3.

REFERENCES

[1] Y. Yamato, "Use case study of hdd-ssd hybrid storage, distributed storage and hdd storage on openstack," in *Proceedings of the 19th International Database Engineering & Applications Symposium*, ser. IDEAS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 228–229. [Online]. Available: https://doi.org/10.1145/2790755.2790795

[2] Q. He, Z. Li, and X. Zhang, "Study on cloud storage system based on distributed storage systems," in *2010 International Conference on Computational and Information Sciences*, 2010, pp. 1332–1335.

[3] B. Cha, S. Park, and J. Kim, "Design and verification of software-defined raid for hybrid cloud storage," in *2016 International Conference on Cloud Computing Research and Innovations (ICCCRI)*, 2016, pp. 128–133.

[4] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "Diskreduce: Raid for data-intensive scalable computing," *Proceedings of the 4th Annual Petascale Data Storage Workshop, PDSW '09, Held in conjunction with Supercomputing '09*, 11 2009.

[5] Y. Yamato, "Cloud storage application area of hdd–ssd hybrid storage, distributed storage, and hdd storage," *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 11, no. 5, pp. 674–675, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/tee.22287

[6] D. Boukhelef, K. Boukhalfa, J. Boukhobza, H. Ouarnoughi, and L. Lemarchand, "Cops: Cost based object placement strategies on hybrid storage system for dbaas cloud," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 659–664.

[7] D. A. Lelewer and D. S. Hirschberg, "Data compression," *ACM Computing Surveys (CSUR)*, vol. 19, no. 3, pp. 261–296, 1987.

[8] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (ToS)*, vol. 7, no. 4, pp. 1–20, 2012.

[9] W. Xia, D. Feng, H. Jiang, Y. Zhang, V. Chang, and X. Zou, "Accelerating content-defined-chunking based data deduplication by exploiting parallelism," *Future Generation Computer Systems*, vol. 98, pp. 406–418, 2019.

[10] F. Ni, X. Lin, and S. Jiang, "Ss-cdc: A two-stage parallel content-defined chunking for deduplicating backup storage," in *Proceedings of the 12th ACM International Conference on Systems and Storage*, 2019, pp. 86–96.

[11] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," in *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*. Springer, 2022, pp. 171–243.

[12] F. Devaux, "The true processing in memory accelerator," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 2019, pp. 1–24.

[13] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi *et al.*, "A case study of {Processing-in-Memory} in {off-the-Shelf} systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 117–130.

[14] B. Hyun, T. Kim, D. Lee, and M. Rhu, "Pathfinding future pim architectures by demystifying a commercial pim technology," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 263–279.

[15] L.-C. Chen, S.-Q. Yu, C.-C. Ho, Y.-H. Chang, D.-W. Chang, W.-C. Wang, and Y.-M. Chang, "Rna-seq quantification on processing in memory architecture: Observation and characterization," in *2022 IEEE 11th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2022, pp. 26–32.

[16] A. Baumstark, M. A. Jibril, and K.-U. Sattler, "Processing-in-memory for databases: Query processing and data transfer," in *Proceedings of the 19th International Workshop on Data Management on New Hardware*, 2023, pp. 107–111.

[17] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures," *ACM SIGMETRICS Performance Evaluation Review*, vol. 50, no. 1, pp. 33–34, 2022.

[18] S. Cai, B. Tian, H. Zhang, and M. Gao, "Pimpam: Efficient graph pattern matching on real processing-in-memory hardware," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–25, 2024.

[19] D. Lavenier, R. Cimadomo, and R. Jodin, "Variant calling parallelization on processor-in-memory architecture," in *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2020, pp. 204–207.

[20] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," in *2021 12th International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2021, pp. 1–7.

[21] L.-C. Chen, C.-C. Ho, and Y.-H. Chang, "Uppipe: A novel pipeline management on in-memory processors for rna-seq quantification," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[22] A. S. M. Saeed and L. E. George, "Fingerprint-based data deduplication using a mathematical bounded linear hash function," *Symmetry*, vol. 13, no. 11, p. 1978, 2021.

[23] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying data deduplication," in *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, 2008, pp. 12–17.

[24] J. Malhotra and J. Bakal, "A survey and comparative study of data deduplication techniques," in *2015 International Conference on Pervasive Computing (ICPC)*. IEEE, 2015, pp. 1–5.

[25] S. Quinlan and S. Dorward, "Venti: A new approach to archival data storage," in *Conference on file and storage technologies (FAST 02)*, 2002.

[26] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu, and Y. Zhang, "{FastCDC}: A fast and efficient {Content-Defined} chunking approach for data deduplication," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 101–114.

[27] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, and Y. Zhou, "Ae: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1337–1345.

[28] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001, pp. 174–187.

[29] N. Bjørner, A. Blass, and Y. Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," *Journal of Computer and System Sciences*, vol. 76, no. 3-4, pp. 154–203, 2010.

[30] VMs Archives, "Vmware virtual machines," http://www.thoughtpolice.co.uk/vmware/.

Quorum: Zero-Training Unsupervised Anomaly Detection using Quantum Autoencoders

Jason Zev Ludmir*, Sophia Rebello*, Jacob Ruiz†, Tirthak Patel*
Rice University, Stanford University†*

***Abstract*—Detecting mission-critical anomalous events and data is a crucial challenge across various industries, including finance, healthcare, and energy. Quantum computing has recently emerged as a powerful tool for tackling several machine learning tasks, but training quantum machine learning models remains challenging, particularly due to the difficulty of gradient calculation. The challenge is even greater for anomaly detection, where unsupervised learning methods are essential to ensure practical applicability. To address these issues, we propose Quorum, the first quantum anomaly detection framework designed for unsupervised learning that operates without requiring any training.**

I. Introduction

Anomaly detection plays an essential role in various industries, from identifying fraudulent transactions in finance to detecting irregularities in power grids [26], [8], [25], [36] As datasets grow in complexity, traditional machine learning (ML) methods struggle with scalability and accuracy. Quantum computing offers a promising new approach, with its potential to accelerate computations and detect subtle patterns and correlations in data. Leveraging quantum algorithms for ML tasks, particularly for anomaly detection, could transform how we tackle these challenges [28], [27], [33].

The Challenge. Applying quantum computing to anomaly detection presents significant hurdles. Quantum machine learning (QML) models typically rely on parameterized circuits that require training, which is challenging due to the complexity of computing gradients in quantum systems [16], [31]. Quantum systems require gradient calculations from first principles using the parameter shift rule, and these gradients are prone to exponential vanishing in "barren plateau" regions [13]. Moreover, anomaly detection, by nature, is unsupervised, adding another layer of difficulty since no labeled data is available to guide the training [29], [15]. These two factors – quantum training complexity and unsupervised learning requirements – create a considerable challenge for developing efficient quantum anomaly detection methods [29], [14].

The Gap. *Current quantum-based approaches for anomaly detection fall short because they still require training and often rely on supervised or semi-supervised learning.* These methods involve optimizing quantum circuits with labeled data, which limits their applicability in real-world scenarios where such data is scarce. This dependency on training creates both computational overheads and reduces the flexibility needed for fully unsupervised tasks [16], [13], [21], [12], [29], [14], [1].

Our Solution. To address this, we propose Quorum, the first quantum anomaly detection framework that requires zero training and is designed for unsupervised learning. Quorum leverages quantum principles such as amplitude encoding, random quantum transformations, and SWAP tests to identify anomalies without needing any parameter optimization [16], [13]. By utilizing quantum correlations and random projections, Quorum detects anomalies based on deviations from the statistical norms of quantum transformations [17], [21], [13].

First, Quorum's open-source technique carefully distributes the data into buckets based on the likelihood of anomalous events in the dataset before embedding it into quantum states using amplitude encoding [12], [2]. Then, Quorum applies random quantum transformations to this data and uses a SWAP test to measure similarly between quantum states [13], [16]. Quorum constructs an "embarrassingly parallelizable" ensemble of such random transformations and leverages statistical measures to identify anomalies. This approach is scalable and flexible, allowing for efficient anomaly detection without the computing cost of gradient calculation and training [29], [21].

Quorum's Evaluation. We evaluate Quorum through extensive experiments on various datasets, including medical, industrial, and lexical data. We compare its performance to a state-of-the-art method that uses a quantum neural network (QNN) which relies on training and supervised labels. Our results show that Quorum has a 23% higher average F1 score over the QNN across evaluated datasets. We also provide an ablation study of how Quorum performs with different-sized subsamples (buckets). *Quorum consistently identifies subtle anomalies that the state-of-the-art method may overlook, proving to be an effective zero-training quantum solution for unsupervised anomaly detection [29], [21].*

II. Background

Before we present the design of Quorum, we first provide some brief but necessary background.

A. Quantum Computing

The fundamental unit of quantum computation is the qubit, which exists in a superposition of binary states, represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers that correspond to the amplitudes of the basis states $|0\rangle$ and $|1\rangle$. The probabilities of measuring the qubit in the $|0\rangle$ or $|1\rangle$ state are $|\alpha|^2$ and $|\beta|^2$, respectively, and these probabilities must sum to 1: $\|\alpha\|^2 + \|\beta\|^2 = 1$. Qubits can also be entangled, meaning their quantum states are correlated in such a way that they cannot be described independently of one another, which

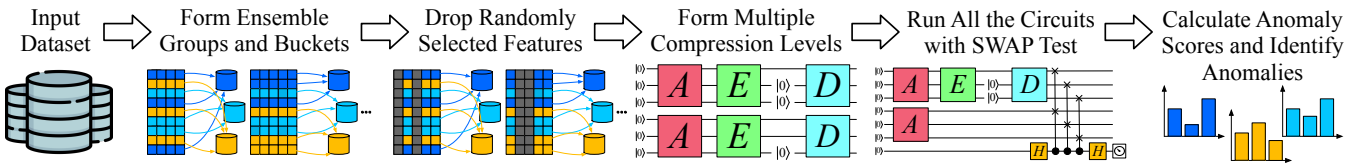


Fig. 1: Visual representation of the steps taken by Quorum to detect the anomalies in a given dataset using quantum computing.

is a key property leveraged in quantum computing applications and algorithms to achieve a quantum advantage [4].

Quantum gates, represented as unitary matrices, are the basic operations applied to qubits. Common single-qubit gates include the parameterized rotation gates, such as the RX, RY, and RZ gates, which rotate a qubit around the x, y, and z axes, respectively. Two-qubit gates, such as the controlled-X (CX) gate, are used to create entanglement between qubits. These single- and two-qubit gates are defined as:

$$RX(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \quad RY(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix},$$

$$RZ(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}, \quad CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Quantum circuits are constructed by applying sequences of these gates to qubits, followed by measurements. Due to the probabilistic nature of quantum measurements, a circuit's execution yields a probability distribution of possible outcomes, which often requires many repetitions (or shots) to obtain statistically significant results [11], [6]

B. Quantum Encoding and SWAP Test

In quantum computing, classical data must be encoded into quantum states. One common method is *Amplitude Encoding*, [35] which maps classical data points $x_0, \ldots, x_{n-1}$ to the quantum state: $|\psi\rangle = \sum_{j=0}^{n-1} x_j |j\rangle$. This method allows n classical data points to be encoded into $\log_2 n$ qubits, enabling efficient representation of high-dimensional data.

The *SWAP Test* is a quantum algorithm used to determine how similar two quantum states are [7], [22]. It computes the inner product $\langle\phi|\psi\rangle$ between two states $|\phi\rangle$ and $|\psi\rangle$. If the states are identical, the test returns a high overlap; otherwise, a lower overlap indicates dissimilarity. This is a key tool in quantum anomaly detection, where dissimilarity between two states may indicate an anomalous data point [22].

C. Anomaly Detection

Anomaly detection involves identifying data points that deviate significantly from the norm [34], [9]. In unsupervised anomaly detection, we assume no labeled data is available, and the goal is to detect anomalies based solely on patterns within the data [5], [30]. Common classical techniques include clustering and Isolation Forests. Clustering groups data points based on similarity, with anomalies detected as points far from cluster centroids [12], [1]. Isolation Forests, a tree-based algorithm, isolates anomalies by recursively splitting the data based on random feature values, where fewer splits indicate an anomaly [19], [18].

In high-dimensional spaces, autoencoders – neural networks trained to reconstruct their input – are often used for anomaly detection [23], [24]. Autoencoders learn compressed representations of normal data, and anomalies are detected when reconstruction errors for new data points are high [1], [20]. This concept extends to the quantum realm, where a quantum autoencoder can be used to compress and decompress quantum states, flagging quantum outliers based on reconstruction errors [16], [13].

III. Related Work

Several quantum-based approaches for anomaly detection have emerged recently, leveraging various quantum algorithms and machine learning techniques. Liu at al. [16] provided early evidence of the speedup and resource efficiency experienced with quantum anomaly detection. Building on this effort, Herr et al.[13] introduce quantum autoencoders trained with generative-adversarial networks. Both methods require structured queries or training processes, which prohibit their adaptability in unsupervised settings. Taking a hybrid quantum-classical approach, Sakhnenko et al. [29] propose supervised training-based solutions that require significant quantum-classical communication. Most recently, Hdaib et al.[12] propose quantum-enhanced anomaly detection with classical post-processing, still requiring circuit training.

Other works provide application-specific anomaly detection. For example, Ngairangbam et al. [21] utilize quantum classifiers that rely on supervised training for high-energy physics applications. On the other, Kukliansky et al.[14] develop quantum anomaly detectors for network anomalies. These approaches rely on domain-specific anomaly properties and are thus not generally applicable. *Thus, there is a strong need for a fully unsupervised, training-free generalized quantum anomaly detection method that leverages the unique strengths of quantum systems without the overhead of classical optimization.*

IV. Design

The design of Quorum leverages quantum dynamics to identify correlations among data features and detect anomalies by comparing quantum-transformed data samples. As shown in Fig. 1, Quorum is structured around several key stages: preprocessing, quantum embedding, bucketing, feature selection, and circuit-based statistical analysis. Each of these steps contributes to an efficient and scalable framework for anomaly detection using quantum methods.

A. Preprocessing and Normalization

Quorum begins with preprocessing the dataset, which involves a variety of steps depending on the dataset's initial

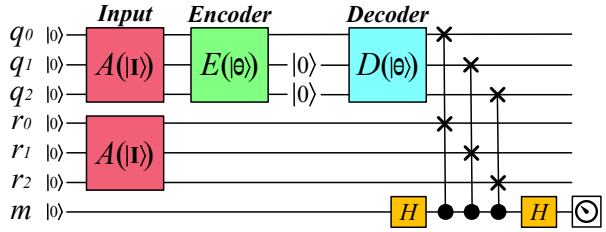


Fig. 2: Quorum's use of SWAP test to determine the similarity between the compressed and original data sample.

format. This typically includes transforming all non-numeric features into float values (e.g., via hashing), removing any label data that could indicate whether a sample is anomalous, and performing a range-based normalization. The normalization process is essential to ensure that all features contribute equally to the quantum state, which is critical for the subsequent quantum encoding.

Given a dataset with M features, Quorum normalizes each feature so that its maximum possible value is $\frac{1}{M}$. This ensures that the sum of the squares of all features for any sample does not exceed 1. The normalization is performed as follows:

$$\text{normalized feature value} = \frac{\text{raw feature value for sample}}{\text{max feature value}} \times \frac{1}{M}$$

This normalization serves two key purposes. First, it equalizes the contribution of all features to the final quantum state, preventing any feature from dominating due to its scale. Second, it simplifies the process of amplitude embedding for quantum states, as the normalized values now range between $[0, \frac{1}{M}]$ for all of the features.

B. Quantum Embedding

After normalization, Quorum embeds the data into quantum states using amplitude encoding. The normalized feature values are squared to convert them into probabilities, and an "overflow state" probability is added to account for any remaining probability mass, ensuring the total probability sums to 1. This ensures that the total probability mass of the quantum state is preserved.

A quantum circuit is then created to prepare a state vector corresponding to these probability amplitudes using amplitude embedding. This process is repeated for each data point (with 2^n features) using two sets of n qubits within the same circuit, creating two identical encodings: one for the transformation and one as a reference (Fig. 2). This dual encoding allows us to compare the transformed data with the original using a SWAP test (we'll discuss in Sec. IV-D why this is required).

Each quantum circuit consists of $2n + 1$ qubits, where the additional qubit serves as an ancilla for reading the SWAP test result. The SWAP test measures the similarity between the original and transformed quantum states, preserving the relative magnitudes of the features in the quantum state.

C. Bucketing and Feature Selection

Following the embedding process, Quorum employs a bucketing strategy and performs feature selection to prepare the data for anomaly detection. As shown in Fig. 3, the dataset is

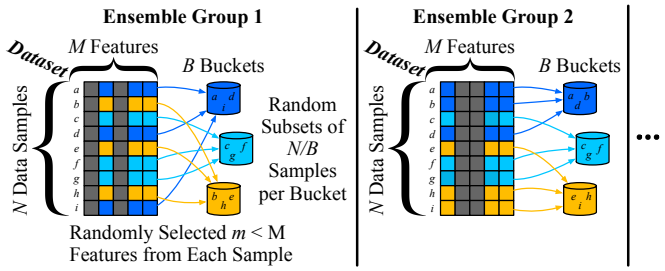


Fig. 3: Quorum's bucketing procedure for distributing the subsampled data samples across different ensemble groups.

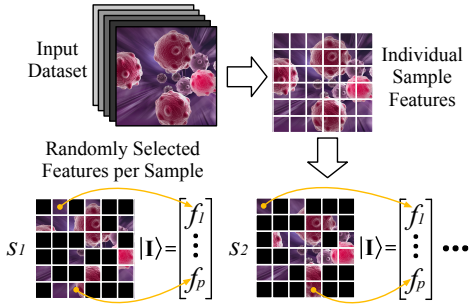


Fig. 4: Quorum subsamples features from the input space.

divided into a series of B random subsets, or buckets. Given a dataset of size N, there are N/B buckets. This bucketing strategy enhances the visibility of anomalies by allowing data points to be compared against smaller, more localized subsets of the dataset. The size of the buckets (B) is determined based on the total number of data points and the estimated proportion of anomalies, both of which determine the probability of having at least one anomaly in each bucket. By distributing the data into smaller buckets, Quorum increases the contrast between normal and anomalous points, making it easier to detect outliers (anomalous data points).

Feature selection is performed after bucketing to ensure that the data samples can fit on quantum circuits, with Quorum using a uniform random selection strategy (Fig. 4). Unlike traditional downsampling or dimensionality reduction techniques like Principal Component Analysis (PCA), random selection offers several advantages. *It is computationally faster, avoids bias towards features that might not indicate anomalies, and allows exploration of feature combinations that might otherwise be overlooked.* For each quantum circuit with n qubits, Quorum selects $m = (2^n - 1)$ features from the dataset, leaving room for an overflow state. This ensures the selected features fit within the quantum state space of n qubits.

D. Quantum Circuit Design and Ansatz

The core of Quorum's anomaly detection framework lies in the design of its quantum circuits. Unlike traditional quantum autoencoders, which train parameterized gate angles to optimize encoding-decoding processes, Quorum does not rely on learning optimal parameters. Instead, it utilizes random quantum transformations and applies statistical analysis to detect anomalies without training.

Each quantum circuit begins with an amplitude encoding of the data, as described above. The first set of encoded qubits

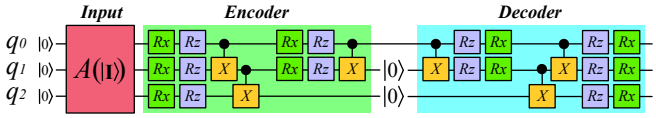


Fig. 5: The ansatz leveraged by Quorum for its autoencoder to establish correlations across input features.

is then passed through an ansatz, which consists of layers of RX and RZ rotations and CNOT gates. The ansatz performs random transformations on the encoded data, ensuring a high degree of variability in the quantum states (Fig. 5).

The ansatz includes three main components: an encoder circuit with randomly initialized parameters, a partial reset operation that simulates an information bottleneck by resetting a subset of qubits, and a decoder circuit that applies the inverse of the encoder. The random angles for the quantum gates in the encoder are initialized from a uniform distribution $U(0, 2\pi)$, and the decoder negates these angles to revert the transformations. A SWAP test is then performed between the transformed and original quantum states to measure their similarity. This similarity score forms the basis for detecting anomalies, as normal and anomalous data points will behave differently under random quantum transformations.

This design enables Quorum to compress data through the autoencoder and then decode it on the other end of the reset, with the anticipation that anomalous data would be more likely to deviate from the original state when the two states are compared using the SWAP test.

E. Ensemble Groups and Statistical Analysis

To ensure robustness, Quorum processes each data point through multiple ensemble groups. Each ensemble group involves randomly initialized quantum circuits (θs), ensuring that the data points are processed differently each time. Additionally, the dataset is divided into new buckets for each ensemble group, providing different perspectives on the data.

Each ensemble group also explores multiple compression levels (Fig. 6). The compression level, determined by the number of qubits reset in the partial reset operation, varies systematically from the highest compression (fewest qubits retained) to the lowest (most qubits retained). Quorum creates a multi-dimensional view of each data point's behavior under various quantum transformations. This approach leverages the principle of random projections, where projecting data into multiple random subspaces can reveal structural information that might not be apparent in any single projection. Each ensemble group, with its unique combination of buckets, feature subsets, circuit parameters, and compression levels, represents a different "quantum projection" of the data.

Statistical analysis is then performed on the SWAP test outputs. *A key innovation in Quorum's approach lies in how these quantum circuit outputs are analyzed to detect anomalies.* For each bucket and each run, the mean and standard deviation of the SWAP test results are calculated. The anomaly score for each data point is derived by calculating the normalized deviation from the bucket mean divided by the standard deviation. These deviations are summed across

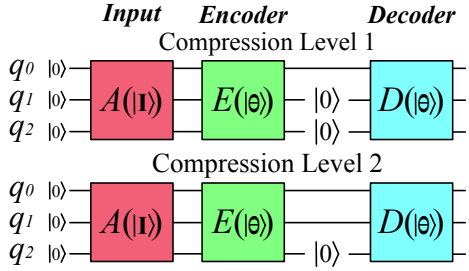


Fig. 6: Quorum leverages multiple compression levels across different ensemble groups to improve anomaly detection.

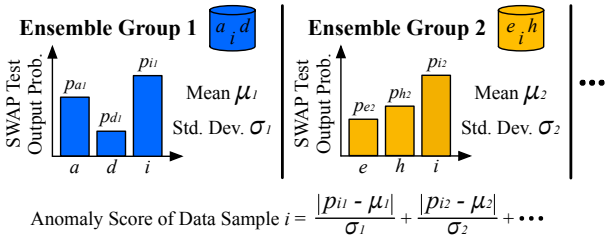


Fig. 7: Quorum's anomaly score calculates the statistical deviation of a data sample across ensemble groups.

all runs and buckets, producing an overall anomaly score for each data point (Fig. 7). A higher score indicates a greater likelihood of an anomaly.

The use of random angles in the quantum circuit is a critical aspect of Quorum's design. Rather than learning to reconstruct inputs accurately, this approach creates a random projection of the data in high-dimensional Hilbert space. The statistical analysis then captures how differently each data point behaves under these random quantum transformations compared to the average behavior of points in its various buckets.

This design allows Quorum to detect anomalies without explicitly learning the structure of normal data or performing any optimization, instead relying on the statistical properties of how data points respond to random quantum transformations. *The effectiveness of this approach lies in the fact that anomalous data points tend to behave differently under these transformations compared to normal data points, as the randomizations add more deviation to anomalous data compared to normal data.* By aggregating these behaviors across multiple random initializations, buckets, and compression levels, Quorum aims to build a comprehensive profile of each data point's response to quantum transformations, potentially unveiling subtle anomalies that more conventional techniques might miss.

F. Scalability and Flexibility

A significant advantage of Quorum's design is its inherent scalability and flexibility. While our initial experiments utilized 3-qubit encodings (resulting in 7-qubit circuits), the approach can be readily scaled to accommodate larger encodings. For instance, moving to 4-qubit or higher encodings would introduce additional "moments" to our results for each compression level, potentially capturing even more nuanced relationships in the data. This scalability allows users to tailor

TABLE I: Datasets used for Quorum's evaluation. The rightmost column refers to the likelihood of at least one anomaly being placed in each bucket.

Dataset	Samples	Anomalies	Features	Pr [Anomaly ∈ Bucket]
Breast Cancer	367	10	30	0.75
Pen-Global	809	90	16	0.6
Letter	533	33	32	0.95
Power Plant	1,000	30	5	0.75

the depth and complexity of the quantum transformations to their specific needs and computational resources.

Furthermore, the design of Quorum lends itself to extensive parallelization. Each ensemble group is entirely independent of the others, making the technique embarrassingly parallel. The scalability of Quorum extends beyond just increasing the number of qubits or ensemble groups. The flexibility in choosing the number of compression levels, the size of buckets, and the number of features selected allows users to fine-tune the balance between computational cost and the granularity of anomaly detection.

In the following section, we delve into the methodology used to obtain our evaluated results.

V. Experimental Methodology

Experimental Setup. We evaluate Quorum using Qiskit Aer's quantum circuit simulator. We use Python 3.10.12 and Qiskit 1.2, IBM's quantum computing language [3], which is used to run simulations of quantum circuits locally. Each quantum circuit is generated via a Qiskit QuantumCircuit object, and the quantum circuits are run through noiseless simulations. We run simulations with Quorum and competitors on a local research cluster with Ubuntu 22.04.2 LTS on a 32-core 2.0 GHz AMD EPYC 7551P processor with 32 GB RAM.

We also perform noisy simulations, where we model our hardware after IBM's Brisbane quantum computer to provide realistic error rates. The noise parameters were obtained directly from IBM's Brisbane backend specifications; median properties are as follows: coherence times ($T_1 = 230.42\ \mu$s, $T_2 = 143.41\ \mu$s), gate errors (single-qubit SX gate error = 2.274×10^{-4}, two-qubit gate error = 2.903×10^{-3}), and readout error (1.38×10^{-2}). Note: due to the over 100,000 runs required for Quorum's evaluation, it was cost-prohibitive to execute on real hardware. Thus, we use faithful simulations.

Datasets. We evaluate Quorum on four distinct datasets representing different anomaly detection scenarios (Table I). Three of the four datasets are directly derived from a related and widely-cited survey of classical unsupervised anomaly detection techniques by Goldstein and Uchida [10]. The fourth dataset, which contains measurements taken from a combined cycle power plant, was taken from UCI's machine learning repository [32]. For the power plant dataset, we inserted "plausible" anomalies into the dataset based on ranges of values that are possible for each feature. All datasets have labels stripped for all operations until the evaluation is performed to facilitate unsupervised anomaly detection.

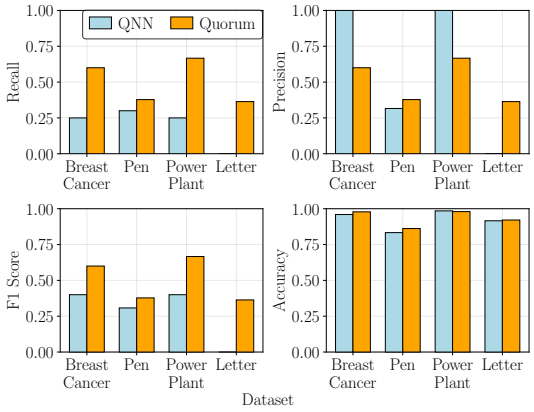


Fig. 8: Comparison of performance metrics between QNN and Quorum across four datasets. Note that the QNN did not detect any anomalies for the letter dataset, and thus, precision, recall, and F1 scores are all zero.

Experimental Framework. Each experiment consists of multiple ensemble groups, where an ensemble group represents a complete run of Quorum with different random initializations. We use 3-qubit encodings for our primary experiments, resulting in 7-qubit circuits (including the ancilla qubit). We chose this circuit size mainly due to limitations on our computational resources as we ran simulations on our local systems. For each dataset, we execute with 1,000 ensemble members, with each member using different random bucket assignments and feature selections. We executed 4,096 shots per circuit for measurements. Increasing both shot count and ensemble members has significant impacts on performance, with benefits diminishing as they increase past a certain point. For noisy simulations, we similarly executed 4,096 shots per circuit. We use different target probabilities for bucket size determination (see Table I) and provide an ablation study on the effect of bucket sizes for each of the different datasets.

Evaluation Metrics. We evaluate Quorum's performance using several metrics: (1) **Detection Rate/Accuracy** at various percentile thresholds, measuring the fraction of true anomalies captured in the top k% of anomaly scores. We chose to include this metric to showcase how well our technique separates anomalies from the rest of the dataset. (2) **Precision**, calculated as the ratio of correctly identified anomalies to the total number of samples flagged as anomalous. (3) **Recall**, measured as the ratio of correctly identified anomalies to the total number of true anomalies in the dataset. And (4) **F1 Score**, which is the harmonic mean of precision and recall.

Competitive Techniques. We compare Quorum against a state-of-the-art quantum anomaly detection technique that utilizes quantum neural networks to find anomalies in labeled datasets [14] (we refer to this technique as "QNN"). The technique is the most suitable among competitors as it shows improved performance over them and is designed for noisy quantum hardware, making it a practical benchmark. We adapted QNN for generic use since it was initially developed for network anomaly detection. Further, comparing against

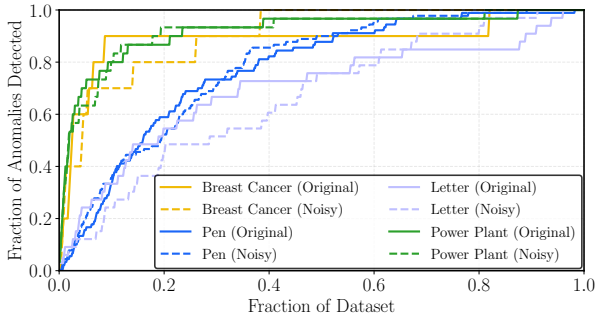


Fig. 9: Quorum groups anomalies consistently amongst data points with the highest absolute average deviations.

QNN's supervised, training-based approach helps demonstrate the advantages of our zero-training unsupervised method, particularly in scenarios where labeled data is unavailable. Note: no unsupervised, zero-training work exists to compare against as Quorum is the first of its kind in this domain.

VI. Evaluation

Our experimental results demonstrate the efficacy of Quorum's anomaly detection approach across diverse datasets.

Flagship Results. Quorum demonstrates balanced anomaly detection performance across all evaluated datasets – Fig. 8. The recall measurements particularly highlight Quorum's strengths, where it consistently outperforms the QNN method. While the QNN achieves perfect precision scores on both the breast cancer and power plant datasets, this comes at the significant cost of being overly conservative in anomaly detection, leading to its notably poor recall performance. Quorum thus achieves superior F1 performance across every dataset tested – 23% higher on average. Quorum's more nuanced approach leads to better overall detection capabilities while maintaining comparable accuracy levels across all datasets. *These results demonstrate that Quorum provides an effective approach to anomaly detection, successfully balancing precision and recall without sacrificing overall classification performance.*

Detection Rates. In Fig. 9, Quorum's detection rate curves exhibit notably steep initial gradients, particularly for the breast cancer and power plant datasets, which achieve approximately 80% detection rate within the first 10% of the highest deviation samples in noiseless simulations (see Fig. 10 for a detailed look at how Quorum grouped the breast cancer data samples). The performance hierarchy among datasets likely reflects their inherent separability characteristics, with the breast cancer dataset showing the most distinctive anomaly patterns (achieving near-perfect detection at the 10th percentile), followed by power plant data. The letter and pen datasets, while requiring a larger percentile of samples for complete detection, still maintain clear separation from random performance, achieving a roughly 60% detection rate within the top 20% of sample deviations for noiseless runs. When subjected to realistic quantum noise, Quorum demonstrates a high degree of resilience. Noisy simulations closely track their noiseless counterparts across all datasets, with only minimal degradation in performance.

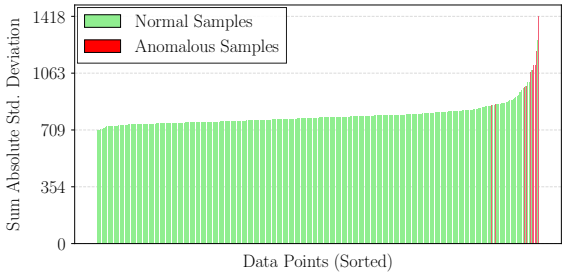


Fig. 10: An example of how Quorum separates anomalies from normal samples on the breast cancer dataset (16K shots).

TABLE II: F1 Scores for different bucket sizes (p is the probability of at least one anomaly in a bucket).

Dataset	$p = 0.5$	$p = 0.6$	$p = 0.75$	$p = 0.95$	$p = 0.98$
Breast Cancer	0.500	0.500	0.600	0.500	0.600
Pen Digits	0.333	0.389	0.367	0.389	0.389
Letter	0.152	0.182	0.242	0.273	0.273
Power Plant	0.600	0.600	0.633	0.533	0.600

Such inherent noise resilience is a significant advantage of Quorum for near-term applications, as Quorum can maintain its effectiveness even on noisy hardware without requiring high-overhead error mitigation or correction methods.

Bucket Size Ablation. Analysis of F1 scores across different bucket size configurations in Table II reveals that, as expected, very small bucket sizes generally lead to degraded performance. However, we observe that moderately sized buckets oftentimes outperform larger ones – for instance, the letter dataset achieves its peak F1 score of 0.273 at $p = 0.95$, while the breast cancer and power plant datasets show improved performance at $p = 0.75$. This behavior can be explained by the trade-off between statistical significance and local sensitivity - while larger buckets provide more robust statistical estimates, smaller buckets may better capture localized anomaly patterns by reducing the "averaging out" effect of mixing anomalous and normal samples from the datasets.

VII. Conclusion

In this paper, we introduced Quorum, a novel quantum anomaly detection framework that operates without requiring any training. The framework's design, which incorporates strategic data bucketing, feature selection, and ensemble analysis, proves particularly effective at identifying anomalies while remaining computationally efficient through its inherent parallelizability. Our evaluation shows that Quorum can achieve up to 80% detection rate within the first 10% of highest-deviation samples, demonstrating strong anomaly-detection power, and high resilience to noise on near term quantum systems. These results suggest that Quorum represents a promising and wholly quantum step forward in anomaly detection.

Acknowledgement. This work was supported by Rice University and the DOE Quantum Testbed Finder Award DE-SC0024301. It was also supported by the Ken Kennedy Institute and Rice Quantum Initiative, which is part of the Smalley-Curl Institute. Quorum's code is open-sourced at `https://github.com/positivetechnologylab/Quorum`.

REFERENCES

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[2] Javier Alcazar, Vicente Leyton-Ortega, and Alejandro Perdomo-Ortiz. Classical versus quantum models in machine learning: insights from a finance application. *Machine Learning: Science and Technology*, 1(3):035003, 2020.

[3] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. Qiskit: An Open-source Framework for Quantum Computing, January 2019.

[4] MD Arafath and A Niranjil Kumar. Quantum computing based neural networks for anomaly classification in real-time surveillance videos. *Computer Systems Science & Engineering*, 46(2), 2023.

[5] Sounak Bhowmik and Himanshu Thaplival. Quantum machine learning for anomaly detection in consumer electronics. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 544–550. IEEE, 2024.

[6] Giuseppe Bisicchia, Jose García-Alonso, Juan M. Murillo, and Antonio Brogi. Distributing quantum computations, by shots. In Flavia Monti, Stefanie Rinderle-Ma, Antonio Ruiz Cortés, Zibin Zheng, and Massimo Mecella, editors, *Service-Oriented Computing*, pages 363–377, Cham, 2023. Springer Nature Switzerland.

[7] Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Phys. Rev. Lett.*, 87:167902, Sep 2001.

[8] Wisam Elmasry and Mohammed Wadi. Enhanced anomaly-based fault detection system in electrical power grids. *International Transactions on Electrical Energy Systems*, 2022(1):1870136, 2022.

[9] Joseph A Gallego-Mejia, Oscar A Bustos-Brinez, and Fabio A González. Inqmad: incremental quantum measurement anomaly detection. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 787–796. IEEE, 2022.

[10] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 11(4):1–31, 04 2016.

[11] Mayank Gupta and Manisha J. Nene. Quantum computing: A measurement and analysis review. *Concurrency and Computation: Practice and Experience*, 33(20):e6344, 2021.

[12] Moe Hdaib, Sutharshan Rajasegarar, and Lei Pan. Quantum autoencoder frameworks for network anomaly detection. In *International Conference on Neural Information Processing*, pages 69–82. Springer, 2023.

[13] Daniel Herr, Benjamin Obert, and Matthias Rosenkranz. Anomaly detection with variational quantum generative adversarial networks. *Quantum Science and Technology*, 6(4):045004, 2021.

[14] Alon Kukliansky, Marko Orescanin, Chad Bollmann, and Theodore Huffmire. Network anomaly detection using quantum neural networks on noisy quantum computers. *IEEE Transactions on Quantum Engineering*, 2024.

[15] W Bernard Lee and Anthony G Constantinides. Computational results for a quantum computing application in real-life finance. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 414–423. IEEE, 2023.

[16] Nana Liu and Patrick Rebentrost. Quantum machine learning for quantum anomaly detection. *Physical Review A*, 97(4):042315, 2018.

[17] Javier Mancilla and Christophe Pere. A preprocessing perspective for quantum machine learning classification advantage in finance using nisq algorithms. *Entropy*, 24(11):1656, 2022.

[18] Elisa Marcelli, Tommaso Barbariol, Davide Sartor, and Gian Antonio Susto. Active learning-based isolation forest (alif): Enhancing anomaly detection with expert feedback. *Information Sciences*, page 121012, 2024.

[19] Antonella Mensi, Alessio Franzoni, David MJ Tax, and Manuele Bicego. An alternative exploitation of isolation forests for outlier detection. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops, S+ SSPR 2020, Padua, Italy, January 21–22, 2021, Proceedings*, pages 34–44. Springer, 2021.

[20] Naji Najari, Samuel Berlemont, Grégoire Lefebvre, Stefan Duffner, and Christophe Garcia. Robust variational autoencoders and normalizing flows for unsupervised network anomaly detection. In *International Conference on Advanced Information Networking and Applications*, pages 281–292. Springer, 2022.

[21] Vishal S Ngairangbam, Michael Spannowsky, and Michihisa Takeuchi. Anomaly detection in high-energy physics using a quantum autoencoder. *Physical Review D*, 105(9):095004, 2022.

[22] Vishal S. Ngairangbam, Michael Spannowsky, and Michihisa Takeuchi. Anomaly detection in high-energy physics using a quantum autoencoder. *Phys. Rev. D*, 105:095004, May 2022.

[23] Oyebade K Oyedotun and Djamila Aouada. A closer look at autoencoders for unsupervised anomaly detection. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3793–3797. IEEE, 2022.

[24] Krishna Patra, Rabi Narayan Sethi, and Dhiren Kkumar Behera. Anomaly detection in rotating machinery using autoencoders based on bidirectional lstm and gru neural networks. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30(4):1637–1653, 2022.

[25] Marco Pistoia, Syed Farhan Ahmad, Akshay Ajagekar, Alexander Buts, Shouvanik Chakrabarti, Dylan Herman, Shaohan Hu, Andrew Jena, Pierre Minssen, Pradeep Niroula, et al. Quantum machine learning for finance iccad special session paper. In *2021 IEEE/ACM international conference on computer aided design (ICCAD)*, pages 1–9. IEEE, 2021.

[26] Tahereh Pourhabibi, Kok-Leong Ong, Booi H. Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, 2020.

[27] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[28] John Preskill. Quantum computing 40 years later. In *Feynman Lectures on Computation*, pages 193–244. CRC Press, 2023.

[29] Alona Sakhnenko, Corey O'Meara, Kumar JB Ghosh, Christian B Mendl, Giorgio Cortiana, and Juan Bernabé-Moreno. Hybrid classical-quantum autoencoder for anomaly detection. *Quantum Machine Intelligence*, 4(2):27, 2022.

[30] Julian Schuhmacher, Laura Boggia, Vasilis Belis, Ema Puljak, Michele Grossi, Maurizio Pierini, Sofia Vallecorsa, Francesco Tacchino, Panagiotis Barkoutsos, and Ivano Tavernelli. Unravelling physics beyond the standard model with classical and quantum anomaly detection. *Machine Learning: Science and Technology*, 4(4):045031, 2023.

[31] Yehui Tang, Junchi Yan, Guoqiang Hu, Baohua Zhang, and Jinzan Zhou. Recent progress and perspectives on quantum computing for finance. *Service Oriented Computing and Applications*, 16(4):227–229, 2022.

[32] Pnar Tfekci and Heysem Kaya. Combined Cycle Power Plant. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5002N.

[33] Sohum Thakkar, Skander Kazdaghli, Natansh Mathur, Iordanis Kerenidis, André J Ferreira-Martins, and Samurai Brito. Improved financial forecasting via quantum machine learning. *Quantum Machine Intelligence*, 6(1):27, 2024.

[34] Maida Wang, Anqi Huang, Yong Liu, Xuming Yi, Junjie Wu, and Siqi Wang. A quantum-classical hybrid solution for deep anomaly detection. *Entropy*, 25(3):427, 2023.

[35] M. Weigold, J. Barzen, F. Leymann, and M. Salm. Encoding patterns for quantum algorithms. *IET Quantum Communication*, 2:141–152, 2021.

[36] Cuicui Zhang, Jiali Sun, Ruixuan Lu, and Peng Wang. Anomaly detection model of power grid data based on stl decomposition. In *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, volume 5, pages 1262–1265, 2021.

InterConFuzz: A Fuzzing-based Comprehensive NoC Verification Framework

Samit Shahnawaz Miftah
Dept of Electrical and Computer Engineering
University of Texas at Dallas
Dallas, Texas, USA
samit.miftah@utdallas.edu

Hyunmin Kim
Technology Innovation Institute
Abu Dhabi, UAE
hyunmin.kim@tii.ae

Kanad Basu
Dept of Electrical and Computer Engineering
University of Texas at Dallas
Dallas, Texas, USA
kanad.basu@utdallas.edu

***Abstract*—Security verification of Network-on-Chip (NoC) systems is essential due to their intricate and high-concurrency structures. Traditional methods often fail to cover all scenarios or scale effectively, leading to prolonged verification and overlooked vulnerabilities. Our proposed solution, *InterConFuzz*, a hybrid hardware fuzzing technique, uses symbolic execution for extensive coverage. Developed on Universal Verification Methodology (UVM), *InterConFuzz* discovered five security flaws in the NoC architecture of the OpenTitan SoC—surpassing existing techniques by three—while reducing memory and computational needs by 24.4% and 29.5%, respectively. Furthermore, *InterConFuzz* furnished comparable functional coverage compared to existing NoC fuzzing approaches, proving its efficiency and robustness.**

***Index Terms*—Hardware Security, Network-on-Chip, Security Verification, Fuzzing**

I. Introduction

The increasing complexity of modern System-on-Chip (SoC) designs stems from the integration of a growing number of Intellectual Property (IP) blocks and sophisticated communication protocols that allow these components to interact efficiently. This complexity is managed by Network-on-Chip (NoC) systems, which are responsible for coordinating data flow, resource sharing, and communication across the SoC. While NoCs enhance performance and modularity, they also introduce potential security vulnerabilities due to the intricate interactions between multiple processing units. Flaws or weaknesses in NoC designs can expose sensitive data or lead to malfunctioning components, resulting in critical system failures. Such vulnerabilities and functional errors can have far-reaching consequences, including delays in product development and significant financial losses due to expensive silicon respins and redesigns, with over 70% of respins attributed to such issues [1], [2]. Therefore, comprehensive NoC verification is essential to identify and mitigate potential issues early in the design process, ensuring both the functional integrity and security of SoCs.

Despite advancements in fault-tolerant designs, the verification of NoC systems remains a significant challenge, particularly in ensuring that routing errors and communication faults are detected and mitigated. While simulation has been the traditional approach for verifying NoC designs, it is often insufficient for capturing the full scope of potential errors, especially as NoC systems become more intricate. Formal methods, although effective at verifying abstract models of NoC, struggle with the scalability and complexity of verifying Register Transfer Level (RTL) implementations. As routers interconnect to form larger networks, the verification of global NoC properties becomes exponentially more difficult, highlighting the need for more robust verification strategies.

This research was supported by the Technology Innovation Institute (TII), Abu Dhabi, UAE.

Nevertheless, given the limitations of formal methods in verifying larger and more complex NoC designs, simulation-based methodologies have become the preferred approach. In this context, the Universal Verification Methodology (UVM) is the industry-standard library for verification [3]–[5]. UVM provides a scalable and reusable framework that simplifies the generation of test cases and coverage analysis, thereby reducing the complexity associated with verifying intricate NoC architectures. However, the reliance on manually generated test cases still demands substantial human effort, making it challenging to achieve full coverage. As NoC designs continue to grow in complexity, the inefficiency of generating comprehensive test cases through manual methods becomes increasingly apparent.

Software fuzzing tools like libFuzzer [6] and AFL [7] have shown great promise in validating large and complex programs. When adapted for hardware verification, fuzzing has demonstrated significant potential [8]–[11]. However, hardware fuzzing techniques face several challenges that hinder their efficacy. These include the need for robust coverage monitoring to accurately measure and guide exploration, the challenge of efficiently directing the fuzzer to uncover new states without incurring excessive computational overhead, and the requirement for a reliable reset mechanism to ensure deterministic testing and manage the complex reset hierarchies of SoCs [11]. Furthermore, simulation tools like Verilator [12], which are often used in hardware fuzzing, present limitations in terms of scalability and compatibility with industrial verification environments. Another critical challenge is ensuring that the fuzzer can capture all hardware events, such as delays and clock generation in HDL, during simulation, as incomplete event tracking can lead to gaps in the verification process.

To address these challenges, we developed the *Interconnect Fuzzer* (*InterConFuzz*), a novel hybrid NoC fuzzing approach built on top of UVM that enhances coverage and operational efficiency while mitigating the limitations of existing simulation tools. Verilator [12], which cannot capture all hardware events, posed significant limitations. To overcome this, we opted for Vivado for simulation [13]. Although Vivado is slower than Verilator, it accurately simulates and captures critical hardware-specific events such as delays and clock generation, ensuring comprehensive event tracking during NoC architecture fuzzing. Furthermore, *InterConFuzz* employs a white-box verification model, enabling direct observation of internal variables within the NoC architecture, thus facilitating precise coverage monitoring and effective state tracking. A control flow graph (CFG) of the NoC is generated to capture branching information, guiding the placement of checkpoints that optimize state exploration. Integrating reset logic derived from the CFG into the RTL design reduces the need for frequent hard resets, ensuring deterministic tests while maintaining computational efficiency. *Moreover, InterConFuzz*

utilizes a two-step guidance mechanism that assesses the feasibility of reaching new states from the current state and employs the CFG to generate tailored seeds for exploring previously uncharted nodes of the NoC CFG. By addressing these challenges, including the need for comprehensive event tracking and the limitations of simulation tools, *InterConFuzz* significantly improves the scalability and effectiveness of NoC fuzzing in industrial verification settings. In this paper, our main contributions are:

- We present *InterConFuzz*, a fuzzing-based NoC verification methodology built upon UVM that systematically encompasses all states associated with the defined security properties. This facilitates its seamless integration into the hardware security development lifecycle (SDL).
- We develop a checkpoint mechanism using a reset operation, which reduces computational cost for executing the proposed NoC fuzzing approach. This enables partial reset to speed up the path exploration process through fuzzing.
- *InterConFuzz* is evaluated on the TL-UL NoC architecture used in the OpenTitan SoC [14], where it was able to identify five types of vulnerabilities, three more than the existing NoC verification methodology typically present in the NoC architectures.
- *InterConFuzz* demonstrates similar coverage gain with low variance compared to existing fuzzing-based verification approaches.

II. Background

A. Coverage-guided Fuzzing

Fuzzing generates tests to probe valid and invalid edge cases, identifying vulnerabilities by uncovering undefined behaviors [15], [16]. It uses mutation-based techniques, altering existing data, or generation-based methods, creating tests from protocols. Widely adopted, platforms like OSS-Fuzz and ClusterFuzz automate fuzzing, revealing security flaws in large codebases [17]–[22]. Coverage-guided fuzzing explores diverse program paths, leveraging run-time coverage data via instrumentation to refine inputs [23]–[26].

B. Symbolic Execution

Symbolic execution is a powerful technique for analyzing the relationship between program inputs and behavior [27]–[29]. Instead of concrete values, it uses symbolic representations for inputs, allowing the interpreter to create expressions that incorporate these symbols. Conditional branches are associated with constraints derived from symbolic values, representing the conditions under which different paths are executed. By solving these constraints, symbolic execution reveals how various inputs influence program execution, identifying which inputs lead to specific execution paths.

C. Universal Verification Methodology (UVM)

The UVM is a standardized framework based on SystemVerilog, designed to improve the verification process for complex digital designs, such as SoC architectures [3]. UVM promotes reusability, scalability, and flexibility by leveraging object-oriented programming principles, enabling the creation of reusable verification components like drivers and monitors. Its importance stems from its ability to streamline verification through constrained-random testing and functional coverage-driven techniques, ensuring comprehensive exploration of a design's behavior under various conditions. UVM's adoption across the verification industry is critical as it addresses the increasing complexity of modern designs while reducing development time and cost by reusing verification IPs. This aligns with industry demands for faster time-to-market and improved product reliability [3], [30], [31]. By offering a scalable and automated verification environment, UVM plays a crucial role in ensuring the functional correctness and robustness of semiconductor products. This is crucial since verification consumes a significant portion of the design flow.

III. Related Works

NoC architectures can be vulnerable to breaches with minimal impact [32], [33]. To secure SoC communication, methods like secure NoC fabric [34] and Design-for-Debug (DfD) features [35] are used. However, these security components can increase power and area costs, challenging the compact and energy-efficient design of SoCs.

Real-time detection and localization of denial of service (DoS) attacks have been proposed to prevent adversaries from exploiting the NoC architecture to execute DoS attacks on the SoC [36]. For example, network-based monitoring mechanisms utilizing unique hardware probes have been proposed to enable runtime observability of system-level security threats [37]. Furthermore, recent studies have explored methods for detecting NoC-based hardware Trojans at runtime [33], [38]. While various verification methods such as concolic testing [39], [40], fuzzing [8]–[10], information flow tracking [41]–[44], and runtime detection [45]–[47] have been developed for SoCs and IPs, few attempts have been made to develop a security verification method specifically tailored for NoC.

Despite these advances in ensuring security for NoCs, most techniques require architectural implementations to enforce security but lack verification strategies for assessing NoC security at the register-transfer level (RT-level). To address this gap, two methodologies have been developed for verifying NoC security: SeVNoC and NoC-Fuzzer [48], [49]. SeVNoC employs control flow graphs (CFGs) to identify vulnerabilities in inter-IP communication, while NoCFuzzer uses fuzzing techniques to verify the security of NoCs. However, these verification tools show limitations in coverage and bug detection capability as discussed in Section V.

IV. Proposed InterConFuzz Methodology

A. InterConFuzz Architecture

The *InterConFuzz* framework, illustrated in Fig. 1, integrates seamlessly into the UVM workflow for NoC security verification. The individual components are explained in the next paragraph; while a standard UVM verification environment consists of components ① to ⑥, *InterConFuzz* extends this with five additional blocks (⑦ to ⑪) to enhance the process through symbolic execution and input fuzzing. The fuzzing is introduced using UVM library functions, while symbolic execution is achieved with an SMT solver.

In a basic UVM verification workflow, the *Scoreboard* or ① is responsible for comparing the expected and actual outputs of the design under verification (DUV) ⑥ to verify its correctness. It typically receives predicted values from reference models and the actual values from the DUV, subsequently flagging any mismatches as errors. This ensures that the DUV behaves as intended according to the testbench specifications. The *UVM Sequencer* or ② generates and controls the flow of transaction-level stimuli to ⑥, managing the order and timing of test scenarios. It acts as a bridge between the testbench and the driver, ensuring proper data flow. The *UVM Driver* or ③ takes transactions from the sequencer and translates them into signal-level inputs for the DUV. It ensures that high-level commands are converted into physical stimuli, enabling the DUV to process the intended test cases. The *UVM Monitor* or ④ passively observes the DUV's interfaces, capturing signal activity and transaction data for analysis. It checks the DUV's behavior without interference and reports information for functional coverage

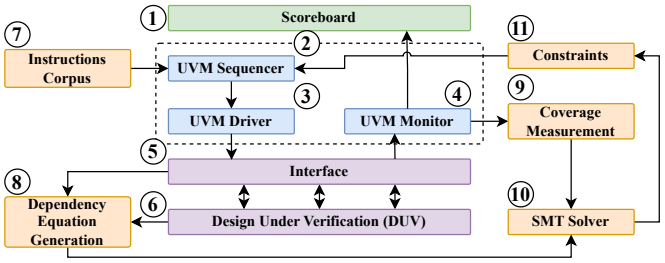


Fig. 1: *InterConFuzz* Architecture.

and checking, contributing to verification results. (6) is connected to (3) and (4) through the interface (5).

An instruction corpus (7), containing the necessary NoC function instructions, is compiled to verify the security of the NoC. This corpus is then linked with (2), enabling it to generate pattern sequences based on the corpus.

Dependency equations are extracted from the DUV through the Control Flow Graph (CFG) of the RTL design, which is generated using PyVerilog [50] to enable symbolic execution and enhance coverage. This process is illustrated by the connection of (8) with (5) and (6). These equations describe a Boolean relationship between the control registers and the I/O ports of the interface. The equations are then processed by the *z3-solver* [51] (10), an SMT solver, to generate and solve constraints (11). Additionally, a coverage measurement module (9) is linked to (4), evaluating the control register values to assess the achieved coverage. If the coverage remains stagnant for a user-specified duration, (9) triggers (10), which produces new constraints (11) to be integrated into (2).

B. InterConFuzz Algorithm

1) Defining UVM Sequencer and Driver: To randomize the instruction sequence for the NoC input, *InterConFuzz first defines a UVM sequencer tailored to the NoC architecture's specific instruction corpus.* The instructions are categorized based on their type, and each category's bit sequence is divided into components representing various commands, such as source address, destination address, opcode, flags, and more. For example, in the TL-UL bus protocol, the opcode is encoded in three bits, with an additional three bits allocated for parameters, a flag indicating data corruption, and such [52]. The protocol supports five message types: (1) *Get*, (2) *PutFullData*, (3) *PutPartialData*, (4) *AccessAck*, and (5) *AccessAckData*.

To execute specific functions, *InterConFuzz* begins by randomizing the input message sequences. Next, it identifies bit value packets such as addresses, data, and other fields. To perform fuzzing, *InterConFuzz* randomizes the values of these bit fields, generating diverse event scenarios while preserving the sequence structure. These modified inputs are then used by the UVM sequencer to drive the DUV with both functional and fuzzed input patterns.

To set up a complete UVM testbench, *InterConFuzz* first extracts the design interface from the NoC architecture, as described in detail in Algorithm 1, discussed in Section IV-B3. Upon defining the UVM sequencer, *InterConFuzz* attaches the sequencer to the UVM driver, and subsequently, the driver is connected to the design through the extracted interface. This integration establishes a fully functional UVM testbench, allowing for thorough testing of the NoC design.

2) Coverage Assessment: InterConFuzz uses control registers to measure coverage during fuzzing. Control registers dictate system behavior by controlling state transitions and monitoring specific operational parameters, enabling precise tracking in processes like

Algorithm 1: *InterConFuzz* Checkpoint Setup Algorithm.

Input: RTL Design, $\mathcal{D}$
Parameter: Interval, $\mathcal{I}$; Exit Threshold, Th
Output: Control Flow Graph, `CFG`; CheckPoints, `ChckPoints`

```
IF ← Extract(D)            /* IF = Interface */
CFG ←Generate— D;
Identify Branching Nodes;
Mark Control Registers;
for each Control Register, Reg_C do
    if branching at Reg_C > Br_th then
        append Reg_C to ChckPoints;
    end
end
Attach ChckPoints to CFG;
return CFG, ChckPoints
```

CFG traversal for coverage measurement. These registers offer a more accurate way to track coverage because they allow for precise monitoring of explored paths within the CFG, effectively capturing the test's progress and addressing coverage challenges commonly encountered in simulation.

Coverage is measured by running a simulation and analyzing the resulting dump file to see which paths were explored. Control registers are carefully monitored throughout this process as they define the sequence of states and transitions during the simulation. Since individual register values alone do not capture the full simulation dynamics—given the interactions between different control registers—it is essential to consider all possible combinations of these values. This approach allows a comprehensive measurement of coverage, with the total number of states calculated based on these register interactions.

To evaluate fuzzing coverage, *InterConFuzz* leverages control registers by generating a CFG from the RTL design. These control registers, which guide traversal through the CFG, enable precise coverage measurement, addressing associated challenges effectively. Following the simulation, the resulting dump file is analyzed to identify the paths explored during the test. *To ensure comprehensive state exploration, it is crucial to monitor these registers since they dictate the simulation's trajectory. Observing individual register values alone is insufficient due to potential interactions; therefore, all combinations of their values must be considered.* The total number of possible states is determined using the Eqn. 1.

$$No.\ of\ total_states = \prod_{j=1}^{N} (n_j) \tag{1}$$

In Eqn. 1, N denotes the total number of control registers, with n_j representing the possible states for control register j. Coverage is measured by comparing the total number of possible states to the number of states that have been reached.

3) Simulation Setup: InterConFuzz utilizes Algorithm 1 to configure the simulation environment. In this process, *InterConFuzz* simulates the DUV with a specified interval, $\mathcal{I}$, and an exit threshold, Th. $\mathcal{I}$ determines the number of clock cycles over which the simulation runs before recording the simulation outcome as a VCD file. The exit threshold, Th, specifies the maximum number of consecutive simulation dumps during which the DUV can remain in the same

Algorithm 2: *InterConFuzz* CFG Traversal Algorithm.

Input: Interface, `IF`; Control Flow Graph, `CFG`; CheckPoints, `ChckPoints`;
Parameter: Interval, $\mathcal{I}$; Exit Threshold, Th
Output: Report, $\mathcal{R}$

```
1  while All ChkPoints not Covered do
2      SimFile <-Dump VCD- Simulate(D) ;
3      Coverage <-Record- Read(SimFile) ;
4      if All states covered then
5          mark current ChckPoint as covered
6      end
7      if NoIncrement(Coverage) > Th then
8          identify(last covered state);
9          find (nearest ChckPoint);
10         while No New State can be reached do
11             find (previous ChckPoint in the CFG);
12         end
13         reset back to the ChckPoint;
14         Constraints ← SolveFor(newStateCondition);
15         Sequencer <-Apply- Constraints;
16     end
17     if Bug found then
18         R ← B_det        /* Bug Details, B_det, contain
               clock cycle time and violated property   */
19     end
20 end
21 return R
```

state; exceeding this limit prompts *InterConFuzz* to initiate symbolic execution for enhanced guidance. After completing the simulation, *InterConFuzz* produces a comprehensive bug report for the design.

InterConFuzz takes the RTL design of the NoC, $\mathcal{D}$, as its input. Initially, *InterConFuzz* extracts the I/O interface of $\mathcal{D}$ (line 1), enabling the attachment of the UVM driver to the RTL design. Following this, *InterConFuzz* generates the `CFG` of $\mathcal{D}$ (line 2) and identifies the branching nodes within this `CFG` (line 3). Subsequently, *InterConFuzz* identifies the control registers by determining those registers that influence the execution flow of the DUV (line 4). Using the `CFG`, *InterConFuzz* detects branch points and compiles a list of registers governing these branches. These registers are then classified as control registers and are employed to evaluate coverage metrics.

To avoid performing a full reset and ensure the fuzzer revisits specific points in the RTL design, InterConFuzz introduces checkpoints within the design's `CFG`. These checkpoints are strategically placed based on the design's branching structure. *InterConFuzz* selects nodes as checkpoints when the branching factor exceeds a predefined threshold, Br_{th}. Specifically, if a node and its subsequent sub-nodes generate more than Br_{th} branches, that node is marked as a checkpoint (lines 6-8). The threshold Br_{th} is configurable by the user: setting a higher Br_{th} results in fewer checkpoints, causing the fuzzer to revisit the same branches more often, while a lower Br_{th} increases the number of checkpoints, which demands more memory. Experimental results indicate that setting Br_{th} to three achieves an optimal balance for performance.

4) CFG Traversal and Constraint Generation: InterConFuzz follows Algorithm 2 to simulate $\mathcal{D}$, traverse through `CFG` and use symbolic execution to generate $Constraints$ to achieve higher coverage. Upon completion of the preliminary steps discussed in Section IV-B1, IV-B2, and IV-B3 *InterConFuzz* initiates a loop that continues until all checkpoints are covered (line 1). The initial step within this loop involves simulating until $\mathcal{I}$ is reached and subsequently generating a simulation record file (line 2). This record file is then processed by *InterConFuzz* to measure coverage using control register values (line 3). If the states associated with the current checkpoint are fully covered, *InterConFuzz* designates the checkpoint as "covered" (lines 4-6). Following this, *InterConFuzz* conducts a status check to assess the increase in coverage compared to previous checks.

In the absence of coverage improvement, InterConFuzz records the number of intervals during which coverage remains unchanged. When this count surpasses the designated exit threshold Th defined in Section IV-B3, InterConFuzz concludes that the fuzzer is experiencing stagnation and requires intervention to enhance coverage (line 7). To address this, *InterConFuzz* first identifies the most recent covered state by analyzing the latest simulation dump file and then determines the nearest checkpoint corresponding to this state (lines 8-9). Next, *InterConFuzz* initiates a procedure that navigates *upwards* in the CFG to locate the closest checkpoint leading to an unexplored state (lines 10-11). Here, *upwards* refers to moving toward the root node or initial default state within the CFG hierarchy. Once the appropriate checkpoint is found, *InterConFuzz* resets the simulation to this checkpoint (line 13) and uses the *z3-solver* [51] to establish constraints within the UVM sequencer (lines 14-15). These constraints guide the fuzzer's path, targeting the achievement of the newly identified state. By restricting specific sequences of NoC instructions, these constraints direct the UVM sequencer to produce input sequences that actively explore previously uncovered states. This guided approach allows *InterConFuzz* to generate instruction sequences with an increased probability of reaching unexplored states.

Conversely, if coverage improvement is detected, *InterConFuzz* continues fuzzing with the same setup. If a bug or property violation is identified during this process, *InterConFuzz* documents the specifics, including the clock cycle time and the violated property, and appends this information to the bug report, $\mathcal{R}$.

To execute checkpointing, *InterConFuzz* logs input pattern sequences as the fuzzer explores new states. When a checkpoint is reached, the sequence is recorded as the path to that checkpoint. If the fuzzer reaches a leaf node or a state with no further unexplored branches, *InterConFuzz* resets the design and replays the sequence to return to the checkpoint. Once all nodes from a checkpoint are visited, *InterConFuzz* removes the current checkpoint and promotes the previous one as the new target, repeating the process.

5) Bug Detection: InterConFuzz employs a property-based verification framework to identify security vulnerabilities in designs by leveraging a predefined set of properties that define key security requirements using the UVM monitor. During the fuzzing process, the simulator continuously monitors these properties for any violations. Upon detecting a breach, the UVM monitor generates a comprehensive report, including the name of the violated property, the precise simulation timestamp, the associated waveform, and the sequence of input vectors starting from activating the *reset* signal. This detailed information is then appended to $\mathcal{R}$.

V. Evaluation and Results

A. Evaluation Setup

We evaluated *InterConFuzz* using security vulnerability detection and coverage analysis. For security assessment, we compared *InterConFuzz* with SeVNoC [48] on the OpenTitan TileLink Unprivileged

TABLE I: Details for the Detected Bugs in the Benchmark SoC.

Bug No.	Bug Description	CWE Number	# of input vectors needed to detect the bug
01.	Omitted Wipe Operation After Transaction.	1331	5.85×10^5
02.	Modifying Destination.	668	4.39×10^5
03.	Modifying Source.	668	4.76×10^5
04.	False Transactions Leading to Denial-of-Service (DoS).	1315	7.38×10^4
05.	Corrupting Message Data	1331	6.34×10^5

(TL-UL) network [52]. For coverage evaluation, we benchmarked *InterConFuzz* against NoCFuzzer [49] on OpenPiton's [53] NoC.

1) Evaluation Platform Details: A server with AMD EPYC 7513 2.6GHz, 32C/64T, 128M Cache (200W) Processor, 16 DDR4-3200 64GB RDIMM, 3200MT/s, Dual Rank, 16 GB RAM, and four A100 SXM4 80GB 500W Nvidia GPUs was used to run the simulations. Xilinx Vivado is used as a simulator [13].

2) Parameter Setup: During fuzzing, the system is configured to dump simulation files at every third interval. An interval represents a user-defined number of clock cycles, which is 300 in this case. This setup strikes a balance between performance and resource utilization, ensuring *InterConFuzz* effectively saves the necessary simulation records for coverage analysis.

3) Evaluation Metrics: To assess the effectiveness of *InterConFuzz*, we concentrate on three metrics: (1) its bug detection capability, (2) coverage gain, and (3) computational resource consumption. We first compare the number of bugs *InterConFuzz* identifies against those found by existing fuzzing techniques. We then quantify the coverage achieved in terms of the number of coverage points–defined as the combined variations of control register values and input patterns generated. We also show the effect of symbolic execution and the performance in different parameter configurations, as well as the variance of achieved coverage to illustrate *InterConFuzz*'s efficacy. Moreover, we evaluate the computational resources and time required by *InterConFuzz* to verify the benchmarks in comparison to other methods, highlighting the efficiency of our approach.

B. Vulnerability Description

Table I summarizes the identified bugs, their CWE numbers, and the input vectors needed for detection. The following subsections provide detailed analysis and relevant RTL code snippets.

```
property reset_chk;
(@posedge clk_i) !rst_ni |-> !(|rdata);
endproperty
```

Listing 1: Security Property for clearing data.

a) ***Bug #1****:* This Bug prevents the system from clearing data in the buffers/registers after each transaction. As a result, residual data remains and gets overwritten with every new transaction.

Every new transaction overwrites the previous data, leading to potential data leakage. If the previous data was sensitive or critical, requesting another transaction can lead to storing the data in an unprotected memory address, leading to privacy violations. To detect this Bug, the property in Listing 1 is used, which dictates that upon a reset, `rdata` should be cleared.

b) ***Bug #2****:* This vulnerability allows alteration of a transaction's destination post-initiation. While the original transaction completes successfully, a copy of the data is covertly sent to an unauthorized location, like a UART port or another attacker-controlled channel. As shown in Line 2 and 3 in Listing 2, if the module's write data flag is raised, it sets the write request and data as adversary-defined.

```
1 if (!rst_ni) begin
2   wr_req_q <= (data_wr?) adv_req : 0;
3   wr_data  <= (data_wr?) adv_data : 0;
```

Listing 2: Writing Data to Another Destination.

This leads to a data breach, leaking confidential information to unauthorized locations. This major security threat was detected using the property shown in Listing 3, which requires `wr_req_q` and `wr_data` to be cleared upon reset.

```
property reset_chk;
(@posedge clk_i) !rst_ni |-> !(|wr_req_q)&&!(|wr_data);
endproperty
```

Listing 3: Security Property for Writing Data to Another Destination.

c) ***Bug #3****:* This security breach allows data injection into the system after a transaction starts. Initially, the system retrieves data from a legitimate source, but additional data is then injected from an attacker-controlled memory address. As shown in Line 3 and 4 in Listing 4, if the module's received flag is raised, it sets the receive source ID and source as the adversary's module ID and data size.

```
1 ...
2   reqid_q <= (data_r?) adv_id : 0;
3   reqsz_q <= (data_r?) adv_sz : 0;;
4   rspop_q <= AccessAck;
5   data_r  <= (@posedge clk_i) 0;
6 ...
```

Listing 4: Receiving Data from Another Source.

This can lead to data integrity issues, where unauthorized data is included in transactions. Such manipulation can result in incorrect or compromised data being sent, potentially disrupting operations or leaking sensitive information. This Bug was detected using the property in Listing 5, which asserts that `reqid_q` and `reqsz_q` should be cleared upon reset.

```
property reset_chk;
(@posedge clk_i) !rst_ni |-> !(|reqid_q)&&!(|reqsz_q);
endproperty
```

Listing 5: Security Property for Receiving Data from Another Source.

d) ***Bug #4****:* The bus initiates fake transactions, sending random values across the communication channel, causing congestion and delays for legitimate transactions, as shown in Listing 6. It initiates random write requests and data values if the adversary flag is raised.

```
1 wr_req_q <= (adv?) $RANDOM : 0;
2 wr_data  <= (adv?) $RANDOM : 0;
```

Listing 6: Initiating random write request to perform DoS attack.

This Bug leads to a Denial-of-Service (DoS) attack, in which the system's performance degrades significantly due to the time wasted processing these fake transactions. Legitimate tasks may experience delays or timeouts, reducing the overall system efficiency.

```
property reset_chk;
(@posedge clk_i) !rst_ni |-> !(|wr_req_q)&&!(|wr_data);
endproperty
```

Listing 7: Security Property for detecting DoS attack.

Here, the property in Listing 7 mandates that upon a reset signal, the `we_req_q` and `wr_data` should be set to Zero.

*e) **Bug #5**:* A rogue IP accesses the network and alters both inbound and outbound traffic, modifying data packets in transit and tampering with communication between components. As illustrated in Listing 8, the output write data register receives corrupted data before writing to the destination.

```
assign wdata_o=adv?`Mut_val^tl_i.a_data:tl_i.a_data;
```

Listing 8: Security property for error flag checking.

A rogue IP can manipulate exchanged data, compromising its integrity and introducing inaccuracies and security risks. This manipulation allows attackers to inject misleading data or responses, causing unintended operations. This property checks if `wdata_o` and `tl_i.a_data` are equal.

```
property reset_chk;
(@posedge clk_i) wdata_o == tl_i.a_data;
endproperty
```

Listing 9: Writing Data to Another Destination.

C. Comparative Analysis of InterConFuzz and SEVNoC

In this section, we present a comparative analysis of *InterConFuzz*'s detection capabilities relative to existing NoC security verification approaches, particularly SEVNoC [48]. Our findings reveal that while SEVNoC successfully detected certain bugs, it was unable to identify three significant issues: Bug #1, Bug #2, and Bug #3. *In contrast, InterConFuzz demonstrated a higher degree of reliability, detecting all five bugs within the NoC, indicating InterConFuzz's potential for comprehensive NoC verification.*

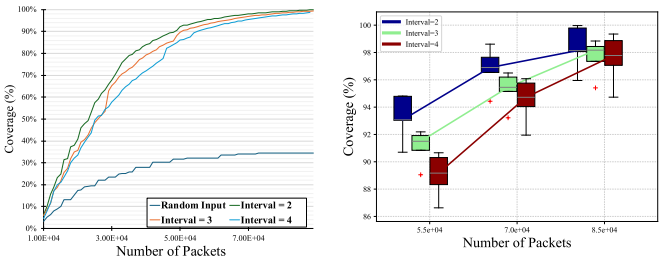
Our investigation found that SeVNoC overlooked Bug #1 because the buffers were often overwritten while evaluating the CFG of the SoC, resulting in an inability to detect mismatches due to not covering all corner cases. Furthermore, SeVNoC failed to identify Bug #2 and Bug #3 since SeVNoC considers a process safe if it completes functions without initially violating security properties. Consequently, these two bugs were not detected by SeVNoC. On the other hand, *InterConFuzz* detected all bugs by reaching the corner cases where the device was reset and initiated the transaction, causing previously written data in the buffer to be sent. Furthermore, *InterConFuzz* does not use a flagging technique to mark paths of the CFGs as secure. As a result, when the corner cases triggered Bug #2 and Bug #3, *InterConFuzz* flagged them as breaching security.

Finally, *our study found that InterConFuzz is more efficient, using 24.4% less memory and 29.5% less computational power compared to SeVNoC.* This demonstrates *InterConFuzz*'s enhanced effectiveness and efficiency in bug detection tasks.

D. Coverage Assessment

In this section, we provide a comparative study of the coverage furnished by *InterConFuzz* against existing approaches. To this end, we used control registers to assess coverage. Each branch of the control path is considered a coverage point for *InterConFuzz*. When multiple control path combinations are possible simultaneously, each combination of taken paths is considered a coverage point, as elaborated in Section IV-B2.

Randomness and Impact of Interval Parameter in Fuzzing: Figure 2 illustrates the coverage improvements of *InterConFuzz* and the variance in coverage gain. Figure 2a and Figure 2b show the coverage gain under different parameter configurations and the variance during three distinct fuzzing stages, respectively.



(a) Coverage comparison with existing fuzzing techniques.

(b) Variance of Coverage for Different Interval Configuration.

Fig. 2: Coverage comparison and variance profile.

Figure 2a underscores the impact of symbolic execution with different parameter configurations in guiding the fuzzing process. Random input generation without symbolic execution achieves only about 30% of the coverage attained with symbolic execution. *InterConFuzz* was tested with three configurations, altering the '$Interval$' parameter. Setting the '$Interval$' to 2, as shown in Figure 2a, results in better coverage but requires approximately 42.1% more processing power due to frequent SMT solver invocations. On the other hand, while setting the '$Interval$' value to 4 reduces processing power by 13.6%, the coverage gain is much slower. Therefore, realizing an '$Interval$' value of 3 provides the best outcome.

Figure 2b shows the variance in *InterConFuzz*'s coverage gain across different parameter setups. Within the selected data range, the impact of the initial randomness is settled down, and the fuzzing algorithm takes the primary effect. *InterConFuzz* displays a maximum coverage gain variance of 2.64%, indicating consistent performance. Notably, with an '$Interval$' value of 2, variance changes more frequently due to increased SMT solver invocations, leading to frequent coverage changes. Conversely, with an '$Interval$' value of 4, the coverage variance remains largely stable.

Comparison with NoCFuzzer: As shown in Figure 2a, the coverage improvement achieved by *InterConFuzz* aligns with the trends observed in the existing NoC fuzzing approach (NoCFuzzer) [49]. Moreover, *Figure 2b demonstrates the consistency of InterConFuzz's coverage, indicated by its low variance, confirming the reliability of InterConFuzz in delivering thorough and deterministic coverage.*

VI. Conclusion

This paper introduces *InterConFuzz*, a novel, coverage-directed hybrid hardware fuzzing framework specifically designed to enhance security verification for NoC architectures. By effectively combining coverage-directed fuzzing with symbolic execution, *InterConFuzz* strategically navigates the fuzzer into previously unexplored states of the NoC, optimizing both the efficiency and efficacy of the verification process. Integrated with UVM, *InterConFuzz* ensures seamless incorporation into existing security verification lifecycles, facilitating its adoption within industry practices. In evaluations conducted on a TL-UL bus network, *InterConFuzz exhibited superior bug detection capabilities, identifying three more bugs than current methodologies while reducing memory consumption by 24.4% and computational demands by approximately 29.5%.* Moreover, *InterConFuzz* achieved similar coverage when evaluated on OpenPiton's NoC architecture, underscoring its efficiency. These compelling results highlight *InterConFuzz*'s potential as a robust and efficient security verification framework for NoC architectures, positioning it as an exceptional option for integration into the security development lifecycle (SDL) of commercial SoC design flows.

References

[1] P. Patra, "On the cusp of a validation wall," *IEEE Design & Test of Computers*, vol. 24, no. 2, pp. 193–196, 2007.

[2] F. Farahmandi *et al.*, *System-on-Chip Security*. Springer International Publishing, Nov. 2019. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-30596-3_1

[3] S. V. Academy, "Universal verification methodology," Online, 2024. [Online]. Available: https://verificationacademy.com/topics/uvm-universal-verification-methodology/

[4] K. Salah, "A uvm-based smart functional verification platform: Concepts, pros, cons, and opportunities," in *2014 9th International Design and Test symposium (IDT)*. IEEE, 2014, pp. 94–99.

[5] W. Ni *et al.*, "Research of reusability based on uvm verification," in *2015 IEEE 11th International Conference on ASIC (ASICON)*. IEEE, 2015, pp. 1–4.

[6] "Libfuzzer – a library for coverage-guided fuzz testing. — llvm 12 documentation," accessed: 08/28/2024. [Online]. Available: https://llvm.org/docs/LibFuzzer.html

[7] "American fuzzy lop (afl) fuzzer." 2014. [Online]. Available: https://lcamtuf.coredump.cx/afl/technical_details.txt

[8] K. Laeufer *et al.*, "Rfuzz: Coverage-directed fuzz testing of rtl on fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[9] J. Hur *et al.*, "Difuzzrtl: Differential fuzz testing to find cpu bugs," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1286–1303.

[10] T. Trippel *et al.*, "Fuzzing hardware like software," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3237–3254.

[11] R. Kande *et al.*, "{TheHuzz}: Instruction fuzzing of processors using {Golden-Reference} models for finding {Software-Exploitable} vulnerabilities," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3219–3236.

[12] W. Snyder, "Verilator and systemperl," in *North American SystemC Users' Group, Design Automation Conference*, 2004.

[13] "Xilinx vivado," accessed: 05/19/2024. [Online]. Available: https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug904-vivado-implementation.pdf

[14] "Opentitan | documentation," accessed: 05/19/2024. [Online]. Available: https://opentitan.org/book/doc/introduction.html

[15] R. Craig *et al.*, *Systematic Software Testing*, ser. Artech House ITS library. Artech House, 2002. [Online]. Available: https://books.google.com/books?id=2_gbZYZcZXgC

[16] M. Sutton *et al.*, *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[17] X. Zhu *et al.*, "Fuzzing: a survey for roadmap," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–36, 2022.

[18] A. Takanen *et al.*, *Fuzzing for software security testing and quality assurance*. Artech House, 2018.

[19] K. Serebryany, "OSS-Fuzz - google's continuous fuzzing service for open source software," in *Proceedings of the 26th USENIX Security Symposium*. Vancouver, BC: USENIX Association, Aug. 2017.

[20] L. Dukes *et al.*, "A case study on web application security testing with tools and manual testing," in *2013 Proceedings of IEEE Southeastcon*. IEEE, 2013, pp. 1–6.

[21] W. Xu *et al.*, "Freedom: Engineering a state-of-the-art dom fuzzer," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 971–986.

[22] https://google.github.io/clusterfuzz/, accessed: 04/03/2024.

[23] Z. Michal, https://lcamtuf.coredump.cx/afl/, accessed: 04/03/2024.

[24] M. Böhme *et al.*, "Coverage-based greybox fuzzing as markov chain," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1032–1043.

[25] M. Beckmann *et al.*, "Coverage-guided fuzzing of embedded systems leveraging hardware tracing," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 362–378.

[26] A. Fioraldi *et al.*, "AFL++ : Combining incremental steps of fuzzing research," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/fioraldi

[27] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[28] R. Baldoni *et al.*, "A survey of symbolic execution techniques," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.

[29] C. Cadar *et al.*, "Klee: unassisted and automatic generation of high-coverage tests for complex systems programs." in *OSDI*, vol. 8, 2008, pp. 209–224.

[30] H. Park *et al.*, "Strategies to maximize reusability of uvm test scenarios in soc verification," *DV-Con*.

[31] S. Vasudevan, *Practical UVM: Step by Step with IEEE 1800.2*. R R BOWKER LLC, 2020. [Online]. Available: https://books.google.com/books?id=LY9qzQEACAAJ

[32] D. M. Ancajas *et al.*, "Fort-nocs: Mitigating the threat of a compromised noc," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[33] T. Boraten *et al.*, "Mitigation of denial of service attack with hardware trojans in noc architectures," in *2016 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2016, pp. 1091–1100.

[34] A. P. D. Nath *et al.*, "Resilient system-on-chip designs with noc fabrics," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2808–2823, 2020.

[35] A. Basak *et al.*, "Exploiting design-for-debug for flexible soc security architecture," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[36] P. K. Mishra *et al.*, "Real-time detection and localization of dos attacks in noc based soc architectures," Oct. 24 2023, uS Patent 11,797,667.

[37] C. Ciordas *et al.*, "An event-based monitoring service for networks on chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 10, no. 4, pp. 702–723, 2005.

[38] V. Y. Raparti *et al.*, "Lightweight mitigation of hardware trojan attacks in noc-based manycore computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[39] L. Zhao *et al.*, "Send hardest problems my way: Probabilistic path prioritization for hybrid fuzzing." Network and Distributed System Security Symposium (NDSS), 2019.

[40] X. Meng *et al.*, "Rtl-contest: Concolic testing on rtl for detecting security vulnerabilities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 466–477, 2021.

[41] M. Tiwari *et al.*, "Crafting a usable microkernel, processor, and i/o system with strict and provable information flow security," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3, pp. 189–200, 2011.

[42] X. Li *et al.*, "Caisson: a hardware description language for secure information flow," *ACM Sigplan Notices*, vol. 46, no. 6, pp. 109–120, 2011.

[43] D. Zhang *et al.*, "A hardware design language for timing-sensitive information-flow security," *Acm Sigplan Notices*, vol. 50, no. 4, pp. 503–516, 2015.

[44] K. Ryan *et al.*, "Augmented symbolic execution for information flow in hardware designs," *arXiv preprint arXiv:2307.11884*, 2023.

[45] M. Hicks *et al.*, "Specs: A lightweight runtime mechanism for protecting software from security-critical processor bugs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 517–529.

[46] S. R. Sarangi *et al.*, "Phoenix: Detecting and recovering from permanent processor design bugs with programmable hardware," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 2006, pp. 26–37.

[47] I. Wagner *et al.*, "Engineering trust with semantic guardians," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[48] X. Meng *et al.*, "Sevnoc: Security validation of system-on-chip designs with noc fabrics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 672–682, 2022.

[49] R. Ma *et al.*, "Nocfuzzer: Automating noc verification in uvm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[50] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *Applied Reconfigurable Computing*, ser. Lecture Notes in Computer Science, vol. 9040. Springer International Publishing, Apr 2015, pp. 451–460. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16214-0_42

[51] L. De Moura *et al.*, "Z3: An efficient smt solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[52] "Sifive tilelink specification," accessed: 08/28/2024. [Online]. Available: https://static.dev.sifive.com/docs/tilelink/tilelink-spec-1.7-draft.pdf

[53] J. Balkind *et al.*, "Openpiton: An open source manycore research framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

ZXNet: ZX Calculus-Driven Graph Neural Network Framework for Quantum Circuit Equivalence Checking

Navnil Choudhury
Electrical and Computer Engineering
The University of Texas at Dallas
Richardson, USA
navnil.choudhury@utdallas.edu

Ameya S Bhave
Electrical and Computer Engineering
The University of Texas at Dallas
Richardson, USA
ameya.bhave@utdallas.edu

Kanad Basu
Electrical and Computer Engineering
The University of Texas at Dallas
Richardson, USA
kanad.basu@utdallas.edu

***Abstract*—Quantum circuit execution often requires transpilation into hardware-compatible instructions, which can significantly alter the original design, making equivalence checking essential. However, existing approaches struggle with scalability and computational overhead. In this paper, we present ZXNet, a transformative framework for quantum circuit equivalence checking using ZX calculus-based graph abstractions. Leveraging graph neural networks, ZXNet captures complex equivalence patterns by integrating critical local and global circuit features. ZXNet achieves 99.4% validation accuracy, and up to 62× speedup over state-of-the-art methods, furnishing improvements of 45.83% in scalability, 42.22% in per-qubit verification time, and 5.94% in accuracy, outperforming state-of-the-art approaches.**

***Index Terms*—Quantum computing, equivalence checking, ZX calculus, graph neural networks.**

I. Introduction

Quantum computing represents a paradigm shift in computational science, offering significant speedups over classical computers by leveraging quantum mechanical principles such as superposition and entanglement, as demonstrated in Shor's and Grover's algorithms [1], [2]. Quantum computers operate on quantum bits (*qubits*), which are manipulated by quantum gates that apply unitary transformations to steer qubits toward desired states, with results determined probabilistically through measurement. The increasing demand for quantum computing, driven by applications in fields like cryptography, biochemistry, and machine learning [3]–[5], has led to the emergence of Quantum-as-a-Service (QaaS), allowing users to execute algorithms on cloud-based quantum hardware. However, given the constraints of current quantum hardware, such as limited connectivity and gate fidelity, the submitted algorithms undergo *quantum circuit transpilation* [6]. This process optimizes gate sequences, maps logical qubits to physical hardware, and decomposes complex gates into supported basis gates. While it ensures hardware compatibility, transpilation can significantly alter the original circuit's structure.

The modifications introduced to a quantum circuit during transpilation renders quantum circuit equivalence checking essential [7]. This process verifies that the transpiled circuit preserves the original quantum functionality by ensuring consistent output states and measurement probabilities across all possible inputs. Equivalence checking is particularly crucial for complex circuits, where even minor discrepancies can lead to significant computational errors [8]. As such, it acts as a safeguard to ensure that the executed quantum algorithm faithfully implements the user's intended operations.

To meet the growing demand for equivalence checking in increasingly complex quantum circuits, various frameworks have been developed, including Decision Diagrams, Matrix Product Operators, and simulation-based methods [9]–[11]. However, the existing approaches suffer from significant limitations in terms of time and memory overhead, which hinders their scalability and limits their applicability to complex quantum circuits, as explained in Section III. In addition to these methods, ZX calculus has emerged as a powerful framework for equivalence checking, utilizing its capability to represent quantum circuits as ZX graphs and systematically apply rewrite rules to transform these graphs into canonical forms that preserve circuit semantics [12].

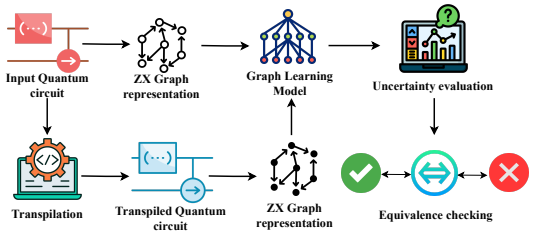


Fig. 1: Overview of our proposed equivalence checking framework using a novel graph learning method, ZXNet.

The graphical representation utilized by ZX calculus facilitates algebraic transformations and rule-based simplifications, enabling the identification of equivalence between circuits. Systematic transformations, such as full reduction, standardize equivalent circuits into near-canonical forms, enhancing the accuracy of equivalence detection. Additionally, ZX calculus effectively encodes both Clifford and non-Clifford operations, ensuring a comprehensive representation of diverse quantum transformations. While the standalone application of ZX calculus for equivalence checking poses computational challenges for large circuits due to extensive rule applications and complex pattern-matching processes, we observe its representation of quantum circuits is highly suitable for graph learning tasks. By encapsulating the circuit's structure and operations in a graphical format, ZX graphs provide a natural input for Graph Convolution Networks (GCNs), which excel at learning relational patterns and structural equivalence [13] [14] [15]. The ability of ZX graphs to represent both local gate operations and global entanglement patterns aligns well with the message-passing mechanisms of GCNs. This compatibility allows ZX calculus to act as a bridge between quantum circuit structures and machine learning, enabling scalable, efficient, and accurate equivalence checking for circuits with varying complexities.

In this paper, we present ZXNet, a transformative equivalence checking framework that, for the first time, leverages graph abstractions derived from ZX calculus, followed by the usage of a novel Graph Convolution Network (GCN) model designed for scalable and efficient quantum circuit equivalence checking. An overview

This work is supported by the National Science Foundation (NSF award #2343652).

of the proposed framework is depicted in Figure 1. ZXNet combines a unique encoding of features extracted from simplified ZX graphs, with a novel GCN architecture designed to learn intricate equivalence patterns efficiently. Furthermore, ZXNet incorporates a novel uncertainty-driven output validation mechanism and a Jaccard-adjusted loss function, which together enhance the model's robustness and accuracy, especially for hard-to-detect errors. *By leveraging ZX graphs to capture essential quantum transformations, ZXNet significantly improves scalability, reduces computational overhead, and establishes a reliable, adaptable solution for equivalence checking across quantum circuits of varying sizes and complexities.* Our main contributions are summarized as follows:

- We present ZXNet, a novel framework for equivalence checking that leverages simplified ZX graph representations of quantum circuits. ZXNet employs an innovative GCN architecture designed to effectively utilize extracted features, enabling efficient and accurate equivalence checking.
- To enhance the robustness of ZXNet, we propose a novel uncertainty-driven validation mechanism utilizing an edge-based Jaccard index value that enables the detection of minor infringements during the transpilation process and significantly improves error detection capabilities.
- ZXNet addresses the challenges of scalability and complexity by being able to learn and subsequently generalize equivalence relations, thereby being able to handle large and highly entangled quantum circuits with up to 120 qubits.
- Our evaluations on extensive benchmarks demonstrate that ZXNet significantly outperforms state-of-the-art equivalence-checking methods, achieving speedups of up to $801\times$ and an accuracy of up to 99.4%, establishing its reliability and effectiveness. ZXNet also surpasses existing techniques, achieving a 45.83% improvement in scalability, a 42.22% reduction in per-qubit verification time, and a 5.94% increase in accuracy.

II. Background and Motivation

A. Quantum Computing Basics

1) Qubits: A qubit, or quantum bit, is the fundamental unit of quantum information, analogous to a classical bit [16]. Unlike classical bits, which exist in either 0 or 1, a qubit can exist in a superposition of both states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

where $|0\rangle$ and $|1\rangle$ denote the basis states, and α and β are complex probability amplitudes such that $|\alpha|^2 + |\beta|^2 = 1$. This superposition property allows quantum computers to process a much larger state space than classical systems [17]. Qubits can also be entangled, enabling quantum circuits to perform powerful computations through correlated states. Measurement of a qubit collapses its state to either $|0\rangle$ or $|1\rangle$, with corresponding probabilities $|\alpha|^2$ and $|\beta|^2$.

2) Quantum Gates: Quantum gates are unitary operations that change the state of qubits. Mathematically, these gates can be expressed as matrices with dimensions of $2^n \times 2^n$, where n is the number of qubits. Depending on the number of qubits they are acting on, there are single-qubit gates (such as `X-gate`, `S-gate`, `H-gate`, *etc.*) and multi-qubit quantum gates (such as `CX-gate` and `CZ-gate`) [18].

B. ZX calculus

ZX calculus has emerged as a graphical language used to represent and reason about quantum circuits through structured diagrams

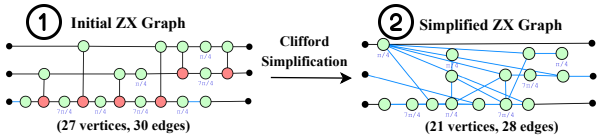


Fig. 2: Example of ZX graph representation and simplification.

instead of traditional gate sequences. It encodes quantum operations as graphs, where Z-spiders (green nodes) and X-spiders (red nodes) represent operations in different bases, and edges represent quantum entanglement or information flow [19] [20]. ZX calculus includes transformation rules—such as spider fusion, Hadamard edge transformation, and Clifford simplification—which simplify circuits while preserving equivalence. ZX calculus has been employed in various quantum computing applications. For instance, mixed quantum-classical optimization methods have utilized it to streamline computations, effectively bridging classical and quantum optimization techniques [21]. It has also been applied to optimize braided circuits on surface codes, demonstrating improvements in circuit efficiency [22]. In comparative studies, ZX calculus-based tools like PyZX have been benchmarked against Qiskit transpilation, where they have shown the potential to reduce gate counts significantly in certain scenarios [23]. An example of ZX graphs and their subsequent simplification is shown in Figure 2, where (1) is the initial ZX graph representation of a quantum circuit, which is subsequently simplified into (2) using Clifford simplification. These rules rooted in formalism render ZX calculus particularly valuable for tasks like circuit equivalence checking and quantum circuit simplification.

C. Graph Neural Networks for Quantum Circuit Equivalence

Our selection of Graph Neural Networks (GNN)s for quantum circuit equivalence checking with ZX graphs is motivated by their capability to directly and effectively process graph-structured data. [24]. Furthermore, unlike other ML models, GNNs natively handle variable-sized graph structures, preserving ZX graph topology and connectivity. Graph Convolution Networks are a type of GNNs, and they aggregate information from a node's neighbors, capturing local interactions (*e.g.*, entanglements) and propagating this information to model global structures [25] [26]. This propagation makes GCNs particularly effective for tasks like ZX graph equivalence checking, where small structural changes or variations in graph layouts (isomorphic graphs) must be handled robustly without being sensitive to the specific graph representation [27].

III. Related Work

Various algorithms have been developed to tackle the unique challenges of quantum circuit equivalence checking. However, they face several limitations, as discussed here. Decision Diagrams (DDs) are widely used for quantum circuit equivalence but struggle with exponential node growth and high memory usage for circuits exceeding 20 qubits [10]. They are also sensitive to variable ordering, which inflates diagram size and complicates verification. In contrast, the proposed ZXNet approach uses scalable, memory-efficient graph representations, effectively handling larger, highly entangled circuits without significant overhead.

Matrix Product Operators (MPO)-based equivalence checking approaches handle low-to-moderate entanglement efficiently using tensor networks, but face exponential scaling with high entanglement or depth, requiring costly tensor operations [9], [28], [29]. Furthermore, they struggle with non-local entanglement and error handling. ZXNet overcomes these challenges using efficient graph representations without intensive computations.

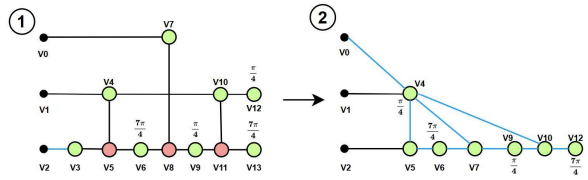


Fig. 3: ZX simplification reducing circuit complexity to enable critical feature extraction.

The existing ZX calculus-based equivalence checking approach struggles with deeply entangled or large circuits due to extensive transformations and inefficiencies [12]. They also lack flexibility for structural variations, limiting their applicability in complex circuits with high entanglement. These limitations primarily arise since this approach does not explicitly utilize any learning model to capture equivalence relations, and primarily relies on algebraic rule-based transformations [19]. In contrast to the existing approach, ZXNet embeds ZX graphs into a learnable framework using GCNs, eliminating exhaustive rule applications and enabling efficient, flexible verification for complex quantum circuits [30].

IV. Proposed ZXNet Framework

A. Feature extraction from ZX graph abstractions

Our approach begins by constructing ZX graph representations from quantum circuits, encoding the circuit's structure into a graphical form suitable for graph learning networks. We apply **full reduction on ZX graphs** to achieve a near-canonical representation, standardizing equivalent circuits and enhancing the model's learning capabilities. This approach preserves circuit equivalence while removing redundancies, thereby minimizing complexity. We have demonstrated an example of this simplification in Figure 3, where the initial ZX graph representation of a quantum circuit G, shown in (1) has 14 nodes, and is simplified to ZX graph (2) G' with 10 nodes, utilizing full reduction. Additionally, since full reduce supports both Clifford and non-Clifford gates, we ensure that the dataset effectively captures essential structural features for equivalence checking.

To design a graph-learning model capable of quantum circuit equivalence checking, we extract representative features from simplified ZX graphs G' that encode critical structural and functional information. This information is categorized into two types: *node features*, capturing critical properties of the vertices, and *edge features*, representing key relationships within the edges of the ZX graph. The extracted features from the simplified ZX graph (2) from Figure 3 are presented in Table I, with the corresponding features detailed subsequently.

a) Node features: These are denoted in columns one to six in Table I and their importance is detailed subsequently.

- **Vertex number**(v_i): The vertex number encodes positional and structural context within the ZX graph, enabling accurate node alignment and equivalence detection between quantum circuits.
- **Node type**(n_i): Node type in ZX graphs distinguishes Z and X-basis operations, ensuring consistent qubit mapping for equivalence checking and identifying mismatches in operation targets.
- **Row Index**(r_i): The row index captures operation sequence, ensuring temporal consistency critical for equivalence, as misalignment alters circuit behavior.
- **Degree**(d_i): The degree captures a node's connectivity and entanglement, aiding equivalence checking by identifying nodes with similar interaction patterns across circuits.

TABLE I: Feature Encoding of an Example Circuit.

Node Features						Edge Features	
Vertex Number (v_i)	Node Type (n_i)	Row Index (r_i)	Degree (d_i)	Phase (p_i)	Qubit Index (q_i)	Associated Edges (v_i, v_j)	Edge Type (e_i)
0	Boundary	0	1	0	0	(0, 7)	H
1	Boundary	0	1	0	1	(1, 4)	S
2	Boundary	0	1	0	2	(2, 5)	S
4	Z	2	3	0	2	(4, 5), (4, 7), (4, 10)	H, H, H
5	Z	2	3	1/4	1	(5, 2), (5, 4), (5, 6)	S, H, H
6	Z	3	2	7/4	2	(6, 5), (6, 7)	H, H
7	Z	4	3	0	2	(7, 6), (7, 9), (7, 4)	H, H, H
9	Z	5	2	1/4	2	(9, 7), (9, 10)	H, H
10	Z	6	3	0	2	(10, 9), (10, 12), (10, 14)	H, H, H
12	Z	7	1	7/4	2	(12, 10)	H

- **Phase**(p_i): The phase parameter encodes rotations or shifts, where differences signal distinct transformations and a lack of equivalence.
- **Qubit Index**(q_i): The qubit index specifies the target qubit of an operation, ensuring consistent qubit mapping and detecting mismatches during equivalence checking.

b) Edge features: These are detailed in columns seven and eight in Table I, with their importance mentioned as follows.

- **Associated Edges** (v_i, v_j): This feature defines connectivity and information flow, requiring identical source-target relationships for equivalence to preserve circuit functionality.
- **Edge type** (e_i): The edge type distinguishes regular connections from Hadamard transformations, and requires consistency across ZX graphs to avoid basis mismatches in equivalence checking.

B. GCN Model Design for Equivalence Checking

1) Model Construction: The ZX graph representation of quantum circuits, encoded into the node and edge features we extracted (Section IV-A), serves as the input for training our GCN model. The model processes pairs of ZX graphs $(\mathbf{G_1}, \mathbf{G_2})$ to produce a joint representation via graph convolutions and outputs a logit vector that indicates the likelihood of equivalence.

2) Model Training: The training process of our GCN model is outlined in Algorithm 1. The model parameters and the optimizer are initialized with a learning rate η to begin the training process (*line 1*). Following this, for each epoch, batches of graph pairs (G_1, G_2) and their labels (y) are extracted from the dataset. For each pair of graphs, the adjacency matrix $\mathbf{A_{N\times N}}$ and node feature matrix $\mathbf{X_{N\times F}}$ (where N represents the number of nodes in the graph and F refers to the number of features per node) are extracted, representing the structural and feature properties of the graphs (*lines 2–3*).

Graph Embedding: Each node $v \in V(G_1) \cup V(G_2)$ (where $V(G)$ denotes the set of all vertices in graph G) is initialized with an embedding $\mathbf{e_v}$, which is a learnable vector representation which encodes the node attributes and relationships (*lines 4–7*).

Graph Convolutional Layers: The embeddings are iteratively refined across L number of graph convolutional layers. At each layer, the embedding of each node v is updated using the operation shown in Equation 2 (*line 8*).

$$e_v^{(l)} = \texttt{AggregateNeighbors}(e_v^{(l-1)}, A, W_l) \tag{2}$$

In Equation 2, the function, `AggregateNeighbors`, combines the embeddings of the node v's neighbors, weighted by the adjacency matrix A and a layer-specific weight matrix W_l [31]. Following this, the updated embeddings are passed through an activation function (*line 9*). For our GCN model, we utilize the `ReLU` (Rectified Linear Unit) activation [32], as shown in Equation 3.

$$e_v^{(l)} = \texttt{ReLU}\left(W_l \cdot e_v^{(l-1)} + b_l\right) \tag{3}$$

In Equation 3, W_l is the learnable weight matrix for layer l, $e_v^{(l-1)}$ is the embedding of node v from the previous layer and b_l

Algorithm 1: GraphMatching Network Training with Jaccard-Adjusted Loss.

Input: Dataset $\mathcal{D}$ with ZX graph pairs (G_1, G_2) and labels y; learning rate η; uncertainty threshold τ; Jaccard index threshold J_{thresh}; scaling factor β
Output: Trained GraphMatching Network model

1 Initialize model parameters and optimizer with learning rate η;
2 **foreach** epoch **do**
3 **foreach** batch $(G_1, G_2, y) \subset \mathcal{D}$ **do**
4 Extract adjacency matrix $A_{N\times N}$ and node feature matrix $X_{N\times F}$ from each graph pair (G_1, G_2);
5 **foreach** node $v \in G_1 \cup G_2$ **do**
6 Initialize embedding e_v for node v;
7 **end**
8 **for** layer $l \in L$ **do**
9 Update $e_v \leftarrow$ `AggregateNeighbours` $(e_v, A_{N\times N}, W_l)$;
10 Apply activation $e_v \leftarrow$ `ReLU` (e_v);
11 **end**
12 Compute logits $f(G_1, G_2)$ via forward pass;
13 Compute softmax probabilities: $P(y|G_1, G_2)$;
14 $\mathcal{L}_{\text{base}} \leftarrow$ `CrossEntropyLoss` $(P(y|G_1, G_2), y)$;
15 $(\hat{y}, \delta, \mathcal{L}_{\text{Jaccard}}) \leftarrow$ `JaccardCheck` $(f(G_1, G_2), E_1, E_2, \tau, J_{\text{thresh}})$;
16 **if** $\delta > \tau$ **then**
17 $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{base}} + \beta \cdot \mathcal{L}_{\text{Jaccard}}$;
18 **end**
19 **else**
20 $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{base}}$;
21 **end**
22 Backpropagate and update model parameters;
23 **end**
24 **end**
25 **return** Trained GraphMatching Network model;

Algorithm 2: Uncertainty Calculation with Jaccard Index Check.

Input: Model logits $f(G_1, G_2)$; edge sets E_1, E_2; uncertainty threshold τ; Jaccard index threshold J_{thresh}; scaling factor α
Output: Adjusted prediction $\hat{y}$, uncertainty δ, Jaccard penalty term $\mathcal{L}_{\text{Jaccard}}$

1 Compute: $P(y|G_1, G_2) = \text{softmax}(f(G_1, G_2))$;
2 Set confidence $\gamma \leftarrow \max(P(y|G_1, G_2))$;
3 Calculate sum of exponentials: $f_{\text{sum}} \leftarrow \sum_c e^{f_c(G_1,G_2)}$;
4 Compute uncertainty δ: $\delta \leftarrow -\log(f_{\text{sum}})$;
5 Initialize Jaccard penalty term: $\mathcal{L}_{\text{Jaccard}} \leftarrow 0$;
6 **if** $\delta > \tau$ **then**
7 Calculate edge intersection $E_\cap \leftarrow$ `Intersect` (E_1, E_2);
8 Calculate edge union $E_\cup \leftarrow$ `Union` (E_1, E_2);
9 Compute Jaccard index $J \leftarrow \frac{|E_\cap|}{|E_\cup|}$;
10 **if** $J > J_{\text{thresh}}$ **then**
11 Adjust prediction $\hat{y} \leftarrow$ equivalence class;
12 **end**
13 Update Jaccard penalty term: $\mathcal{L}_{\text{Jaccard}} \leftarrow \alpha \cdot |y - J|$;
14 **end**
15 **return** $\hat{y}$, δ, $\mathcal{L}_{\text{Jaccard}}$;

is the bias term that shifts the output of the linear transformation, allowing the model to represent features that are not centered at zero. The `ReLU` activation ensures that only positive values from the transformed embedding $W_l \cdot e_v^{(l-1)} + b_l$ contribute to the next layer's embedding $e_v^{(l)}$ (*line 10*). Taking only positive values through `ReLU` is important because it induces sparsity in the embeddings, reducing noise from less significant features (negative values) and focusing on meaningful contributions, which enhances model efficiency and robustness. After passing through all layers, logits $f(G_1, G_2)$ are computed via a forward pass, representing the model's predictions for equivalence classification (*line 11*). These logits are then converted into softmax probabilities for classification (*line 12*). The base cross-entropy loss, $\mathcal{L}_{\text{base}}$, is computed using the predicted probabilities and the true labels, quantifying the model's prediction error (*line 13*). However, there may be cases where the model's predictions are uncertain, indicating the model has low confidence associated with its prediction. In such cases, the **`JaccardCheck`** function is invoked (described in Section IV-C). This function computes the uncertainty metric δ and adjusts the loss term, if required, to include the Jaccard-adjusted loss $\mathcal{L}_{\text{Jaccard}}$. The adjustment is contingent on the Jaccard index threshold J_{thresh} and the uncertainty threshold τ (*line 14*).

When $\delta > \tau$, the total loss is updated, as shown in Equation 4 (*lines 15–17*):

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{base}} + \beta \cdot \mathcal{L}_{\text{Jaccard}} \tag{4}$$

In Equation 4, β is a scaling factor controlling the contribution of the adjusted loss, and can be treated as a hyperparameter of the GCN model we develop.

If the uncertainty δ is below τ, the total loss is set to the base cross-entropy loss $\mathcal{L}_{\text{base}}$ (*lines 18–20*). The Jaccard-based adjustment ensures that structural discrepancies in the graphs are penalized, thereby improving the robustness of the training process. Moreover, this adaptive loss adjustment enables the model to dynamically focus on hard-to-classify instances where structural similarity is a factor. The model parameters are updated via backpropagation using the total loss $\mathcal{L}_{\text{total}}$, gradually minimizing it over the course of training (*line 21*). After all the epochs are complete, the final trained model is returned, ready for validation and testing (*lines 22–24*).

C. Uncertainty-Driven Loss Adjustment

A central element of our methodology is a novel *uncertainty-driven Jaccard index validation* with loss adjustment, which enhances both the robustness and accuracy of equivalence checking. When our GCN model's prediction confidence drops below a predefined threshold, we incorporate the Jaccard index to serve as a secondary validation measure. The Jaccard index is utilized for its efficiency and effectiveness in capturing the overlap between the edge sets of graphs [33]. In the context of equivalence checking, it serves to identify discrepancies arising from subtle changes in the ZX graph representations, often caused by erroneous gate insertions in the original quantum circuit. We integrated the Jaccard index into a specially designed loss function during training, ensuring the model prioritizes learning in high-uncertainty scenarios. This process, (summarized as the `JaccardCheck` function in Section IV-B) is detailed in Algorithm 2. This algorithm can be broken down into *three* distinct phases, which are explained subsequently.

1) Uncertainty Threshold and Jaccard Index Validation: The GCN model outputs a probability distribution over the two classes (equivalent and non-equivalent) for each pair of graphs, represented in Equation 5 as follows:

$$P(y|G_1, G_2) = \text{softmax}(f(G_1, G_2)) \tag{5}$$

In Equation 5, $f(G_1, G_2)$ denotes the logits for the classes, and $P(y|G_1, G_2)$ is the probability that graphs G_1 and G_2 are equivalent. The *confidence score* and the *uncertainty* is computed are given by Equations 6 and 7, respectively, as follows:

$$\gamma = \max(P(y|G_1, G_2)) \tag{6}$$

$$\delta = -\log\left(\sum_c e^{f_c(G_1,G_2)}\right) \tag{7}$$

In Equation 6, the softmax probabilities $P(y|G_1, G_2)$ are computed to determine the model's predicted probabilities (*line 1*) in Algorithm 2. The model's confidence score γ is set to the maximum of these probabilities, representing the model's confidence in its prediction (*line 2*). Following this, the uncertainty score δ is calculated, as shown in Equation 7, using the sum of exponentials of the logits $f_c(G_1, G_2)$ for each class c, capturing the model's uncertainty in its prediction (*lines 3-4*) of Algorithm 2.

2) Jaccard Index Calculation and Threshold Adjustment: Let E_1 and E_2 be the edge sets of graphs G_1 and G_2. The Jaccard index $J(E_1, E_2)$ is defined in Equation 8:

$$J(E_1, E_2) = \frac{|E_1 \cap E_2|}{|E_1 \cup E_2|} \tag{8}$$

If the uncertainty score δ is above a set threshold τ, we proceed with calculating the Jaccard index J on the edge sets of the two graphs. The intersection $E_\cap$ and union $E_\cup$ of the edge sets E_1 and E_2 are calculated to compute J, (*lines 6-9*) of Algorithm 2. A threshold, J_{thresh} is selected empirically to maximize the separation between equivalent and non-equivalent pairs. If $J(E_1, E_2) > J_{\text{thresh}}$, we adjust the prediction $\hat{y}$ to indicate equivalence (*lines 10-12*).

3) Loss Function Modification with Jaccard Index: To incorporate the Jaccard index into training, we adjust the loss function based on both the uncertainty score and the Jaccard index value. Let $\mathcal{L}_{\text{base}}$ denote the standard cross-entropy loss for the GCN model, as shown in Equation 9:

$$\mathcal{L}_{\text{base}} = -\sum_{i=1}^{N} y_i \log P(y_i|G_1, G_2) \tag{9}$$

In Equation 9, y_i is the true label for the i-th example in the batch, and N is the batch size. A Jaccard-based penalty term $\mathcal{L}_{\text{Jaccard}}$ is defined to reinforce correct predictions for high-uncertainty cases. This term is based on the absolute difference between the model's prediction and the Jaccard index decision, shown in Equation 10:

$$\mathcal{L}_{\text{Jaccard}} = \alpha \cdot |\hat{y} - J(E_1, E_2)| \tag{10}$$

In Equation 10 α is a scaling factor that controls the impact of the Jaccard-based adjustment, and $\hat{y}$ is the model's predicted equivalence probability. This penalty term is updated (*line 13*). The total loss $\mathcal{L}_{\text{total}}$ is computed as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{base}} + \beta \cdot \mathcal{L}_{\text{Jaccard}} \tag{11}$$

where the uncertainty-based weighting factor β is defined by:

$$\beta = \begin{cases} 1 & \text{if} \quad \delta > \tau, \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Our proposed loss function incorporates a penalty term for uncertain predictions, aligning the model's output with the Jaccard index for structurally similar graphs. This adjustment improves the GCN's accuracy and robustness by reinforcing learning in high-uncertainty cases where structural similarity is high.

V. Evaluation of the Proposed Framework

A. Experimental Setup

Benchmarks for training: For training our proposed GCN model, we utilized 16 distinct benchmarks, with eight different transpilation configurations at different stages of compilation, obtained from MQTBench [34]. Furthermore, each benchmark circuit was considered for a range of four to sixteen qubits, to furnish a total of *1792* quantum circuits to allow our GCN model to adequately capture traits pertaining to equivalence of quantum circuits.

Benchmarks for evaluation: To evaluate our verification framework, we consider a total of 25 distinct benchmark circuits (*all scalable benchmarks*), with 4-120 qubits, available from MQTBench and their transpiled configurations, conducted on IBM Qiskit [35]. The circuits are converted into ZX graph representations using PyZX [36].

Model evaluation metrics: We assess our GCN model's performance using accuracy, precision, and recall: *accuracy* measures overall correctness in classifying circuit equivalence, *precision* evaluates the reliability of positive (equivalent) classifications, and *recall* gauges the model's ability to identify all actual equivalent circuits.

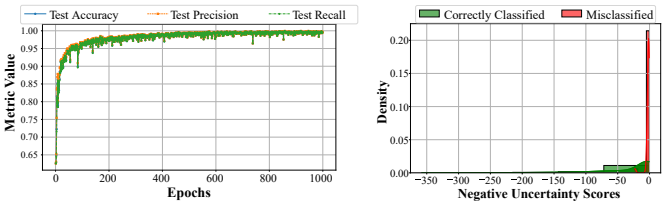
Comparison metrics: To compare the performance of our equivalence checking framework with existing models, we utilize *scalability* of the approach, signifying the number of qubits up to which we can verify quantum circuit equivalence, followed by *time consumption*, signifying the time taken to verify per qubit. Finally, we also compare the *accuracy* for verification of ZXNet with state-of-the-art.

B. Evaluation

For quantum circuit equivalence checking, we utilize our novel GCN model, which consists of *two graph convolutional layers with 64 hidden channels each*, a *single global mean pooling layer*, and *two fully connected layers* with ReLU activations. The architecture is designed to process paired ZX graph representations and output logits for equivalence classification. *It should be noted that our model conducts binary classification, where we feed the model with two quantum circuits to subsequently determine their equivalence.*

Figure 4a depicts the precision, recall, and accuracy of our model across epochs. The y-axis indicates the values each metric achieves across epochs, represented in the x-axis. From this figure, it can be observed that ZXNet furnishes accuracy, precision, and recall values of up to 99.4%, 99.2%, and 99.8%, respectively. Furthermore, the figure depicts a strong overlap among these metrics, indicating balanced performance across classes. This alignment suggests that our model effectively generalizes new data [37]. Additionally, the close values of precision and recall demonstrate that our model neither over-predicts equivalences nor misses true ones. This balance highlights the reliability of our approach in accurately identifying circuit equivalences.

Figure 4b illustrates the distribution of negative uncertainty scores for correctly classified and misclassified instances in our GCN model. The x-axis represents negative uncertainty scores, where lower (more negative) values denote higher model confidence, while values closer to zero indicate lower confidence. The y-axis shows the density of data points at specific uncertainty levels. The correctly classified instances, denoted by the green curve, span a wide range of scores from -350 to 0 with a modest peak density of 0.015, reflecting the model's ability to confidently classify across diverse uncertainty



(a) Accuracy, Precision, and Recall values during GCN model validation.

(b) Uncertainty distribution following GCN model training.

Fig. 4: GCN model evaluation of our proposed ZXNet framework.

TABLE II: Error detection accuracy using ZXNet for different benchmarks and induced errors.

Benchmark	Detection of Induced Error (%)					
	Add 10 CX gates	Remove 10 CX gates	Add 5 CX gates	Remove 5 CX gates	Add 1 CX gate	Remove 1 CX gate
Grover	98.21	97.46	97.1	97.3	96.6	96.18
Random walk	99.16	98.75	97.68	98.24	96.36	97.3
QPE-exact	98.88	98.24	97.25	97.25	96.2	96.01

levels. In contrast, the red curve, representing misclassified instances, is sharply concentrated near -0.1 with a peak density of 0.23, indicating that misclassifications primarily occur at low confidence levels. This separation highlights our model's robustness in distinguishing between confident predictions and uncertain misclassification.

C. Error detection using ZXNet

Accurate equivalence-checking frameworks must be sensitive enough to detect and distinguish even minor discrepancies that lead to non-equivalence between quantum circuits. To evaluate this capability, we assessed error detection using our ZXNet framework. Specifically, we introduced variations in quantum circuits by arbitrarily adding or removing `CX` gates and tested whether ZXNet could identify these changes. The results, summarized in Table II, highlight the framework's accuracy in detecting non-equivalence across various benchmarks. The first column lists the benchmark circuits (these were selected since they are commonly used algorithms with real-world applications), while the subsequent columns show detection accuracy for different types of errors. It should be noted that the errors of addition and removal of `CX` gates were conducted randomly throughout the quantum circuit, and the accuracy of detection was averaged over 10 executions to observe the general performance of ZXNet. `CX` gates were chosen for this study due to their significant impact on functional equivalence, making any modifications to their numbers during transpilation particularly impactful [38]. From the table, it can be observed that **ZXNet can detect errors with an accuracy of up to 99.16%** across the tested benchmarks. Moreover, ZXNet demonstrates remarkable sensitivity, **being able to detect discrepancies caused by the addition or removal of a single `CX` gate with up to 97.3% accuracy**, further underscoring the robustness of our framework.

D. Comparative analysis against state-of-the-art approaches

We compare ZXNet, our learning-based equivalence checking framework, against three state-of-the-art methods: MPO-based, DD-based, and rule-based ZX calculus verification [9], [10], [12]. The comparison evaluates two critical metrics: *scalability* and *runtime efficiency*. Parameterized two-local random circuits with linear entanglement are utilized as benchmarks to maintain consistency, as existing results for MPO-based methods are available exclusively for this circuit class. The results, illustrated in Figure 5, present a scalability analysis, with the x-axis representing the number of qubits and the y-axis denoting runtime. The figure demonstrates that ZXNet is consistently faster compared to state-of-the-art methods, **achieving speedups of up to 62×.** *Please note that since Parameterized Two-local random was the only available benchmark circuit that has been evaluated using all existing approaches*, we utilized it in this section for comparison.

TABLE III: Comparison vs state-of-the-art methods for verification.

Name	Gate Depth	Equivalent			Speedup (%)	Non-equivalent			Speedup (%)
		t_{zx} (s)	t_{dd} (s)	t_{zxnet} (s)		t_{zx} (s)	t_{dd} (s)	t_{zxnet} (s)	
Grover_n9	12482	12.15	0.14	0.1	**28.57**	3.16	8.26	0.31	**90.19**
Random walk_n8	14084	1289.13	0.57	0.3	**47.37**	78.91	62.2	2	**96.78**
QPE-Exact_n39	3823	3.19	>3600	0.32	**89.97**	2.89	>3600	0.3	**86.16**

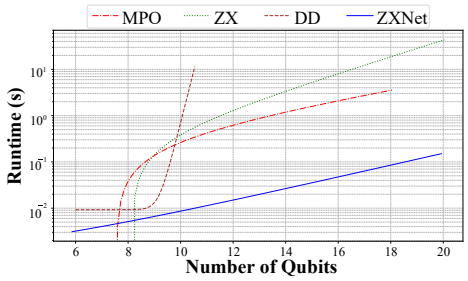


Fig. 5: Scalability comparison of ZXNet with state-of-the-art.

Next, we conduct a more detailed evaluation of ZXNet's performance and compare the results against rule-based ZX calculus and decision diagrams (DD) [10], [12]. The MPO-based method is excluded from this comparison because its results are limited to QFT circuits [9]. For brevity, the evaluation results are limited to three widely used benchmark circuits, previously analyzed by state-of-the-art methods, as depicted in Table III. In this table, columns one to three depict the name, number of qubits, and number of gates of the benchmark circuit, respectively. Columns four through six indicate the time taken by each approach to verify the functional equivalence of the transpiled benchmark with the original in cases when they are equivalent. Column seven showcases the speedup provided by our ZXNet framework over the state-of-the-art methods. Similarly, columns eight to eleven replicate the contents of columns four to seven, but for scenarios when the transpiled benchmark is not equivalent to the existing benchmark. From the table, it is evident that as gate depth increases, ZXNet, consistently outperforms the rule-based ZX verification and DD-based verification, **achieving speedups of up to 96.78%.** These findings demonstrate that ZXNet improves verification efficiency and scales effectively to handle larger and more complex quantum circuits compared to existing methods.

Furthermore, we conducted a comprehensive evaluation of our ZXNet framework across all the scalable benchmarks obtained from MQTBench, with qubit counts ranging from 4 to 120 [34]. Upon evaluation, ZXNet demonstrated its superiority over state-of-the-art methods by **successfully verifying quantum circuits with up to 120 qubits, achieving a 45.83% increase in circuit size, a 5.94% improvement in verification accuracy, and a 42.22% reduction in verification time**, compared to existing approaches. These results establish ZXNet as a more scalable, efficient, and accurate framework for quantum circuit equivalence checking.

VI. Conclusion

In this paper, we introduced ZXNet, a novel equivalence-checking framework for quantum circuits that leverages ZX graph representations in an encoded format for training a GCN model to verify the functional equivalence of quantum circuits. Our experimental results demonstrated that ZXNet achieves notable **accuracy, precision, and recall values of up to 99.4%, 99.2%, and 99.8%**, respectively. Compared to traditional equivalence-checking methods, ZXNet significantly improves computational efficiency, achieving a speedup of up to 62× over existing methods. Moreover, **our model furnishes improvements of up to 45.83%, 42.22%, and 5.94% in terms of scalability, time consumption, and accuracy**, respectively, over state-of-the-art methods. This establishes a robust framework for reliable quantum circuit verification by reducing computational overhead and enhancing the scalability and adaptability to varying circuit complexities.

References

[1] T. Monz et al., "Realization of a Scalable Shor Algorithm," Science, vol. 351, no. 6277, pp. 1068–1070, 2016.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in ACM TC, 1996, pp. 212–219.

[3] J. P. Lowe and K. Peterson, Quantum chemistry. Elsevier, 2011.

[4] Y. Li, M. Tian, G. Liu, C. Peng, and L. Jiao, "Quantum optimization and quantum learning: A survey," Ieee Access, vol. 8, pp. 23 568–23 593, 2020.

[5] C. Lu, S. Kundu, A. Arunachalam, and K. Basu, "Survey on quantum noise-aware machine learning," in 2022 IEEE 15th Dallas Circuit And System Conference (DCAS). IEEE, 2022, pp. 1–2.

[6] E. Wilson, S. Singh, and F. Mueller, "Just-in-time quantum circuit transpilation reduces noise," in 2020 IEEE international conference on quantum computing and engineering (QCE). IEEE, 2020, pp. 345–355.

[7] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 9, pp. 1810–1824, 2020.

[8] X. Hong, M. Ying, Y. Feng, X. Zhou, and S. Li, "Approximate equivalence checking of noisy quantum circuits," in 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 637–642.

[9] A. Sander, L. Burgholzer, and R. Wille, "Equivalence checking of quantum circuits via intermediary matrix product operator," 2024. [Online]. Available: https://arxiv.org/abs/2410.10946

[10] C. Lu, N. Choudhury, U. Banerjee, A. A. Saki, and K. Basu, "Qubec: Boosting equivalence checking for quantum circuits with qec embedding," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 7, pp. 2037–2042, 2024.

[11] L. Burgholzer and R. Wille, "The power of simulation for equivalence checking in quantum computing," in 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020, pp. 1–6.

[12] T. Peham, L. Burgholzer, and R. Wille, "Equivalence checking of quantum circuits with the zx-calculus," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 12, no. 3, pp. 662–675, 2022.

[13] S. Das, S. Kundu, P. Madhusoodhanan, P. Viswanathan Pillai, R. Parekhji, A. Raha, S. Banerjee, S. Natarajan, and K. Basu, "Graph learning-based fault criticality analysis for enhancing functional safety of e/e systems," in Proceedings of the 61st ACM/IEEE Design Automation Conference, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3656498

[14] M. Balcilar, P. Heroux, B. Gauzere, P. Vasseur, S. Adam, and P. Honeine, "Breaking the limits of message passing graph neural networks," in Proceedings of the 38th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 599–608. [Online]. Available: https://proceedings.mlr.press/v139/balcilar21a.html

[15] H. Cardot and O. Lezoray, "Graph of neural networks for pattern recognition," in 2002 International Conference on Pattern Recognition, vol. 2, 2002, pp. 873–876 vol.2.

[16] T. Wong, Introduction to Classical and Quantum Computing. Rooted Grove, 2022. [Online]. Available: https://books.google.com/books?id=M3jqzgEACAAJ

[17] N. S. Yanofsky and M. A. Mannucci, Quantum Computing for Computer Scientists. Cambridge University Press, 2008.

[18] M. A. Nielsen et al., Quantum Computation and Quantum Information. Cambridge University Press, 2010.

[19] B. Coecke and R. Duncan, "Tutorial: Graphical calculus for quantum circuits," in Reversible Computation, R. Glück and T. Yokoyama, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–13.

[20] B. Coecke, "Quantum picturalism," Contemporary Physics, vol. 51, pp. 59–83, 2009.

[21] A. Borgna, S. Perdrix, and B. Valiron, "Hybrid quantum-classical circuit simplification with the ZX-calculus," in Programming Languages and Systems, H. Oh, Ed. Cham: Springer International Publishing, 2021, pp. 121–139.

[22] M. Hanks, M. P. Estarellas, W. J. Munro, and K. Nemoto, "Effective compression of quantum braided circuits aided by zx-calculus," Phys. Rev. X, vol. 10, p. 041030, Nov 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.10.041030

[23] L. Yeh, E. Dasgupta, and E. Winston, "Benchmarking ZX-Calculus Circuit Optimization Against Qiskit Transpilation," ACM SRC Grand Finals Candidates, 2020.

[24] A. Said, M. Shabbir, B. Broll, W. Abbas, P. Völgyesi, and X. Koutsoukos, "Circuit design completion using graph neural networks," Neural Computing and Applications, vol. 35, no. 16, pp. 12 145–12 157, 06/01 2023. [Online]. Available: https://doi.org/10.1007/s00521-023-08346-x

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[26] X. Hong, M. Ying, Y. Feng, X. Zhou, and S. Li, "Approximate equivalence checking of noisy quantum circuits," in 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 637–642.

[27] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2019. [Online]. Available: https://arxiv.org/abs/1810.00826

[28] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," Annals of Physics, vol. 326, no. 1, p. 96–192, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.aop.2010.09.012

[29] X. Hong, X. Zhou, S. Li, Y. Feng, and M. Ying, "A tensor network based decision diagram for representation of quantum circuits," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 27, no. 6, pp. 1–30, 2022.

[30] M. Nägele and F. Marquardt, "Optimizing zx-diagrams with deep reinforcement learning," Machine Learning: Science and Technology, vol. 5, no. 3, p. 035077, Sep. 2024. [Online]. Available: http://dx.doi.org/10.1088/2632-2153/ad76f7

[31] H. Chen, Z. Huang, Y. Xu, Z. Deng, F. Huang, P. He, and Z. Li, "Neighbor enhanced graph convolutional networks for node classification and recommendation," Knowledge-Based Systems, vol. 246, p. 108594, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705122002660

[32] C. Banerjee, T. Mukherjee, and E. Pasiliao, "An empirical study on generalizations of the relu activation function," in Proceedings of the 2019 ACM Southeast Conference, ser. ACMSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 164–167. [Online]. Available: https://doi.org/10.1145/3299815.3314450

[33] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," Bulletin de la Société Vaudoise des Sciences Naturelles, vol. 37, pp. 547–579, 1901.

[34] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," Quantum, 2023, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[35] IBM Research, "Qiskit: Open-Source Quantum Development," Accessed: March 2022. [Online]. Available: https://qiskit.org.

[36] A. Kissinger and J. van de Wetering, "Pyzx: Large scale automated diagrammatic reasoning," arXiv preprint arXiv:1904.04735, 2019.

[37] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," BMC genomics, vol. 21, pp. 1–13, 2020.

[38] M. Bataille, "Quantum circuits of cnot gates: optimization and entanglement," Quantum Information Processing, vol. 21, no. 7, p. 269, 2022.

Machine Learning-Driven STL Generation for Enhancing Functional Safety of E/E Systems

Sanjay Das[1*], Swastik Bhattacharya[1], Anand Menon[1], Shamik Kundu[3], Pooja Madhusoodhanan[2], Prasanth Viswanathan Pillai[2], Rubin Parekhji[2], Arnab Raha[3], Suvadeep Banerjee[3], Suriyaprakash Natarajan[4] and Kanad Basu[1]
[1]*University of Texas at Dallas, TX, USA*; [2]*Texas Instruments, Bangalore, India*; [3]*Intel Corporation, CA, USA*; [4]*Siemens EDA, OR, USA*

***Abstract*—The increasing complexity of safety-critical hardware systems demands advanced methods for ensuring functional safety (FuSa). Traditional techniques like ATPG and BIST are intrusive, requiring additional hardware and disrupting operations, making them unsuitable for in-field testing. To address this, for the first time, we propose a machine learning (ML)-driven automated Self-Test Library (STL) generation for seamless in-field testing during idle periods, ensuring uninterrupted fault detection and high system performance. Utilizing reinforcement learning, the STL generates design-specific test patterns, achieving up to 57.57% improvement in fault coverage and up to 85% efficiency compared to existing pattern-based testing, enhancing FuSa in mission-critical applications.**

***Index Terms*—Functional Safety, Fault Injection Analysis, Test Pattern Generation, Self-Test Library Development.**

I. Introduction

With increasing design complexity, shrinking feature sizes, and accelerated time-to-market requirements, hardware systems are exposed to a broad spectrum of potential failures, primarily due to hardware faults. This has led to the development of Functional Safety (FuSa) standards, such as ISO 26262, which provide a technical framework for detecting and managing faults, ensuring safe operational states [1]. Faults in critical circuits can have severe consequences during operation, with the potential to cause catastrophic failures [2]. For instance, a fault in the control system of an autonomous vehicle could lead to uncontrollable acceleration, resulting in accidents or even loss of life, as shown in Figure 1. The severity of these faults depends on their location within the circuit and the structural characteristics of the design. Fault propagation is closely tied to these structural features, making it crucial to assess both the impact and criticality of faults in relation to their location within the design. This comprehensive approach is imperative for ensuring FuSa in mission-critical applications. A paradigm shift in safety assurance is anticipated with the prospect of machine learning (ML)-driven automation in FuSa. This is set to strengthen the strictness of fault management procedures in sectors that depend on risky, safety-critical hardware systems. With ramifications for the future of autonomous systems, the Internet of Things (IoT), and other mission-critical sectors, this research represents a crucial frontier. In addition to being a tempting opportunity, the ongoing investigation of ML for automated functional safety is also essential to stay up with the complexity of next-generation hardware.

The increasing complexity of modern E/E systems has rendered traditional fault injection and simulation techniques inadequate for ensuring FuSa. These methods face substantial timing overheads, are resource-intensive, and often fail to identify critical design components that could lead to functional failures. Furthermore, they lack the capability to quantify the varying degrees of criticality among circuit nodes, which is essential for prioritizing fault mitigation efforts. Conventional fault analysis techniques, therefore, fall short of addressing the growing intricacy of contemporary systems. To overcome these challenges, ML-based approaches have emerged as a necessary solution for enhancing FuSa [3]–[5]. By efficiently generating high-quality test patterns, these methods not only improve fault coverage but also provide a more systematic framework for detecting critical faults within complex designs. The use of graph-based analysis, specifically, has proven effective in understanding the structural features of E/E systems, allowing for more precise fault criticality analysis [6].

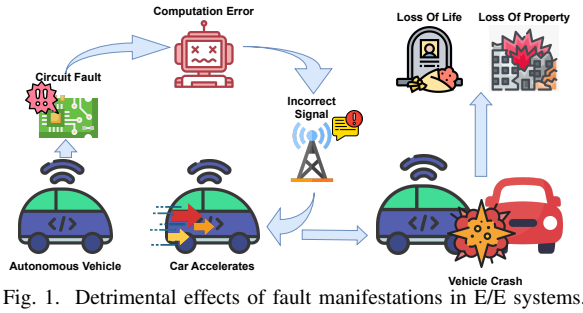


Fig. 1. Detrimental effects of fault manifestations in E/E systems.

The automotive functional safety standard ISO 26262 [1] and the industrial standard IEC 61508 [7] stipulate the necessity for evaluating both permanent and transient fault coverage during runtime operations. Attaining the requisite level of permanent fault coverage presents considerable challenges. There are three principal mechanisms available for achieving this coverage: (i) hardware redundancy, (ii) hardware-based Built-In Self-Test (BIST) mechanisms, and (iii) software-based BIST mechanisms. **While hardware redundancy is an effective method, it is generally eschewed in non-mission-critical modules due to the substantial area overhead it incurs. Hardware-based BIST mechanisms are frequently utilized; however, their implementation may not be feasible under certain conditions [8], [9].** For instance, a given module may not have been originally designed with BIST capabilities. Furthermore, such mechanisms necessitate that the device transitions into a scan/Automatic Test Pattern Generation (ATPG) mode.

The transition to ATPG mode requires the preservation of the complete Intellectual Property (IP) or module context prior to testing, along with its restoration afterward, to ensure seamless execution of the application. **This procedure may prove impractical within the constraints of the designated testing time window.** This invasive reconfiguration of the system into a non-operational state during testing not only disrupts system functionality but also risks overtesting, resulting in downtime and inefficiencies. Such interruptions render these techniques unsuitable, particularly in safety-critical systems

*Corresponding author: Sanjay Das (Email: *sanjay.das@utdallas.edu*)
This work is supported by the Semiconductor Research Corporation (SRC GRC Task: 3192.001) and NSF award # 2343652.

like automotive applications, where continuous operation is essential. **Therefore, non-scan-based methods, such as software-based self-test (SBST) mechanisms, offer a more flexible and practical approach to achieving the essential fault coverage required in contemporary systems [10].**

For in-field testing, where halting system operations is impractical, the invasive nature of design-for-testability (DfT) techniques presents a significant challenge. In contrast, SBST approaches, such as self-test libraries (STLs), provide a more efficient and less invasive solution [11], [12]. STLs execute functional programs to detect faults without altering the design or interrupting system operations. These libraries can be run at full operational speed, enabling fault detection during normal operation without any disruption. Moreover, STLs can divide the testing process into small segments during idle periods, making them ideal for in-field applications. This method not only prevents overtesting but also maintains system integrity and supports long-term functional safety, in line with stringent standards like ISO 26262. Therefore, the use of STLs allows for continuous and non-intrusive fault detection, ensuring the system's reliability and safety throughout its operational lifecycle without compromising performance or introducing inefficiencies. **Commercial STLs are proprietary, device-specific, and typically handcrafted for each target design. Therefore, it becomes incumbent to automate the STL generation based on the functional test patterns in order to optimize SBST overhead.** ML-based pattern generation can aid in this automated STL-based testing by increasing the fault coverage. This can reduce the memory requirement to periodically run tests on the device when it is in use.

In this work, we propose a novel framework for automated STL generation to ensure FuSa in mission-critical applications by enabling in-field testing with minimal operational disruption. Our framework introduces an innovative ML-based approach for generating functional test patterns without relying on scan-based test mode assertions, thereby avoiding operational disruptions. It leverages advanced machine learning techniques, such as Variational Autoencoders (VAE) and Reinforcement Learning (RL), to optimize test pattern generation. **Since this is the first automated STL generation technique, our approach is evaluated against random pattern generation methods.** Our key contributions in this work include:

- We introduce a novel, automated framework for STL generation that ensures FuSa in mission-critical applications by enabling in-field testing with minimal operational disruption.
- We propose an ML-driven framework to generate functional test patterns without the need for traditional scan-based testing methods.
- The proposed ML framework integrates VAE with RL, using VAE to initialize the RL-based pattern optimization process, ensuring effective exploration of the solution space.
- The proposed solution delivers high operational fault coverage, with up to 57.57% improvement over random pattern-based testing, while avoiding the pitfalls of test mode assertion and system downtime associated with ATPG-based techniques.

II. Background

In this section, we provide the essential background necessary to elucidate our proposed approach for machine learning-based test pattern generation and the development of STLs.

A. Self-Test Library

The Self-Test Library (STL) in VLSI testing is a suite of tools and methodologies designed to facilitate efficient on-chip testing of integrated circuits (ICs) [12]. It includes techniques like Built-In Self-Test (BIST) and non-scan-based testing to detect faults and ensure circuit functionality without external testing equipment. The STL allows for the embedded test circuitry to perform in-situ tests, which is particularly beneficial for detecting manufacturing defects and operational faults in complex VLSI circuits, reducing reliance on external testers and equipment.

Compared to traditional methods like BIST and ATPG, STLs offer advantages in terms of flexibility and scalability. BIST introduces test pattern generators and output response analyzers within the chip, providing autonomous testing but often at the cost of significant area, power, and timing overheads. ATPG, which typically uses scan chains to shift test patterns, also adds overhead in terms of area and performance but can be limited by the length and complexity of the scan chains. In contrast, STLs, especially those leveraging non-scan-based techniques, provide a more scalable solution that can adapt to both sequential and combinational circuits, reducing external resource dependence. For non-scan-based testing, techniques such as Linear Feedback Shift Registers (LFSRs) and pseudorandom pattern Generators (PRPGs) are employed to generate high fault coverage test patterns [13], [14]. These methods are effective in achieving high coverage with minimal area and test time overheads, as LFSRs and PRPGs efficiently produce large sets of pseudo-random patterns that can detect a wide range of faults. While these techniques do not completely eliminate the need for scan chains in some designs, they help significantly reduce the overhead associated with traditional scan-based testing.

Thus, STLs provide a robust mechanism to enhance the testability and reliability of VLSI circuits by combining the efficiency of on-chip testing with reduced dependence on external testing resources. This makes it a preferred choice for modern IC designs, particularly in domains requiring high reliability, such as automotive and aerospace systems [15], [16]. The manual and proprietary nature of commercial STLs, tailored specifically for individual designs, creates significant challenges in scalability and standardization. To overcome these issues, this work, for the first time, introduces an automated STL generation framework powered by machine learning, aiming to improve FuSa in mission-critical systems.

B. Variational Autoencoder

A Variational Autoencoder (VAE) is a type of generative model used for unsupervised learning, built on the foundation of traditional autoencoders [17]. To understand VAEs, it is essential to first grasp the concept of a standard autoencoder. An autoencoder is a neural network designed to learn a compressed, lower-dimensional representation (encoding) of input data, such as images or text, and then reconstruct the input from this representation. The goal is to learn useful features or patterns from the data by minimizing the difference between the original input and its reconstruction. This process is driven by a loss function (to represent the said difference), usually the mean squared error or cross-entropy.

The VAE extends this idea by introducing a probabilistic interpretation of the latent space (the space where the compressed data representation resides). Instead of learning fixed values for the compressed representation, the VAE assumes that the data is generated from an underlying probability distribution. In this way, the encoder network learns to output the parameters of this distribution (typically a mean and variance) rather than a single deterministic point. A key innovation in VAEs is the reparameterization trick, which allows for back-propagation through the stochastic sampling of the latent variables by expressing the randomness in a differentiable way. During training,

the VAE attempts to reconstruct the input data accurately and ensure that the learned latent space follows a known probability distribution (often a normal distribution). This is achieved by optimizing a loss function that includes two components: the reconstruction loss, which measures how well the VAE can recreate the input data, and a regularization term called the Kullback-Leibler divergence [18]. This term ensures that the learned distribution is close to the prior distribution, which allows for efficient sampling of new, unseen data points from the model after training. This combination of learning a probabilistic latent space and reconstruction capability makes VAEs a powerful tool for generating new data and discovering underlying patterns in complex datasets, which we exploit for our proposed test pattern generation procedure.

C. Reinforcement Learning

Reinforcement Learning (RL) is a specialized area of machine learning that focuses on enabling agents to learn optimal behaviors through interactions with their environment [19]. Distinct from supervised learning, where models learn from labeled datasets, RL requires agents to make decisions within uncertain and dynamic settings to maximize cumulative rewards over time. The core components of an RL framework include the agent, the environment, states, actions, and rewards. In this paradigm, the agent represents the learner or decision-maker, while the environment encompasses all external factors influencing the agent. States reflect the current condition of the environment, and actions are the choices available to the agent that may alter this condition. Upon executing an action, the agent receives a reward, quantifying the immediate benefit derived from that action. The primary objective is to derive a policy—a mapping from states to actions—that maximizes expected cumulative rewards.

Reinforcement Learning algorithms typically utilize trial-and-error methods to explore the environment, balancing exploration (trying new actions) with exploitation (selecting known advantageous actions). Common strategies to achieve this balance include the epsilon-greedy approach and Upper Confidence Bound (UCB) algorithms. Notable RL methodologies encompass model-free techniques, such as Q-learning and Policy Gradient methods, and model-based approaches, which involve constructing a model of the environment's dynamics. Q-learning, a value-based method, estimates the expected utility of actions in given states, while Policy Gradient methods directly optimize the policy, accommodating complex action spaces.

Reinforcement Learning has demonstrated effectiveness across diverse domains, including robotics, game playing, and resource management, due to its capacity for handling intricate decision-making tasks and adapting to dynamic environments. Therefore, we translate our STL generation process into an RL-based optimization for enhancing FuSa analysis.

III. Methodology

In this section, we present a comprehensive methodology for generating STLs comprising high-quality functional test patterns aimed at enhancing fault coverage in hardware systems. Our approach integrates advanced machine learning techniques, including VAE and RL, to optimize test pattern generation as illustrated in Figure 2. Moreover, this framework incorporates an STL generation process that enables non-intrusive, in-field testing without halting system operations.

A. VAE-Based Learning for Functional Pattern Generation

In this section, we describe the process of developing and using a VAE to generate new test patterns for high fault coverage in E/E

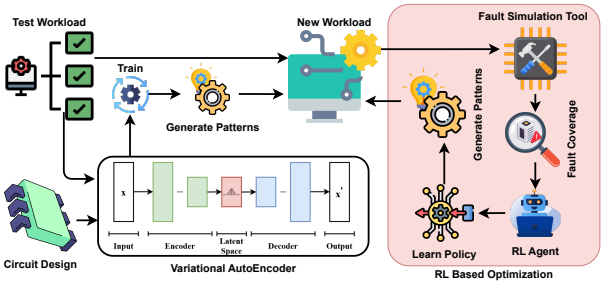


Fig. 2. Overview of the proposed test pattern generation framework.

systems. The prposed approach leverages both the structure of the design and the existing set of functional test patterns. The primary goal is to learn the underlying distribution of the test patterns and generate new patterns that effectively target critical regions of the design.

We start by preparing the input data, which includes a set of test patterns P and the design netlist graph G. The netlist graph is first encoded into a feature representation, G_{features}, using a graph neural network, where the node features represent the circuit components and the edges capture the connectivity. This encoding allows us to incorporate the structural information of the design into the learning process. Once G_{features} is obtained, we concatenate it with each test pattern P from the pattern set. This combined representation $[P, G_{\text{features}}]$ is then passed through the encoder of the VAE, which transforms it into a latent vector z by approximating the posterior distribution using a multivariate Gaussian:

$$z \sim q_\phi(z|[P, G_{\text{features}}]) = \mathcal{N}(\mu_\phi([P, G_{\text{features}}]), \sigma_\phi([P, G_{\text{features}}])) \tag{1}$$

Here, μ_ϕ and σ_ϕ are the mean and standard deviation vectors learned by the encoder, parameterized by ϕ.

Next, we sample the latent vector z from this distribution and pass it through the decoder to generate a reconstructed test pattern $\hat{P}$:

$$\hat{P} = \text{Decoder}_\phi(z) \tag{2}$$

This process allows the VAE to reconstruct patterns that are similar to the original ones but with variability introduced by the latent space, thus encouraging the generation of novel test patterns.

To ensure that the generated patterns are meaningful, we define a reconstruction loss L_{recon}, which measures the difference between the original patterns P and the reconstructed patterns $\hat{P}$. This is typically computed as the mean squared error:

$$L_{\text{recon}} = \frac{1}{B} \sum_{i=1}^{B} \|P_i - \hat{P}_i\|^2 \tag{3}$$

where B is the batch size of patterns processed during training. In addition to the reconstruction loss, we impose a Kullback-Leibler (KL) divergence term L_{KL} to regularize the latent space. The KL divergence ensures that the learned distribution $q_\phi(z|P)$ remains close to a standard normal distribution $p(z)$:

$$L_{\text{KL}} = D_{\text{KL}}(q_\phi(z|P) \| p(z)) \tag{4}$$

The total loss L that we minimize during training is a combination of the reconstruction loss and the KL divergence, weighted by a hyper-parameter λ:

$$L = L_{\text{recon}} + \lambda L_{\text{KL}} \tag{5}$$

This regularization not only prevents overfitting but also ensures that the latent space is smooth and can generate a wide variety of test

Algorithm 1 VAE Learning from Pattern Set

Input: Pattern set $\mathcal{P}$, design netlist graph $\mathcal{G}$, learning rate α, number of epochs E, batch size B

1: Initialize VAE model VAE_ϕ with parameters ϕ
2: Initialize optimizer for VAE_ϕ
3: **for** each epoch $e = 1$ to E **do**
4: Shuffle the pattern set $\mathcal{P}$
5: **for** each mini-batch of patterns P_b from $\mathcal{P}$ **do**
6: Encode $\mathcal{G}$ into a graph feature representation G_{features}
7: **for** each pattern $P \in P_b$ **do**
8: Concatenate pattern P with G_{features}
9: Pass data through VAE encoder to obtain latent vector z
10: Decode latent vector z to reconstruct pattern $\hat{P}$
11: **end for**
12: Compute reconstruction loss $\mathcal{L}_{\text{recon}}$ between P_b and $\hat{P}_b$
13: Compute KL divergence loss $\mathcal{L}_{\text{KL}}$
14: Total loss $\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda\mathcal{L}_{\text{KL}}$
15: Back-propagate and update VAE parameters ϕ
16: **end for**
17: **end for**
18: **Output**: Trained VAE model VAE_ϕ

Algorithm 2 Reinforcement Learning for Pattern Generation

Input: Design netlist $\mathcal{D}$, fault analysis tool $\mathcal{F}$, pre-trained VAE model VAE_ϕ

1: Initialize policy model π_θ with random weights θ
2: Initialize best fault coverage $C_{\text{best}} \leftarrow 0$
3: Initialize reward $r \leftarrow 0$
4: **for** each episode $t = 1$ to T **do**
5: Sample $z \sim \mathcal{N}(0, I)$ from the VAE's latent space
6: Decode z to generate an initial pattern $P_0 = \text{VAE}_\phi(z)$
7: Apply pattern P_0 compute initial fault coverage C_0
8: **for** each step $k = 1$ to K **do**
9: Perturb P_{k-1} using π_θ to generate P_k
10: Apply pattern P_k to compute new fault coverage C_k
11: Calculate reward: $r_k = C_k - C_{k-1}$
12: Update policy model π_θ using the reward r_k
13: **if** $C_k > C_{\text{best}}$ **then**
14: Update $C_{\text{best}} \leftarrow C_k$
15: **end if**
16: **if** reward $r_k \leq 0$ (no improvement) **then**
17: Break the loop and stop perturbing
18: **end if**
19: **end for**
20: **end for**
21: **Output**: The pattern P_{best} that achieves the highest coverage C_{best}

patterns. By training the VAE on the existing pattern set P, we effectively learn a latent space that captures the essential features of the design and its corresponding test patterns. Once training is complete, we can generate new test patterns by sampling from the latent space $z \sim \mathcal{N}(0, 1)$ and decoding them into test patterns using the decoder ($\hat{P}_{\text{new}} = \text{Decoder}_\phi(z)$).

The VAE-based learning and pattern generation is outlined in Algorithm 1, that begins by initializing the VAE model with parameters ϕ and setting up an optimizer for the VAE model (lines 1-2). Then, for each epoch from 1 to E, the pattern set P is shuffled to ensure randomness in the training process (lines 3-4). Next, for each mini-batch of patterns P_b drawn from the shuffled pattern set, the design netlist graph G is encoded into a graph feature representation G_{features} (lines 5-6). Subsequently, each pattern P in the mini-batch is concatenated with G_{features} to create the input data for the VAE encoder (lines 7-8). The concatenated data is then passed through the VAE encoder to generate a latent vector z, which is sampled from the encoder (line 9). The latent vector is decoded to reconstruct the pattern $\hat{P}$ (line 10). Subsequently, the reconstruction loss L_{recon} is calculated between the original patterns and their reconstructed versions (line 12), followed by the computation of the KL divergence loss L_{KL} for latent space regularization (line 13). The total loss L is then computed as the sum of the reconstruction loss and the regularization loss (line 14). Finally, the VAE parameters ϕ are updated using back-propagation through the optimizer (line 15). The process repeats over the mini-batches for each epoch until the VAE model is fully trained, at which point the trained VAE model VAE_ϕ is output (lines 16-18).

These newly generated patterns are tailored to the specific characteristics of the design and provide enhanced test coverage by targeting regions of the design that may not have been fully excited by the original patterns.

B. RL-based Functional Test Pattern Optimization

In this section, we elaborate on the integration of a trained VAE with a RL framework to efficiently generate high-quality functional test patterns. The VAE, which learns the underlying distribution of existing test patterns, is used to sample potential new patterns, while the RL agent optimizes the selection and refinement of these patterns to maximize fault coverage. This combined approach leverages the generative capabilities of the VAE and the optimization power of RL to dynamically adapt the pattern generation process, improving overall testing efficiency.

The VAE model samples patterns from a latent space, represented by $z \sim \mathcal{N}(0, 1)$, and decodes this latent vector into test patterns $\hat{P} = \text{Decoder}(z)$. These generated patterns are then evaluated by the RL agent, which learns a policy $\pi(a_t|s_t)$ to decide how to use the patterns in the context of the design. Here, s_t represents the system state at time step t, which encodes critical design information, such as circuit netlist features G_{features} and current fault coverage metrics. The agent's action a_t, chosen from a predefined action space, can involve sampling a new test pattern from the VAE, modifying an existing one, or reapplying a previously used pattern, depending on the current state.

The reward function for the RL agent is tied to improvements in fault coverage. At each time step, the reward r_t is calculated as the increase in coverage from the previous step:

$$r_t = \text{Coverage}(t) - \text{Coverage}(t-1) \tag{6}$$

This reward structure encourages the agent to prioritize actions that lead to significant fault coverage improvements, penalizing ineffective or redundant actions. The RL agent's goal is to maximize the cumulative discounted reward R_t, defined as:

$$R_t = \mathbb{E}\left[\sum_{k=t}^{T} \gamma^{k-t} r_k\right] \tag{7}$$

where γ is the discount factor. Policy gradient methods, such as Proximal Policy Optimization (PPO), are employed to iteratively refine the agent's policy $\pi(a_t|s_t)$, ensuring that the agent learns to select actions that yield optimal test coverage.

The RL-based test pattern optimization is detailed in the Algorithm 2, which starts by initializing the policy model π_θ with random

weights θ, setting the best fault coverage C_{best} to 0, and initializing the reward r to 0 (lines 1-3). For each episode from 1 to T, a latent vector z is sampled from the VAE's latent space, where z follows a normal distribution $N(0, I)$ (lines 4-5). This latent vector is decoded by the pre-trained VAE model VAE_ϕ to generate an initial test pattern P_0 (line 6). The generated pattern P_0 is applied to the design netlist D, and its initial fault coverage C_0 is computed using the fault analysis tool F (line 7). Next, for each step from 1 to K, the current pattern P_{k-1} is perturbed by the policy model π_θ to generate a new pattern P_k (line 9). This new pattern P_k is applied to the design netlist D, and the updated fault coverage C_k is calculated using the fault analysis tool F (line 10). The reward r_k is then computed as the difference between the new fault coverage C_k and the previous fault coverage C_{k-1} (line 11). The policy model π_θ is updated based on the reward r_k (line 12). If the new fault coverage C_k surpasses the best fault coverage C_{best}, the best fault coverage C_{best} is updated to C_k (lines 13-14). However, if the reward r_k is less than or equal to 0, meaning no improvement in fault coverage, the loop is terminated, and further perturbation stops (lines 16-17). This process repeats for each episode until all episodes are completed (lines 19-20). Finally, the algorithm outputs the pattern P_{best} that achieved the highest fault coverage C_{best} (line 21).

C. Self-Test Library (STL) Generation for In-Field Testing

In this section, we outline the methodology for generating a STL from functional test patterns to facilitate efficient in-field testing of VLSI designs. The STL enables continuous fault detection and real-time fault coverage evaluation without halting system operations, as illustrated in the Figure 3.

The process begins with the utilization of functional test patterns, $P = \{p_1, p_2, \ldots, p_N\}$, generated by the RL-VAE approach. Each pattern p_i consists of an input-output mapping derived from the design's expected behavior, $f_{\text{design}} : I \rightarrow O$, where I represents the set of input signals and O the corresponding output signals.

These patterns are transformed into machine-executable instructions by translating the RTL description of the design into a higher-level programming language, typically C or C++. The transformation can be expressed as:

$$C_{\text{design}} = \mathcal{T}(V_{\text{design}}) \tag{8}$$

where $\mathcal{T}$ represents the transformation function that converts the Verilog representation V_{design} into its C/C++ equivalent C_{design}.

Once transformed, the STL generation process automates the creation of testbenches. For each pattern p_i, the corresponding input vector I_i is assigned to the input signals of the testbench, which drives the evaluation of the C++ design model. The process involves repeatedly evaluating the system's response to each input vector:

$$O_i^{\text{actual}} = f_{\text{STL}}(I_i) \tag{9}$$

where f_{STL} represents the behavior of the design under test as executed by the generated STL. The actual output O_i^{actual} is compared against the expected output O_i^{expected} to detect discrepancies or faults:

$$\Delta O_i = O_i^{\text{actual}} - O_i^{\text{expected}} \tag{10}$$

If $\Delta O_i \neq 0$, the test reveals a fault.

The STL is integrated into the system's memory, enabling non-intrusive execution during idle cycles or specific test periods. This reduces the performance overhead by running tests concurrently with normal operations.

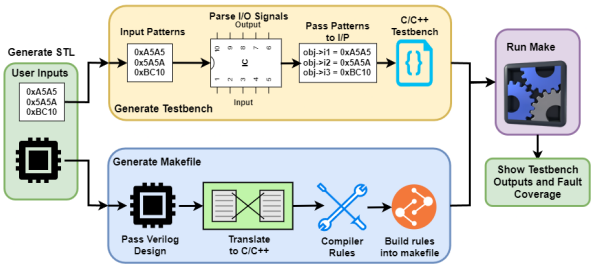


Fig. 3. Flow of processes to generate STL.

IV. Experimental Evaluation

A. Experimental Setup

The proposed framework is evaluated using four distinct designs: a Synchronous Dynamic Random Access Memory (SDRAM) Controller, the OpenCores Memory Controller Main Timing (OCMCMT) module, the OpenRISC 1200 instruction fetch (ORIF) module, and a Single Port Synchronous RAM (ORSPRAM) module. These designs represent a variety of circuitry and critical functionalities, providing a comprehensive basis for evaluating the framework's effectiveness.

To construct the initial dataset, each design's register-transfer level (RTL) representation is synthesized into a gate-level netlist using Synopsys Design Vision. Fault injection experiments are then conducted using Cadence Xcelium [20], which simulates the circuit and injects faults across different input vectors (workloads). Fault detection reports are generated, detailing fault criticalities and coverage metrics. These reports are compiled into a dataset that is used to train a VAE, which learns the latent distribution of test patterns and generates new synthetic patterns. Upon training, the VAE is used in the RL-based framework to generate high-quality test patterns aimed at maximizing fault coverage. For generating the STL, we employ Verilator [21], an open-source tool known for its efficiency in converting high-level hardware designs into cycle-accurate simulations, thus facilitating the rapid verification of the design during STL development.

B. Performance Metrics

Model performance is assessed using fault coverage, representing the percentage of detected faults by the applied functional test patterns. We also use the number of patterns needed as another metric to showcase the efficiency of the pattern-generation process. A higher fault coverage indicates better model performance, while a lower number of required patterns shows the efficiency of the pattern generation approach.

C. STL Generation

The process of generating an STL involves the use of a script that takes a Verilog RTL design and an input pattern set as its primary inputs, producing four essential output files. To ensure compatibility, these input files must adhere to a specific format. Below, we detail the STL generation procedure using the ORIF design as an example.

In this process, great care is taken to avoid conflicts with the module's port names. Each signal in the generated files is distinctly separated, and comments are carefully structured to avoid the use of reserved keywords such as *input* and *output*, minimizing the risk of errors or ambiguities during module port parsing. This precaution ensures a smooth and error-free generation process.

The four files produced by the script are the `Makefile`, `Makefile_obj`, `inputs.vc`, and a C++ testbench. The `Makefile` is responsible for compiling the C++ executable using Verilator, that converts RTL designs into C++ models. It handles

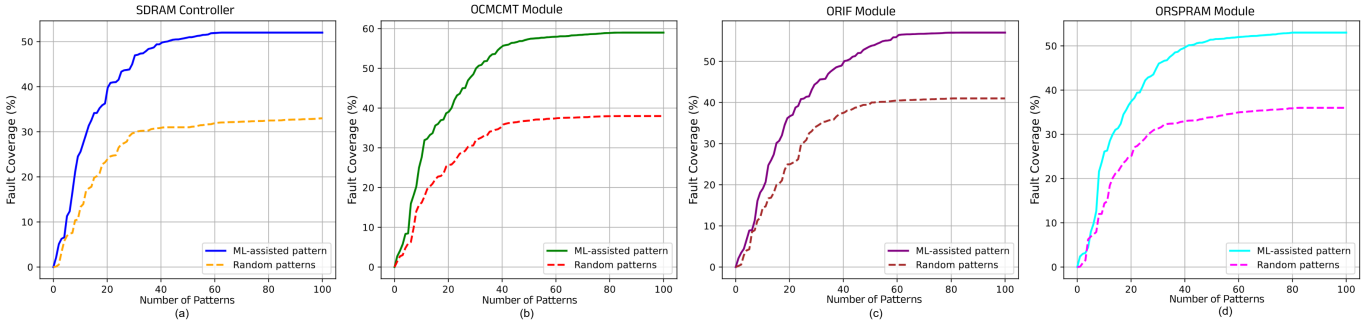


Fig. 4. Fault coverage analysis across various designs: (a) SDRAM controller, (b) OCMCMT module, (c) ORIF module and (d) SPRAM module.

Snippet of the generated testbench

```
void test_or1200_if() {
    Vor1200_if *tb = new Vor1200_if;
    // Initialize all input signals to 0
    tb->clk = 0;  tb->rst = 0;
    tb->icpu_dat_i = 0;
    tb->icpu_adr_i = 0;
    ...
    // Apply input patterns
    // Pattern 1
    tb->rst = 1;
    tb->icpu_dat_i = 0xAAAA;
    tb->icpu_adr_i = 0x1000;
    ...
    tb->clk = !tb->clk;
    tb->eval();
    // Simulate time progression
    Verilated::timeInc(1);
    //Add more test patterns
    delete tb;}
```

the setup of basic compiler flags and initiates the Verilation process, which translates RTL signals into C++ object representations. The `Makefile_obj` file manages the building and linking of the Verilated design objects referenced during the compilation process. It specifies the necessary compilation flags, linking rules, and execution procedures for the C++ hardware executables. Additionally, the script generates an `inputs.vc` file, which imports the required libraries and directories. This file is modular, allowing for easy modification and management of project imports, ensuring that all dependencies remain organized.

Finally, the input pattern set is used to generate a C++ testbench, which is employed to simulate the RTL design and conduct fault coverage analysis. Paired with the `Makefile` and `Makefile_obj`, the testbench enables the complete execution of the STL, simulating the functionality of the design based on the provided input test patterns. Through this simulation process, the STL aids in verifying the design's fault coverage and ensuring its operational integrity. We will open source the framework upon acceptance.

D. Fault Coverage Results

In our evaluation, we examined four distinct hardware designs to assess the efficacy of our fault coverage improvement method compared to random patterns for a specific number of patterns, which is set to 100 for our analysis, to reduce the overhead (often associated with ATPG-based techniques as discussed in Section I). Table I compares fault coverage achieved by two pattern generation methods: ML-assisted and random pattern generation. It includes design details, the number of gates and flip-flops, fault coverage for each method, and the improvement provided by the ML-assisted approach. The

TABLE I
COMPARISON OF FAULT COVERAGE ANALYSIS.

Design	Gates	Flip-flops	Maximum fault coverage (%) with 100 patterns		# ML functional patterns to match random coverage
			Random	ML	
SDRAM Controller	312	91	33%	52%	15
OCMCMT Module	873	188	38%	59%	18
ORIF Module	238	67	41%	57%	25
ORSPRAM Module	625	132	36%	53%	19

results emphasize the superior performance of ML-assisted pattern generation in enhancing fault detection.

For the SDRAM Controller, random patterns achieved 33% fault coverage with 100 test patterns. In contrast, ML-assisted patterns improved fault coverage to 52%, as shown in Figure 4(a). Moreover, using only 15 ML-assisted patterns achieved the same 33% fault coverage as 100 random patterns, demonstrating an 85% efficiency improvement.

The OCMCMT module showed 38% fault coverage with random patterns, while ML-assisted patterns increased coverage to 59%, as illustrated in Figure 4(b). Remarkably, 18 ML-assisted patterns matched the 38% coverage of random patterns, yielding an 82% efficiency gain while maintaining baseline performance.

For the ORIF module, random patterns achieved 41% fault coverage, which improved to 57% with ML-assisted patterns, as shown in Figure 4(c). Similarly, the SPRAM module exhibited 36% fault coverage with random patterns, while ML-assisted patterns raised coverage to 53%, as depicted in Figure 4(d).

The comparative analysis presented in Table I demonstrates that ML-assisted pattern generation significantly improves fault coverage across all evaluated designs. These improvements include up to a 57.57% increase in fault coverage compared to random patterns and an efficiency enhancement of up to 85%. These results underscore the effectiveness of ML-assisted methodologies in optimizing fault detection capabilities in integrated circuit designs.

V. CONCLUSION

The proposed ML-driven STL generation framework addresses the limitations of traditional fault detection techniques by enabling seamless, non-intrusive in-field testing for safety-critical hardware systems. Unlike commercial STLs, our approach automates the STL generation, enabling less manual effort. By leveraging reinforcement learning, the STL generates design-specific test patterns that achieve up to a 57.57% improvement in fault coverage and up to 85% testing efficiency compared to random pattern-based methods. This approach ensures uninterrupted system operation during fault detection, making it highly suitable for mission-critical domains such as automotive, aerospace, and medical devices. These results demonstrate the framework's potential to significantly enhance functional safety in increasingly complex hardware systems.

References

[1] R. Palin, D. Ward, I. Habli, and R. Rivett, "Iso 26262 safety cases: Compliance and assurance," 2011.

[2] A. Chaudhuri *et al.*, "Fault-criticality assessment for ai accelerators using graph convolutional networks," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1596–1599.

[3] M. Reddy, B. Satyanarayana, M. Ravi, P. Krishnaiah, C. Dileep, and B. Annapoorna, "Machine learning-based fault tolerance techniques for vlsi circuit design," in *International Conference on Data Science, Machine Learning and Applications*. Springer, 2023, pp. 1359–1369.

[4] Y. Wang, J. Xiao, Z. Wei, Y. Zheng, K.-T. Tang, and C. H. Chang, "Security and functional safety for ai in embedded automotive system–a tutorial," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[5] N. Akter, M. R. Siddiquee, M. Shur, and N. Pala, "Ai-powered terahertz vlsi testing technology for ensuring hardware security and reliability," *IEEE Access*, vol. 9, pp. 64 499–64 509, 2021.

[6] S. Das, S. Kundu, P. Madhusoodhanan, P. Viswanathan Pillai, R. Parekhji, A. Raha, S. Banerjee, S. Natarajan, and K. Basu, "Graph learning-based fault criticality analysis for enhancing functional safety of e/e systems," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[7] R. Bell, "Introduction to iec 61508," in *Acm international conference proceeding series*, vol. 162. Citeseer, 2006, pp. 3–12.

[8] S. Shibahara, "Functional safety soc for autonomous driving," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, 2018, pp. 1–8.

[9] C. Eychenne and Y. Zorian, "An effective functional safety infrastructure for system-on-chips," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2017, pp. 63–66.

[10] A. Dutta, M. Shah, G. Swathi, and R. A. Parekhji, "Design techniques and tradeoffs in implementing non-destructive field test using logic bist self-test," in *2009 15th IEEE International On-Line Testing Symposium*. IEEE, 2009, pp. 237–242.

[11] L. Anghel, R. Cantoro, R. Masante, M. Portolan, S. Sartoni, and M. S. Reorda, "Self-test library generation for in-field test of path delay faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4246–4259, 2023.

[12] R. Cantoro, P. Girard, R. Masante, S. Sartoni, M. S. Reorda, and A. Virazel, "Self-test libraries analysis for pipelined processors transition fault coverage improvement," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2021, pp. 1–4.

[13] K. Devika and R. Bhakthavatchalu, "Design of reconfigurable lfsr for vlsi ic testing in asic and fpga," in *2017 international conference on communication and signal processing (ICCSP)*. IEEE, 2017, pp. 0928–0932.

[14] T. Garbolino, "A new, fast pseudo-random pattern generator for advanced logic built-in self-test structures," *Applied Sciences*, vol. 11, no. 20, p. 9476, 2021.

[15] P. Bernardi, R. Cantoro, A. Floridia, D. Piumatti, C. Pogonea, A. Ruospo, E. Sánchez, S. De Luca, and A. Sansonetti, "Non-intrusive self-test library for automotive critical applications: Constraints and solutions," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 920–923.

[16] F. Pratas, T. Dedes, A. Webber, M. Bemanian, and I. Yarom, "Measuring the effectiveness of iso26262 compliant self test library," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2018, pp. 156–161.

[17] L. Pinheiro Cinelli, M. Araújo Marins, E. A. Barros da Silva, and S. Lima Netto, "Variational autoencoder," in *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer, 2021, pp. 111–149.

[18] T. Kim, J. Oh, N. Kim, S. Cho, and S.-Y. Yun, "Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation," *arXiv preprint arXiv:2105.08919*, 2021.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[20] Cadence, "Xcelium fault simulator," 2024, https://www.cadence.com/en_US/home/training/all-courses/86246.html, Last accessed on 2024-09-10.

[21] W. Snyder, "Verilator 4.0: open simulation goes multithreaded," in *Open Source Digital Design Conference (ORConf)*, 2018.

ANGraph: A GNN-Based Performance Prediction Framework for Asynchronous Neuromorphic Hardware

Yuan Hua†, Jian Zhang†, Jilin Zhang, Xiang Zhang, and Hong Chen*
School of Integrated Circuits, Tsinghua University
Beijing China
hongchen@tsinghua.edu.cn

***Abstract*—Design space exploration (DSE) through system-level simulation is essential for designing energy-efficient asynchronous neuromorphic hardware, which is increasingly promising in edge AI applications. However, there are significant mismatches between system-level predictions and gate-level simulations, resulting in low precision when predicting performance during the DSE process for asynchronous neuromorphic hardware. To address this issue, we put forward ANGraph, a graph neural network (GNN)-based performance prediction framework for asynchronous neuromorphic hardware. In the ANGraph framework, we transform the intermediate representation of system-level simulations into graphs, collect gate-level circuit simulation results to build benchmarks with over one million samples, and train a GNN model to predict hardware latency for asynchronous neuromorphic hardware. Additionally, we use a residual network (ResNet)-based method to predict the power consumption of asynchronous neuromorphic hardware. We evaluate these two models on additional datasets without extra training across different scales, process nodes, and traffic patterns of input data. Compared to the latency predictions from the state-of-the-art simulator, we improve the R-square score by 0.69 and reduce root mean square error (RMSE) by 76% on average across all datasets. We also achieve an R-square score of 0.98 and a mean absolute percentage error (MAPE) of 0.88% for the power consumption prediction task. The benchmarks and models are available at https://github.com/HuaGuaiGuai/ANGraph.**

***Index Terms*—asynchronous neuromorphic hardware, performance prediction, graph neural networks, benchmark**

I. Introduction

Recently, most neuromorphic hardware has been designed with asynchronous circuits to improve energy efficiency, exploiting their event-driven feature to match sparse computation of spikes, such as Loihi2 [1], ANP-I [2], ANP-G [3], and many others in [4, 5, 6]. To assist in efficient design space exploration (DSE) for asynchronous neuromorphic hardware, system-level simulators such as CanMore [7] and TrueAsync [8] have been developed.

The TrueAsync simulator is a fully asynchronous system-level simulator developed with an actor-based concurrent programming model, offering over 2× runtime speedup compared to the cycle-accurate CanMore simulator. However, even though manufacturing process-related circuit details are considered in the modeling of these simulators, their system-level performance prediction results remain significant mismatches against gate-level simulations of circuit netlists, as illustrated in Figure 1.

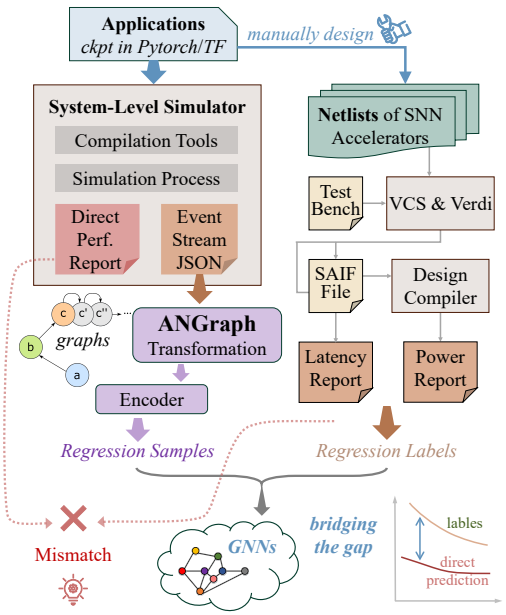


Fig. 1. The overall performance prediction workflow.

Predicting performance for asynchronous neuromorphic hardware using system-level simulations is challenging for two main reasons. First, many gate-level circuit details are ignored in system-level modeling to reduce runtime, resulting in inconsistence in performance predictions. Second, hardware latency predictions for synchronous circuits are accurate when system-level cycle-accurate simulations align with gate-level circuit cycles. However, asynchronous circuits do not have global

†Yuan Hua and Jian Zhang contributed equally to this work.
*Hong Chen is the corresponding author.

clocks or cycles. Thus, if system-level simulators for asynchronous circuits are required to predict fine-grained timing down to nanoseconds or even picoseconds, we must assume a relationship between simulation runtime and hardware latency. For instance, the TrueAsync simulator builds a fully asynchronous actor system to model the clock-free asynchronous circuits, where one second of simulation runtime corresponds to 1 μs of circuit latency. In fact, the mismatches between system-level models and gate-level circuits are reasonable. The mean absolute percentage error (MAPE) and root mean square error (RMSE) of the hardware latency system-level prediction results by the TrueAsync are 50.06% and 7113, respectively, compared to the gate-level simulation results.

Generally, hardware latency and power consumption are the two key metrics in performance predictions. To predict the hardware performance at system-level, many studies focus on using graph neural network (GNN)-based approaches to avoid the execution of time-consuming and expensive simulations, analyses or experiments [9]. Examples include a GNN-based timing engine to predict arrival time and slack at timing endpoints [10], high-level synthesis (HLS) performance prediction methods using GNNs by representing C/C++ programs as graphs [11], and a preplacement prediction flow with estimators on both net length and timing by customized GNN [12]. However, previous performance prediction methods for asynchronous circuits still rely on analytical models and have not applied promising machine learning techniques [13, 14, 15].

In this paper, we propose a novel performance prediction framework for asynchronous neuromorphic hardware, which transforms the intermediate representation (IR) of the TrueAsync simulator into asynchronous neuromorphic graphs and predicts hardware performance including latency and power consumption. Our main contributions are as follows:

- **Transforming.** We introduce a method to analyze event streams and define asynchronous neuromorphic graphs as IRs. By transforming the event streams into graphs, we bridge system-level simulation and graph-level representations, enabling GNN training and inference for hardware latency predictions.
- **Benchmarking.** We design register-transfer level (RTL) and netlist circuits for asynchronous neuromorphic hardware at various scales and manufacturing process nodes. These gate-level netlists are simulated using diverse compiled input data from different applications. By pairing system-level event streams from the TrueAsync simulator with gate-level hardware performance results, we build a standard benchmark suite with more than one million samples, including hardware latency and power consumption prediction tasks.
- **Predicting.** We put forward the GNN-based *ANGraph-L* and the residual network (ResNet)-based *ANGraph-P* models for predicting hardware latency and power consumption, respectively. Compared to the latency prediction results from the TrueAsync simulator, we improve the R^2 score by 0.69 and reduce RMSE by 78%. Furthermore, we evaluate these two models on additional datasets without extra training across different scales, input traffic patterns, and process nodes. *ANGraph-L* achieves an average R^2 score of 0.96, while *ANGraph-P* achieves an average R^2 score of 0.98 and a MAPE of 0.88%.
- **Opensource.** The benchmarks and models are available at https://github.com/HuaGuaiGuai/ANGraph.

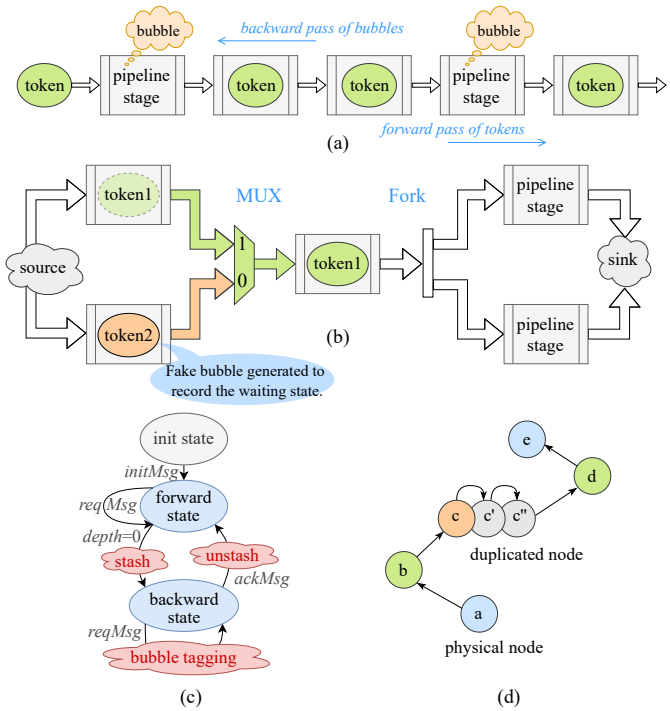


Fig. 2. (a) In asynchronous pipelines, data tokens move forward while bubbles move backward, resulting in forward and backward pass asynchronous events. (b) The fake bubble event occurs at the MUX point of asynchronous pipelines when data tokens are congested. (c) The finite-state machine of asynchronous controllers we designed is able to simulate fake bubble events. (d) Duplicated nodes are copies of a physical node.

II. Transforming Event Streams into Graphs

To enable GNN-based performance predictions, we reconstruct essential information during the system-level simulation of asynchronous neuromorphic hardware from the perspective of asynchronous events. We define and generate asynchronous neuromorphic graphs with encoded features, as depicted in Figure 1.

A. Event Stream Extraction

As we know, events are crucial in analyzing event-driven asynchronous circuits. The TrueAsync simulator builds a fully asynchronous, high-concurrency actor system to model the asynchronous circuits, which recognizes two types of events during its finite-state machine (FSM)-based modeling and simulation, including forward and backward pass events. These two events describe the flow of data tokens, with the backward pass event occurring in a high congestion state of circuits and

having a higher latency than the forward pass event, as shown in Figure 2(a).

However, another asynchronous state of data tokens, the waiting state, must be considered when transforming dynamic real-time simulations into static event streams. To address this, we introduce a new asynchronous event, the fake bubble event, to represent the waiting states of data tokens in asynchronous pipelines, as illustrated in Figure 2(b). Assuming all pipeline stages are initially empty, each pipeline stage generates the forward pass and the backward pass events for data tokens that pass through it. Tokens in congestion are held in place, where the pipeline stage generates a fake bubble event for each congested token. These three asynchronous events describe all the system-level behaviors of asynchronous pipeline stages with respect to data tokens, including forward pass event, backward pass event, and fake bubble event.

Different from the TrueAsync simulator, we design FSM-based asynchronous controllers that incorporate the fake bubble feature, as shown in Figure 2(c). At the end of a simulation, a JSON file is generated containing all the asynchronous events occurred during the simulation, preparing for transformation into graphs.

B. Asynchronous Neuromorphic Graph

Following the movement of a data token in the asynchronous pipeline, we define an asynchronous neuromorphic graph G, consisting of a set of nodes V representing the circuit modules that a data token passes through, and a set of edges E describing the direction of the node sequence. The structure of G is denoted by an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$, where $A_{uv} = 1$ if there is an edge $(u, v) \in E$ exists and 0 otherwise. In asynchronous neuromorphic graphs, each data token generates an independent graph, traveling from source to sink and passing through several circuit modules as graph nodes. Therefore, the asynchronous neuromorphic graph for a data token is an unweighted directed acyclic graph (DAG) with a single start point, as shown in Figure 2(d).

We encode the node features based on the inherent features of circuit modules (e.g., delays) and the event streams of the data token. Each node $v \in V$ has a feature vector $X_v \in \mathbb{R}^F$, where F denotes the feature dimension. All feature vectors are collected in the matrix:

$$X = [X_1^\intercal, X_2^\intercal, \ldots, X_{|V|}^\intercal]^\intercal \tag{1}$$

where $\intercal$ is the transpose operation. Since event stream lengths vary among different circuit modules, we set the dimension F to 100 to standardize of size of matrix X. For the event stream whose length exceeds dimension F, we duplicate the graph node to encode additional events as node features, as illustrated in Figure 2(d). The duplicated nodes help avoid a matrix X that has too many zero elements or is excessively large due to unbalanced and varying event stream lengths. Meanwhile, in order to reduce simulation runtime, we limit the maximum event stream length to $10k$ by splitting the input data tokens from the source into several subtasks and predicting the overall performance by summing them up.

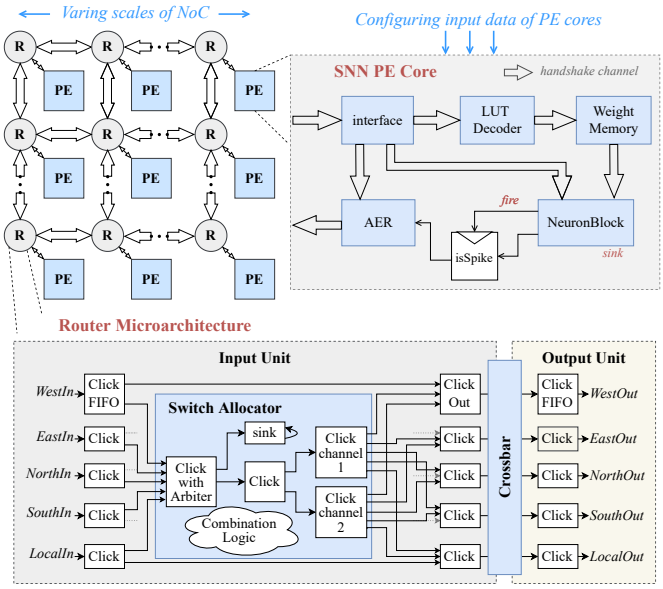


Fig. 3. The microarchitecture of fully asynchronous neuromorphic hardware.

III. Benchmarking for Asynchronous Neuromorphic Hardware

We establish a standard benchmark suite for performance prediction by ANGraph in asynchronous neuromorphic hardware across various network scales, traffic patterns of input data, and manufacturing process nodes. The overall workflow of benchmarking is depicted in Figure 1.

A. Asynchronous Neuromorphic Hardware

The fully asynchronous neuromorphic hardware used in this work is depicted in Figure 3, inspired by the spiking neural network (SNN) accelerator in [4]. The mesh-based network-on-chip (NoC) architecture consists of multiple routers connected to SNN processing element (PE) cores. Thses cores can be configured as different applications by adjusting the LUT decoder, weight memory, and address-event representation (AER) modules. For example, to create the training dataset named L/P-MNIST-4×4-T180, we design fully asynchronous neuromorphic hardware with 4×4 SNN inference cores and mesh-based NoC routers, configured as a two-layer fully connected network for MNIST classification [16], where *L* for latency prediction dataset and *P* for power prediction dataset. Meanwhile, configure the hardware as another five-layer convolutional network for MNIST classification to extend the training dataset.

B. Workflow with EDA Tools

Aimed at providing comprehensive benchmarks for performance predictions, we design asynchronous neuromorphic hardware from RTL to netlists using design tools reported in [17] and the Synopsys Design Compiler (DC) . Through gate-level simulations with Verilog compile simulator (VCS) and Verdi tools, we obtain detailed analysis results of our

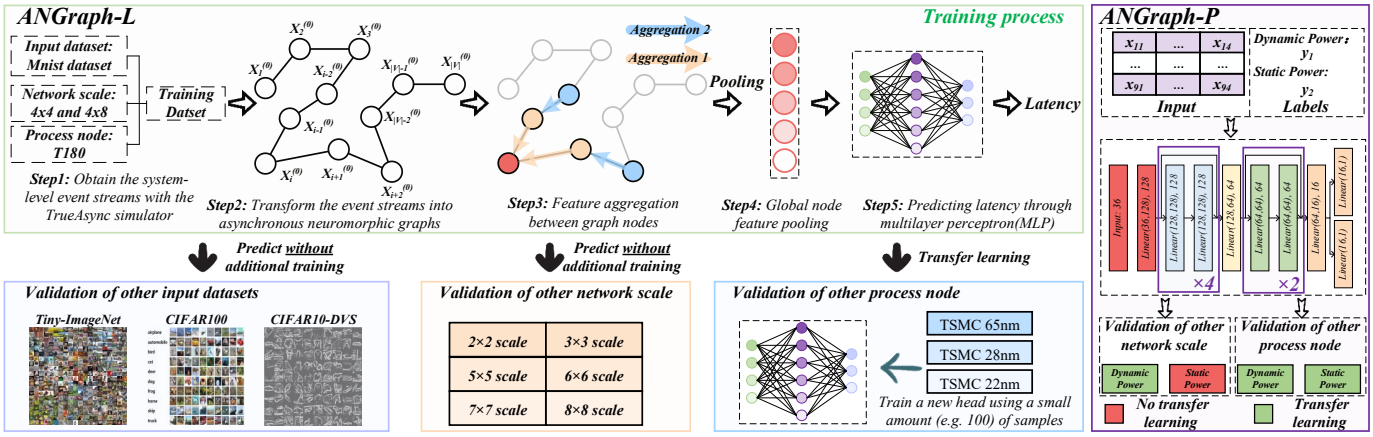


Fig. 4. The structure diagram of the ANGraph framework.

netlists, including the latency of each data token (in picoseconds) and static and dynamic power consumption.

After obtaining the system-level event streams from the TrueAsync simulator, we transform them into asynchronous neuromorphic graphs for the latency prediction task using the transformation method discussed earlier. For the power prediction task, we create activity matrices by recording asynchronous events for each module during the system-level simulation. By pairing gate-level analysis results as labels with system-level graphs and activity matrices as samples, we build nineteen datasets with two types of tasks: latency prediction as shown in Table I and power prediction as shown in Table II in Section V.

C. Extended Datasets

In addition to using a specific combination of circuits, input traffic patterns, and manufacturing process nodes to train our GNN model, we test the model on additional datasets with varying circuit scales, input traffic patterns, and process nodes. To create datasets with different network scales of asynchronous neuromorphic hardware, we adjust the number of PE cores and NoC routers, rearranging the mesh array from 2×2 to 8×8. Since the circuit testbenches compiled from applications for the same task (e.g., MNIST classification) may have certain input traffic patterns, we generate additional datasets using various asynchronous neuromorphic hardware configured as different applications and input datasets, such as CIFAR10-DVS [18], CIFAR100 [19], and Tiny-ImageNet [20]. All the datasets are generated with hardware netlists synthesized with the TSMC 180nm process, as well as TSMC 65nm, 28nm, and 22nm, where latency and power consumption vary significantly across different process nodes.

IV. Predicting with ANGraph Framework

In this section, we describe the structure of the ANGraph models used in this paper. The overall structure of the ANGraph framework is shown in Figure 4. We propose a GNN-based *ANGraph-L* model for latency predictions to learn from asynchronous neuromorphic graphs. In addition, since the power consumption prediction task deals with activity matrices instead of graphs, we propose a ResNet-based *ANGraph-P* for power consumption predictions.

A. ANGraph-L for Latency Prediction Task

ANGraph-L leverages a general inductive framework that efficiently generates node features for previously unseen data, relying solely on the feature information of nodes, which are the system-level event streams extracted from the TrueAsync simulator [21]. This approach demonstrates robust generalization capabilities, particularly when dealing with different input traffic patterns and varying scales of asynchronous neuromorphic hardware. For the input X in Equation (1), *ANGraph-L* follows the steps outlined below to predict latency.

Feature Aggregation. The *ANGraph-L* uses mean graph convolution to integrate the feature information of a node and its neighboring nodes, generating an updated feature for the node X_i. It can be represented as:

$$X_i^{(k+1)} = \sigma(W_1 X_i^{(k)} + W_2 \text{mean}_{j \in \mathcal{N}(i)} X_j^{(k)}) \tag{2}$$

where σ is activation function and W_1, W_2 are learnable parameters. $X_i^{(k)}$ represents the output of the i-th node at the k-th layer of graph convolution. $\mathcal{N}(i)$ stands for the set of neighboring nodes of the i-th node. After stacking k layers of graph convolution, *ANGraph-L* performs feature aggregation on nodes, summing up the node features from k graph convolutions to obtain the aggregated node features, expressed as:

$$Y_j = \sum_{i=1}^{k} X_j^{(i)} \tag{3}$$

where $Y = [Y_1, Y_2, \ldots, Y_{|V|}]$ represents the aggregated node features.

By stacking graph convolutional layers, the receptive field of the model can be enlarged. With each additional layer, node features aggregate information from further neighborhoods, helping the model capture long-range dependencies. Through

hierarchical feature aggregation, multi-level node features in the graph can be integrated, which allows node representations to include not only local information but also higher-order neighbor information. This provides the model a more comprehensive understanding of the relationships between nodes. In *ANGraph-L*, the number of stacked graph convolutional layers $k = 5$.

Global Pooling and Prediction. After feature aggregation, each node obtains its corresponding aggregated node feature Y_i. *ANGraph-L* integrates the aggregated features Y_i of each node into a graph-level feature $G \in \mathbb{R}^F$ through global sum pooling, which can be represented as:

$$G = \sum_{i=1}^{|V|} Y_i \tag{4}$$

The next step is to learn the final predictor using the multilayer perceptron (MLP). For each graph-level feature G, we can obtain its corresponding latency through the predictor.

As we adopt stacked graph convolution and feature aggregation learning methods, *ANGraph-L* is able to understand relationships among event streams. This approach enables *ANGraph-L* to effectively predict latency performance across datasets of varying scales and process nodes with diverse traffic patterns, demonstrating great generalization capabilities. Moreover, it is unnecessary to retrain the model with a large dataset for different process nodes. Structure-based transfer learning is employed to learn the fundamental features of new process nodes. During the transfer learning process, only a small number of samples are used to train a new predictor, achieving high performance while preserving the learned relationships among event streams.

B. ANGraph-P for Power Prediction Task

In order to predict both the dynamic and static power consumption of the circuit at the same time, we implement a ResNet structure with dual output heads, as shown in Figure 4. For the input $X \in \mathbb{R}^{L=36}$, when it is fed into *ANGraph-P* for power consumption prediction, $Y = [y_1, y_2]$ will be obtained, where y_1 represents the dynamic power consumption and y_2 represents the static power consumption. The total power consumption is equal to the sum of static and dynamic power consumption.

V. Experimental Results

A. Experimental Setup

We implement the GNN model with Pytorch Geometric [22]. During the training process, we use the Adam optimizer with a learning rate of 0.001, dropout set to 0.5, and the loss function is HuberLoss with delta=5. In the transfer learning process, *ANGraph-L* only needs to learn new predictors, while the rest of the parameters remain unchanged. *ANGraph-P* only requires relearning the dynamic power output head and the static power output head.

We compare the prediction results with the TrueAsync simulator by implementing the same hardware models as circuits and simulating the same input data. Comparison results are shown in Table I, with improvements listed in the parentheses. For instance, 0.99 (↑0.16) of R^2 score indicates that *ANGraph-L* achieves 0.99, improving by 0.16 compared to TrueAsync, and an RMSE of 1530 (↓78%) indicates that *ANGraph-L* achieves 1530, reducing by 78% compared to TrueAsync.

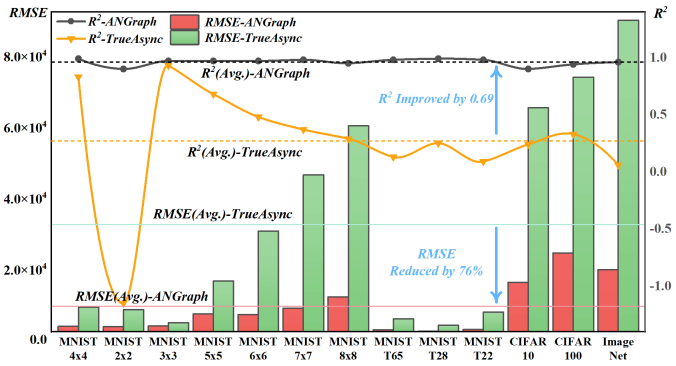


Fig. 5. The latency prediction results with ANGraph-L comparing to those from the TrueAsync simulator.

B. Latency Prediction Results with ANGraph-L

We conduct experiments to predict hardware latency on various datasets with *ANGraph-L* and compare the results to those from the TrueAsync simulator, as shown in Figure 5

On the training dataset, we randomly split the L-MNIST-4×4-T180 dataset into 80% train, 10% validation and 10% test for the latency prediction task. As shown in Table I, the *ANGraph-L* model achieves an R^2 score of 0.99. Compared to the prediction results from the TrueAsync simulator, the ANGraph method improves $R2$ score by 0.16 and reduces RMSE by 78.49%.

On the scale-testing datasets, we directly test datasets of varying scales from 2×2 to 8×8 without additional training or tuning. Results in Table I show that the *ANGraph-L model* achieves R^2 scores of over 0.9, with an average of 0.96, demonstrating that the model can accurately predict the latencies of asynchronous neuromorphic hardware at various scales.

On the traffic-testing datasets, to show that the *ANGraph-L* model adapts to different traffic patterns of input data, we test datasets compiled from various classification tasks, such as CIFAR10-DVS, CIFAR100, and Tiny-ImageNet. Results in Table I show that the *ANGraph-L* model achieves R^2 scores of over 0.9, with an average of 0.93 on these three datasets, indicating that the model is not biased toward specific input traffic patterns, enabling the prediction of the latency performance for asynchronous neuromorphic hardware with various applications.

On the process-testing datasets, we train new predictors for the *ANGraph-L* model with a few new samples to learn the basic features of different process nodes, as shown in Table I. After transfer learning, the tested model achieves an average R^2 score of 0.98 across these three datasets, demonstrating that *ANGraph-L* is able to predict latency of hardware with various

TABLE I
LATENCY PREDICTION RESULTS WITH ANGRAPH-L COMPARING TO THOSE FROM THE TRUEASYNC SIMULATOR

Dataset Name*	Usage†	R^2 Score	RMSE
L-MNIST-4x4-T180	train	0.99 (↑**0.16**)	1530 (↓**78%**)
L-MNIST-2x2-T180	test-s	0.90 (↑**2.05**)	1429 (↓**77%**)
L-MNIST-3x3-T180	test-s	0.97 (↑**0.04**)	1632 (↓**36%**)
L-MNIST-5x5-T180	test-s	0.97 (↑**0.29**)	5011 (↓**65%**)
L-MNIST-6x6-T180	test-s	0.97 (↑**0.49**)	4818 (↓**83%**)
L-MNIST-7x7-T180	test-s	0.98 (↑**0.61**)	6619 (↓**85%**)
L-MNIST-8x8-T180	test-s	0.95 (↑**0.66**)	9846 (↓**83%**)
L-MNIST-4x4-T65	test-p	0.98 (↑**0.85**)	551 (↓**85%**)
L-MNIST-4x4-T28	test-p	0.99 (↑**0.74**)	201 (↓**89%**)
L-MNIST-4x4-T22	test-p	0.98 (↑**0.89**)	661 (↓**88%**)
L-CIFAR10DVS-4x8-T180	test-t	0.90 (↑**0.66**)	13864 (↓**78%**)
L-CIFAR100-4x8-T180	test-t	0.94 (↑**0.61**)	22195 (↓**69%**)
L-TinyImageNet-4x8-T180	test-t	0.96 (↑**0.90**)	17519 (↓**80%**)
Average Improvement		↑ **0.69**	↓ **76%**

* Naming rule of datasets: [latency or power prediction task]-[input data source]-[network scale]-[process node in TSMC].

† **test-s**: test dataset for different scales. **test-p**: test dataset for different process nodes. **test-t**: test dataset for different traffic patterns of input data.

process nodes after transfer learning using a small number of new samples.

Comparing to the TrueAsync predictions, the *ANGraph-L* model improves the R^2 score by 0.69 and reduces the RMSE by 76% on average across all datasets of the latency prediction task. Higher R^2 scores and lower RMSE values indicate a better model fit for these regression tasks.

C. Power Consumption Prediction Results with ANGraph-P

For power consumption predictions, we use *ANGraph-P* to train and test on various datasets. The power consumption predictions cannot be compared since the state-of-the-art (SOTA) TrueAsync simulator is unable to predict it.

On the training dataset, we randomly split the P-MNIST-4×4-T180 dataset into 80% train, 10% validation and 10% test for power consumption prediction task. As shown in Table II, the *ANGraph-P* model achieves an R^2 score of 0.99 and a MAPE of 0.21%. The ResNet architecture of *ANGraph-P* is well-suited to handling matrix inputs such as activity matrices, leading to the high R^2 score and low MAPE.

On the scale-testing datasets, we test the *ANGraph-P* model on the P-MNIST-2×2-T180 and P-MNIST-8×8-T180 datasets without additional training. Results in Table II show that the *ANGraph-P* model achieves R^2 scores of 0.99 and 0.98, with MAPE values of 0.31% and 0.98%, respectively, indicating that it is capable of predicting the power consumption of hardware at various scales.

TABLE II
POWER PREDICTION RESULTS WITH ANGRAPH-P

Dataset Name	Usage	R^2 Score	MAPE
P-MNIST-4x4-T180	train	0.99	0.21 %
P-MNIST-2x2-T180	test-s	0.99	0.31 %
P-MNIST-8x8-T180	test-s	0.98	0.98 %
P-MNIST-4x4-T65	test-p	0.98	1.00 %
P-MNIST-4x4-T28	test-p	0.98	1.54 %
P-MNIST-4x4-T22	test-p	0.98	1.21 %
Average Result		**0.98**	**0.88** %

On the process-testing datasets, similarly, we train new predictors for the *ANGraph-P* model with a few new samples, as shown in Table II. After transfer learning, the tested model achieves an R^2 score of 0.98 and a MAPE of 1.25% on average across these three datasets. As a result, we can transfer train the *ANGraph-P* model to predict the power consumption of hardware with different process nodes.

VI. CONCLUSION

The system-level performance prediction methods for asynchronous circuits have been studied for decades. However, these methods still rely on analytical models and face challenges due to the complexity of asynchronous systems. In this paper, ANGraph presents a promising GNN-based approach to predict the latency performance of asynchronous neuromorphic hardware, bridging the gap between system-level predictions and gate-level simulations. We put forward a transformation method that extracts event streams from system-level simulations and converts them into graphs. Additionally, we provide a standard benchmark suite for predicting the performance of asynchronous neuromorphic hardware across various network scales, process nodes, and input data. Using these benchmarks, we introduce the GNN-based *ANGraph-L* and ResNet-based *ANGraph-P* model to predict latency and power consumption of asynchronous neuromorphic hardware, respectively. For the latency prediction task, we achieve an average R^2 score of 0.96 on all datasets. Compared to the direct prediction results from the SOTA simulator, the *ANGraph-L* model improves R^2 score by 0.69 and reduces RMSE by 76%. For the power consumption prediction task, we achieve an R^2 score of 0.98 and a MAPE of 1.21% on average across all datasets. Moreover, with the ANGraph framework, we can accurately predict the performance of asynchronous neuromorphic hardware at different scales, process nodes, and traffic patterns of input data, demonstrating its robust generalization ability.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (NSFC) (Grant No. 62334014).

REFERENCES

[1] Garrick Orchard et al. "Efficient neuromorphic signal processing with loihi 2". In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2021, pp. 254–259.

[2] Jilin Zhang et al. "ANP-I: A 28-nm 1.5-pJ/SOP Asynchronous Spiking Neural Network Processor Enabling Sub-0.1-μJ/Sample On-Chip Learning for Edge-AI Applications". In: *IEEE Journal of Solid-State Circuits* (2024).

[3] Dexuan Huo et al. "ANP-G: A 28nm 1.04 pJ/SOP Sub-mm2 Spiking and Back-propagation Hybrid Neural Network Asynchronous Olfactory Processor Enabling Few-shot Class-incremental On-chip Learning". In: *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE. 2023, pp. 1–2.

[4] Jilin Zhang et al. "A 28nm configurable asynchronous SNN accelerator with energy-efficient learning". In: *2021 27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE. 2021, pp. 34–39.

[5] Ying Liu et al. "An 82nW 0.53 pJ/SOP clock-free spiking neural network with 40μs latency for AIoT wake-up functions using ultimate-event-driven bionic architecture and computing-in-memory technique". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE. 2022, pp. 372–374.

[6] Dexuan Huo et al. "A 28nm Energy-efficient Asynchronous SNN Accelerator with On-chip Learning for Gas Recognition". In: *2023 28th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE. 2023, pp. 42–47.

[7] Jian Zhang et al. "ANAS: Asynchronous Neuromorphic Hardware Architecture Search Based on a System-Level Simulator". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2023, pp. 1–6.

[8] Jian Zhang et al. "ANCoEF: Asynchronous Neuromorphic Algorithm/Hardware Co-Exploration Framework with a Fully Asynchronous Simulator". In: *arXiv preprint arXiv:2411.06059* (2024).

[9] Eli Chien et al. "Opportunities and challenges of graph neural networks in electrical engineering". In: *Nature Reviews Electrical Engineering* 1.8 (2024), pp. 529–546.

[10] Zizheng Guo et al. "A timing engine inspired graph neural network model for pre-routing slack prediction". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 1207–1212.

[11] Nan Wu et al. "High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 49–54.

[12] Zhiyao Xie et al. "Preplacement net length and timing estimation by customized graph neural network". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 4667–4680.

[13] PL Penzes and Alain J Martin. "An energy estimation method for asynchronous circuits with application to an asynchronous microprocessor". In: *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. IEEE. 2002, pp. 640–647.

[14] M Salehi et al. "High-level energy estimation of template-based QDI asynchronous circuits based on transition counting". In: *Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004.* IEEE. 2004, pp. 489–492.

[15] Maryem Benyoussef, Claude Thibeault, and Yvon Savaria. "A prediction model for implementing DVS in single-rail bundled-data handshake-Free asynchronous circuits". In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2019, pp. 1–5.

[16] Li Deng. "The mnist database of handwritten digit images for machine learning research [best of the web]". In: *IEEE signal processing magazine* 29.6 (2012), pp. 141–142.

[17] Jilin Zhang et al. "Designing Self-timed Asynchronous Circuits with Chisel". In: *2023 28th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE. 2023, pp. 27–33.

[18] Hongmin Li et al. "Cifar10-dvs: an event-stream dataset for object classification". In: *Frontiers in neuroscience* 11 (2017), p. 244131.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[20] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. "A downsampled variant of imagenet as an alternative to the cifar datasets". In: *arXiv preprint arXiv:1707.08819* (2017).

[21] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[22] Matthias Fey and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric". In: *arXiv preprint arXiv:1903.02428* (2019).

PIMoE: Towards Efficient MoE Transformer Deployment on NPU-PIM System through Throttle-Aware Task Offloading

Lizhou Wu*†, Haozhe Zhu*†, Siqi He†, Xuanda Lin†, Xiaoyang Zeng†, and Chixiao Chen†‡
† *State Key Laboratory of Integrated Chips and Systems (SKLICS), Fudan University, Shanghai, China*
‡ *Shaoxin Laboratory, Shaoxing, Zhejiang, China*
E-mail: zhuhz@fudan.edu.cn

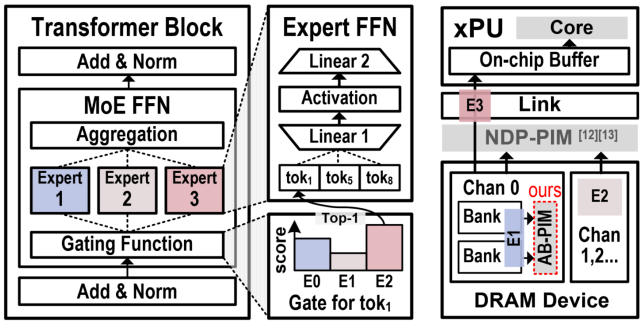
***Abstract*—Mixture-of-experts (MoE) technique holds significant promise for scaling up Transformer models. However, the data transfer overhead and imbalanced workload hinder efficient deployment. This work presents PIMoE, a heterogeneous system combining processing-in-memory (PIM) and neural-processing-unit (NPU) to facilitate efficient MoE Transformer inference. We propose a throttle-aware task offloading method that addresses workload imbalance between NPU and PIM, achieving optimal task distribution. Furthermore, we design a near-memory-controller data condenser to address the mismatch of sparse data layout between NPU and PIM, enhancing data transfer efficiency. Experimental results demonstrate that PIMoE achieves 4.5× speedup and 13.7× greater energy efficiency compared to the A100, and 1.4× speedup over a state-of-the-art MoE platform.**

I. Introduction

Transformer-based large language models (LLMs) have performed remarkably across various natural language processing tasks. This exceptional performance is largely due to the unprecedented growth in model sizes over recent years [1], [2]. However, increasing model capacity results in higher computational costs and memory requirements, making training and deployment increasingly challenging.

The Mixture-of-Experts (MoE) technique has emerged as a promising approach to scaling model sizes while minimizing proportional increases in computational overhead [3]–[7]. In MoE-based Transformers, the sparse gating mechanism allows only a subset of experts to process an input token, resulting in lower computational expenses compared to non-MoE models with equivalent parameter counts. Consequently, MoE Transformer has been practically applied to LLM productions [2].

Although MoE Transformers significantly increase model capacity without greatly raising computation costs, deploying these models for inference faces two main challenges: data transfer overhead and workload imbalance. Firstly, the limited on-chip buffer capacity causes frequent eviction and reloading of expert parameters, leading to significant data transfer overhead. This issue is worse for experts that receive few tokens, resulting in low parameter reuse and memory-bound tasks that increase inefficiencies and latency. Secondly,



(a) (b)

Fig. 1. (a) Diagram of Transformer block with MoE FFN layer. (b) Multi-levels of expert offloading in a heterogeneous NPU-PIM system.

the dynamic sparse allocation in MoE Transformers causes workload imbalance, with some experts overburdened while others are underutilized. This is particularly problematic in heterogeneous multi-core systems, where imbalanced token distribution leads to latency bottlenecks in the critical path.

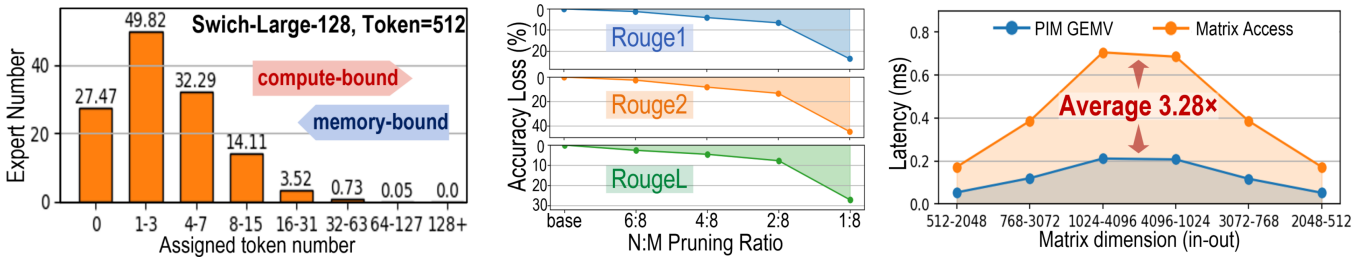
Many works have integrated xPU (NPU/GPU) with PIM to address data transfer and memory overhead. IANUS [8], AttAcc [9], NeuPIMs [10] propose offloading memory-bound matrix-vector multiplications to PIM devices, leveraging the high bandwidth. Cambricon-LLM [11] proposes a NPU-NAND hybrid architecture to support tremendous storage demand. While designed for general transformer models, these works do not fully concern the specific expert imbalance in the MoE context. MoNDE [12] proposes a near-data-processing (NDP) approach for MoE Transformers. It trade-off parameter movement with activation movement based on the workload characteristics of different experts. Duplex [13] proposes a new HBM device with increased TSV amount to enlarge the NDP's bandwidth for better computing throughput.

In this work, we present PIMoE, a co-optimized NPU-PIM system for efficient MoE Transformer deployment. In summary, this paper makes the following contributions:

- We fully leverages the all-bank parallelism of commercial PIM products, addressing the memory-bounded expert computation associated with MoE FFN.
- We propose a throttle-aware task offloading method designed to optimize the distribution of experts be-

* Lizhou Wu and Haozhe Zhu are co-first authors. Haozhe Zhu is the corresponding author.

This work was supported by the National Natural Science Foundation of China (NSFC) under Grants 62304047 and 62488101.



(a) (b) (c)

Fig. 2. Preliminary Experiments (a) Histogram of experts by allocated token count. (b) Accuracy degradation across various N:M pruning ratios. (c) Latency Comparison between PIM-based GEMV and matrix data loading.

tween PIM and NPU, mitigating performance bottlenecks caused by imbalanced workloads.

- We propose a near-memory-controller data condenser that addresses the mismatch between DRAM and NPU sparse data layouts. Condensing transfer data also alleviates NoC congestion and enhances data transfer efficiency.
- PIMoE achieves a 4.5× speedup and 13.7× energy efficiency compared to the A100. It also delivers a 1.4× speedup over a state-of-the-art NDP MoE platform.

II. Background and Motivation

A. MoE-based Transformers

The Mixture-of-Experts (MoE) technique [14] partitions a complex task into separated sub-tasks, enhancing the model's fitting ability. Sparse MoE [3] further introduces a gating network that selectively activates a subset of experts for each workload. As for Transformer models, the sparse MoE has been applied to the feed-forward layers (FFNs) [6], [7] to relax computation as the model increasingly scales up. Fig. 1 illustrates a typical MoE-based FFN (dubbed MoE FFN) within a Transformer block. The MoE FFN replicates the original FFN into N experts, each comprising two linear layers with an activation layer in between. At the block's entrance, the gating function infers the probability of each input token being allocated to the N experts. Then the token is routed to the top-K experts for inference. The results from each expert are weighted according to their corresponding probabilities and aggregated at the output.

The token workloads across FFN experts feature uneven distribution. We profile the switch-large-128 model [6] consisting of 128 experts and operates with 512 tokens. Fig. 2(a) shows the histogram of experts by allocated token count. Nearly half of the experts are allocated fewer tokens (i.e., 1-3), featuring memory-bound characteristics with limited weight reuse. In contrast, those experts with heavier token allocations are compute-bound, dominated by matrix multiplication.

Besides the sparse expert selection, the expert can further benefit from weight pruning, particularly the N:M method that selects N significant weights from each M-element vector. As indicated by Fig. 2(b), this approach reduces on-chip buffer overhead without severe degradation in performance. Furthermore, this weight sparsity can be effectively leveraged to accelerate computation by skipping the zero operation.

B. Commercial PIM Platforms

Processing-in-memory (PIM) covers a variety of techniques that deploy computing resources near or within DRAM devices. Among them, the all-bank (AB) paradigm, which deploys processing units besides the DRAM banks, has gained favor among several semiconductor manufacturers (PIM-HBM [15], AiM [16]). AB-PIM fully utilizes the internal bus bandwidth without intrusive change to the memory cell, striking a balance between throughput and storage density.

Commercial AB-PIM (e.g., PIM-HBM, AiM) prioritizes compatibility with existing protocols: High-level computing commands are converted into low-level JEDEC commands (write/read) to initiate PIM execution, typically managed by the memory controller. Therefore, both the PIM computation and the normal memory access occupy the memory controller, incrementally increasing the latency.

To characterize the PIM's performance, we profile the PIM-HBM with the corresponding PIMSimulator [17]. As shown in Fig. 2(c), the latency of per token matrix-vector multiplication (GEMV) linearly scales with the size of the weight parameters. Moreover, the throughput ratio between PIM computation and regular memory access is also nearly in proportion. Specifically, an N-to-M GEMV achieves 3.28× higher throughput than loading an $N \times M$ weight parameter. This falls short of the ideal 8× bank-level parallelism in PIM-HBM, due to PIM-specific overheads like activation writes and barrier insertions. This result indicates that PIM is well-suited for 'cold' experts with fewer tokens. For 'hot' experts with many tokens, the PIM's efficiency diminishes.

C. Motivation

To efficiently accommodate experts with diverse workloads, we propose an NPU-PIM hybrid system that leverages the PIM's weight-stationary feature and bank-level parallelism, as well as the NPU's high computational throughput. To prevent system performance from being throttled by critical experts on specific components, we propose a throttle-aware task offloading method that optimizes task allocation based on a latency model of the heterogeneous components. Furthermore, even though the NPU can effectively leverage sparsity, current PIMs do not well support sparse FFN inference, where the sparse weights have to be stored in a structured pattern. To bridge the gap of sparse data layout between NPU and PIM, and

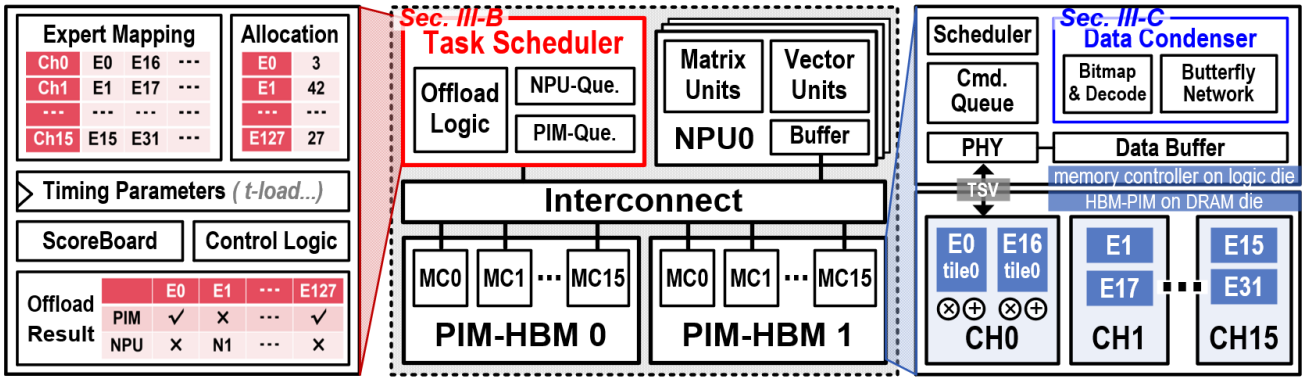


Fig. 3. The overall architecture of PIMoE. The task scheduler is explained in section III-B. The data condense is elaborated in section III-C.

enhance the bandwidth utilization for preventing congestion, we propose a near-memory-controller data condenser.

III. PIMoE

This section introduces the architecture of PIMoE, the throttle-aware task offloading method that optimizes the heterogeneous system, and the near-memory-controller data condenser that improves transfer efficiency.

A. PIMoE Architecture

Fig. 3 illustrates the overall architecture of the proposed PIMoE, the components of which are introduced below.

NPU: The NPU consists of matrix units (MUs), vector units (VUs), and scratchpad buffers. The MUs perform matrix multiplications involved in the gating function and the two linear layers within FFN experts. The MU is designed as a systolic array that supports sparse acceleration. The weights are pruned using the N:M scheme and are densely stored in the buffer, along with the corresponding bitmap. During computation, the activations pass through a gather unit that decodes the bitmap to select N non-zero elements from M inputs [18], while the multiply-accumulate (MAC) unit calculates the inner product of the condensed vectors. The VUs handle the element-wise operations including bias addition, residual addition, activation function, and weighted sum of multiple experts' results.

PIM: PIMoE adopts Sumsang's commercial product, PIM-HBM [15], which is based on HBM2 DRAM that consists of 16 channels, offering a total of 1024 I/O pins and a maximum bandwidth of 307GB/s. Each channel contains 16 banks, with PIM-HBM deploying 8 processing units among these banks. Due to the limited routing resources in DRAM devices, it is preferable to localize the computation with the channel to minimize the global communication and synchronization overhead. Therefore, each expert's parameters are consecutively mapped within the channel's address range and only tiled across different DRAM devices to fit the per-channel storage limit. To support straightforward scaling of the number of experts, the experts are interleaved across channels, with the E_{th} expert being mapped to the channel $E_{th}\%N_{chan}$. Each DRAM channel includes a memory controller that supports both normal memory access and PIM-specific commands. The memory controller is equipped with a data condenser, as discussed in Section III-C, to efficiently transfer structured weight parameters sparsely stored in DRAM to the sparsity-optimized NPU that requires a dense weight format.

Task Scheduler: The task scheduler dispatches the experts across the NPUs and PIMs based on the runtime token distribution. It takes the results of the gating function as input and decides each expert's computing medium according to the throttle-aware offloading method described in Section III-B. For the NPU-offloaded expert, it requests the DMA to load the parameters from DRAM to the on-chip buffer and triggers the matrix multiplication once the data is prepared. For the PIM-offloaded task, it issues the PIM commands to the target memory controller according to the expert address and loop for each assigned token. To avoid inefficiency caused by frequent mode switching, the task scheduler inserts barriers between two types of tasks.

Network-on-chip: The NoC manages data transfer between the DRAM and NPU. It employs a 2D mesh topology with NPUs and memory controllers as nodes. For PIM-offloaded tasks, the NPU initiates the transfer, sending activation data to the target DRAM partition. For NPU-offloaded tasks, expert parameters are loaded from DRAM to the target NPU.

B. Throttle-Aware Task Offloading Method

The heterogeneous platform introduces more factors regarding workload imbalance, which can be categorized as PIM-throttle and NPU-throttle within the context of PIMoE.

PIM throttle: For a batch of input tokens offloaded to PIM, each DRAM channel handles those allocated to the subset of experts within its address range. The memory controller operates in parallel, with the critical channel managing the largest number of tokens. Due to the requirement for output synchronization across experts, the critical channel constrained the execution latency of the PIM-offloaded MoE FFN.

NPU throttle: For those experts offloaded to NPU, their parameters are loaded from the DRAM to the on-chip buffer. As the number of loaded experts increases, the latency of the critical NPU may exceed that of the PIM latency, causing the latter to wait for the completion of the NPU.

Existing work, such as IANUS [8] and MoNDE [12], employs a straightforward offloading method, not fully accounting for potential runtime throttle. IANUS offloads a task

Algorithm 1: throttle-aware offloading method

Input: $M_{CH\text{-}EP}$, $M_{EP\text{-}TK}$, $threshold$
Output: $M_{EP\text{-}PIM}$

$M_{EP\text{-}PIM} = [EP]$;
Initialize($[t_{CH}]$, $[t_{NPU}]$, $M_{EP\text{-}TK}$);
// Until NPU becomes the throttle
while $max([t_{CH}]) < max([t_{NPU}])$ **do**
  $CH_{thro} = argmax([t_{CH}])$;
  $EP_{candi} = M_{CH\text{-}EP}[CH_{thro}] \cap M_{EP\text{-}PIM}$;
  if $EP_{candi} \neq \emptyset$ **then**
    $N_{EP\text{-}TK} = M_{EP\text{-}TK}[EP_{candi}]$;
    $EP_{thro} = argmax(N_{EP\text{-}TK})$;
    $t_{PIM} = GetPimLatency(EP_{thro})$;
    if $N_{EP\text{-}TK}[EP_{thro}] > threshold$ **then**
      // Offload the expert to NPU
      $M_{EP\text{-}PIM} = M_{EP\text{-}PIM} \setminus EP_{thro}$;
      $NPU_{target} = argmin([t_{NPU}])$;
      $t_{NPU}[NPU_{target}]$ += t_{LD};
      $t_{CH}[CH_{thro}]$ += $t_{LD} - t_{PIM}$;
    else
      break; // The CH has optimized
    end
  else
    break; // No experts left for PIM
  end
end

to PIM if its latency is lower than that of on-chip execution. MoNDE adjusts the number of experts to offload to NDP based on the bandwidth ratio between the NDP and PCIe. We thus propose a throttle-aware offloading method to address performance limitations caused by PIM and NPU throttle.

Given that the memory controller processes normal access and PIM commands sequentially, each DRAM channel latency (t_{CH}) includes two components: the expert loading latency (t_{LD}) and the PIM execution latency (t_{PIM}). The latency t_{PIM} for a given expert is the product of the number of offloaded tokens N_{TK} and the latency for one PIM-based FFN, t_{FFN}, which itself includes matrix-vector multiplication latency t_{GEMV} and element-wise operation latency t_{ELEM}. The loading latency t_{LD} is determined by the parameter size S_{EP} and the channel's effective bandwidth BW. As for the NPU latency, it is approximated by the product of the number of offloaded experts N_{EP} and the loading latency t_{LD}, considering the tiled pipeline execution and data transfer bottleneck. The timing model is given below:

$$t_{CH} = \sum t_{LD} + \sum t_{PIM} \begin{cases} t_{LD} = S_{EP} \div BW \\ t_{PIM} = N_{TK} \times t_{FFN} \end{cases} \quad (1)$$

$$t_{NPU} = N_{EP} \times (t_{LD} + t_{EXE}) \approx N_{EP} \times t_{LD} \quad (2)$$

Algorithm 1 depicts the process of the proposed offloading algorithm. The expert-channel mapping table ($M_{CH\text{-}EP}$) and expert-token allocation table ($M_{EP\text{-}TK}$) are taken as inputs to generate the PIM-expert table ($M_{EP\text{-}PIM}$), which represents the set of experts offloaded to PIM. Initially, all experts are offloaded to the PIM. The algorithm iteratively selects a target expert to offload to the NPU until the NPU becomes the bottleneck. It begins by identifying the channel with the highest latency and checking for any active experts in that channel. If there are experts whose workloads exceed the defined $threshold$ (the latency ratio of matrix loading to PIM-GEMV, as noted in Section II-B), the expert with the most tokens is selected. It is then offloaded to the NPU with the least latency, and the latency scoreboards for both the NPU and PIM are updated according to the predefined model.

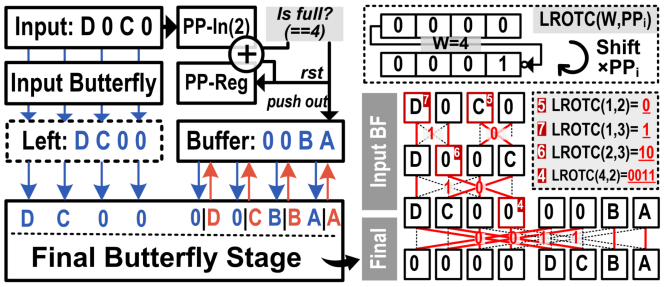


Fig. 4. Diagram of the near-memory-controller data condenser.

At the circuit level, the offloading algorithm is hardcoded. The timing parameters (i.e., t_{LD}/t_{GEMV}/$threshold$) are implemented as registers, which can be programmed with measured values. The offloading logic looks up the parameters and places the task into the target queue for dynamic scheduling.

C. Near-memory-controller Data Condenser

Given DRAM's limited logic budget, PIM struggles with irregular, non-extreme sparse GEMV, whereas NPUs can reorganize sparse computation into a densified pattern. Additionally, DRAM transfers are triggered in bursts, accessing consecutive data that may contain zero elements. This brings a mismatch in data layout between PIM and NPU, and leads to underutilization of NoC bandwidth and potential congestion.

To address this, we deploy a near-memory-controller data condenser to filter out zero elements from the input buffer, injecting dense 256-bit flits into the NoC. The condenser's core is a butterfly network reorganizing scattered non-zero elements into a consecutive sequence. As shown in Fig. 4, the butterfly network consists of multiple switch layers, each controlled by a bit to determine whether to swap the inputs. Using the input vector's bitmap and the rightmost offset, the LROTC (left rotate and complement on wrap) operation [19] recursively generates configuration bits for each switch layer, implemented through nested bitwise barrel rotators. A counter tracks the number of non-zero elements in the output buffer to align the inputs and outputs. This counter provides the offset to guide the LROTC within the input stage of the butterfly network, which permutes the input vector. The final butterfly stage further consolidates the reordered input into the output buffer. After each input cycle, the counter is incremented by the input's pop-count (i.e., the number of non-zero elements). Once the counter reaches its threshold, indicating that the

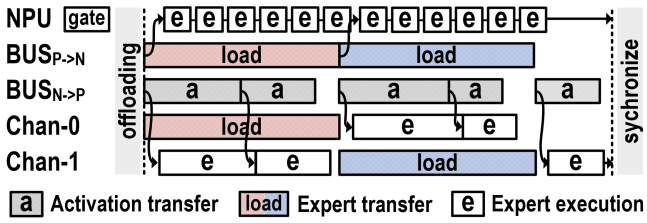


Fig. 5. Behavior model of the PIMoE simulator.

buffer is fully populated with dense elements, the data is pushed to the output queue, and the counter resets.

IV. Evaluation

A. Experimental Setup

Methodology: To evaluate the performance of the PIMoE platform, we develop a simulator composed of several subsystems. For the PIM subsystem, we use Samsung's open-source PIM-HBM simulator, PIMSimulator [17], which is based on the cycle-accurate DRAMSim [20]. PIMSimulator implements a PIM-specific command set to support common PIM testbenches, such as GEMV and element-wise computations. Regarding the execution latency of a PIM-offloaded expert, we simulate the two linear layers and an intermediate activation layer. With barriers inserted between each phase, the total latency is computed as the sum of the individual phases, which are executed sequentially. For the NPU subsystem, a pipelined scheme is adopted for weight loading and computation, where the expert is tiled for fine-grained execution. Therefore, the data transfer determines the latency, which is also evaluated by the PIMSimulator. For the NoC subsystem, we utilize the noxim simulator [21] to evaluate the congestion. We generate synthetic traffic patterns to represent the practical transfer workload in PIMoE. The injection rate is determined by the ratio of the DRAM channel's I/O throughput to the product of flit size (i.e., the NoC bus width, set as 256 bits) and the packet size (set as 8). The behavior model of PIMoE for simulating MoE FFN execution is illustrated in Fig. 5. Initially, the simulator determines the offloading based on the proposed throttle-aware offloading method. Then it tracks the state of each processing unit based on the pre-evaluated latencies. Once the occupancy of a unit is released, the simulator schedules a new task for execution.

TABLE I
PIMoE System Configuration and Workload Details

PIMoE	Composition	4 NPU, 2 PIM-HBM
NPU	Frequency	1GHz
	Matrix Unit	4096 MAC units, 8.2 TFLOPS
	Scratchpad	Activation 3MB, Weight 1MB
PIM	Memory configuration	HBM2, 2GB(4H), 2.4Gb/s, 307GB/s 16 channels/cube, 16 banks/channel
	Timing parameters	tCK=1ns, tCCDS=2ns, tCCDL=4ns, tRAS=33ns, tWR=16ns, tRP=14ns, tRCDRD=14ns, tRCDWR=10ns
	Processing Unit (PU)	300MHz, 8 PUs per channel 9.6 GFLOPS per PU

Model	d_{model}	d_{ff}	N_{expert}	Expert Params
Switch-Large-128	1024	4096	128	48 GB
Swicth-Base-16	768	3072	16	1.7 GB

Implementation: Table I presents the configuration of the PIMoE system. To accommodate an entire MoE FFN layer on PIMoE, we integrate two PIM-HBM devices with a total capacity of 4GB. This capacity is sufficient to store the layer-wise weight parameters (2GB per switch-large-128 MoE FFN layer), intermediate results, bitmaps, and other necessary data. Each expert is tiled along the output channels, enabling concurrent execution across the two devices. For GPU comparison, we measure the latency of the MoE layers using PyTorch's CUDA Event API, placing hooks at the entry and exit points of the inference process. To evaluate the energy consumption, we refer to the physical parameters provided in [23]. To evaluate the area overhead of the data condenser on the logic die, we implement it with RTL using a commercial 28nm technology library and synthesize it with Design Compiler.

Workloads: We evaluate PIMoE using the pre-trained Switch Transformers in the Hugging Face repository. We adopt the bfloat16 datatype, which is widely adopted for inference tasks. Since PIMoE is primarily customized for MoE tasks, we conduct our experiments solely on the MoE FFN layers. The details of the model are listed in Table I. As for the datasets, we choose the C4 dataset, a large-scale web-crawled text collection designed for evaluating language models. When evaluating performance under different token numbers, we disable padding and truncate sequences that exceed the target token count. This ensures that only effective tokens are considered during execution.

B. Experimental Analysis

Comparison of Performance. Fig. 6 compares the execution latency of an MoE FFN layer. Each scenario is normalized to the GPU baseline, NVIDIA's A100. For the switch-base-16 model, PIMoE achieves an average of $1.6\times$ speedup compared to the A100. For the larger switch-large-128 model, PIMoE demonstrates a significant latency reduction of $8.47 \sim 3.25\times$ across various token numbers. The results indicate that, with fewer total tokens, the memory-bound cold experts become more prominent, allowing PIM execution to yield greater

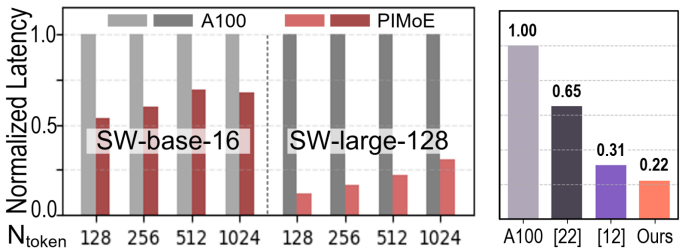
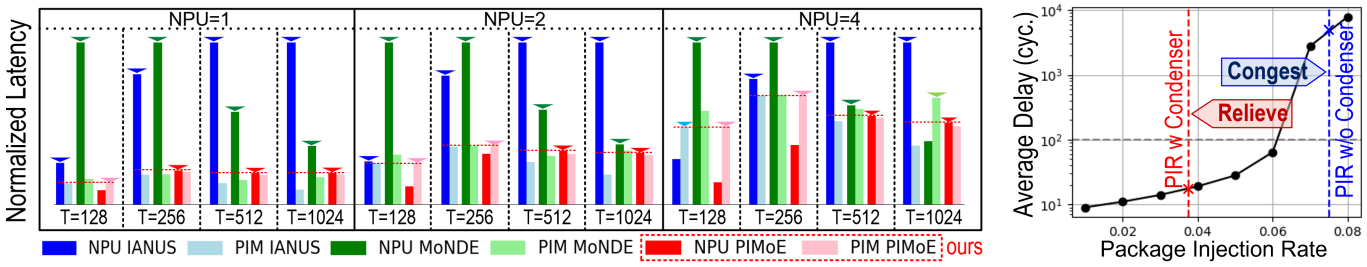
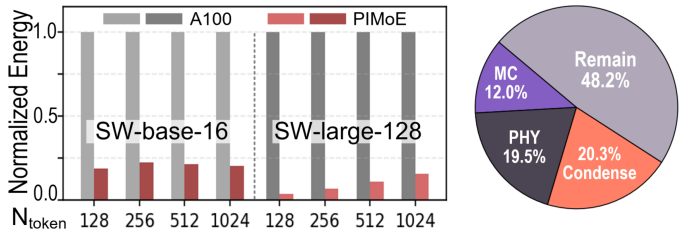


Fig. 6. Comparison of normalized latency (a) between A100 and PIMoE (b) between FLAME [22], MoNDE [12], and PIMoE.



(a) (b)

Fig. 7. (a) Comparison of the performance between the proposed offloading method and other NPU-PIM systems. (b) Congestion condition of the NoC under different packet injection rates (PIR).



(a) (b)

Fig. 8. (a) Comparison of energy efficiency between A100 and PIMoE across different scenarios. (b) Area breakdown of the logic die.

performance gains. We also compared PIMoE with other state-of-the-art accelerators for MoE transformers, specifically FLAME [22] and MoNDE [12]. We use their reported results normalized to the same GPU baseline (i.e., A100). Results show that PIMoE achieves an average improvement of 1.4× over MoNDE. PIMoE benefits from the higher bandwidth of bank-wise parallelism, unlike the channel-wise parallelism adopted in MoNDE. Additionally, PIMoE's tighter coupling between NPU and PIM bypasses the PCIe bottleneck in the GPU and CXL-based DRAM.

Comparison with other offloading scheme. Fig. 7(a) compares the effect of the proposed throttle-aware offloading scheme with the load balancing methods introduced in IANUS [8] and MoNDE [12], respectively. We implement their schemes on the PIMoE hardware baseline to ensure an apples-to-apples comparison. Each scheme's NPU and PIM latencies are shown separately to intuitively highlight the system's throttle points, indicated by a triangular arrow at the top of the corresponding bar. We also evaluate the sensitivity to varying numbers of NPUs and token workloads. For each scenario, the latencies are normalized to the highest value for comparison. The results show that throttle-aware offloading maintains a balance between NPU and PIM, particularly under heavy total token workloads and fewer NPU resources, leading to optimal performance across various configurations.

Effect of the data condenser. The number of effective packets injected into the NoC can be reduced through near-memory-controller data condensing. Particularly, the injection rate is halved with a 4:8 pruning. This prevents the NoC congestion under concurrent expert weight loading. Fig. 7 shows the average delay of data transfer under different packet injection rates (PIR). When no condensing is applied, the injection rate reaches 0.075, resulting in significant delays (over 5000 cycles) and indicating severe congestion. In contrast, with the condenser, NoC saturation is postponed. At an injection rate of 0.0375, the average delay is reduced to around 16 cycles, resulting in a negligible impact on overall performance.

Area and Power Consumption. As shown in Fig. 8, PIMoE achieves an average 4.8× and 13.7× improvement in energy efficiency compared to the A100 for the switch-base-16 and switch-large-128 models, respectively. This energy saving is attributed to the reduced external data transfer energy enabled by near-bank PIM. As for the near-memory-controller data condenser, we compare its area requirements with those of the memory controller, PHY, and the available area on the logic die. By efficiently utilizing the area budget, the condenser alleviates NoC congestion with a modest 20.3% overhead.

C. Discussion on Scalability

As the expert amount scales up, additional PIM-HBM devices can be added, with each expert tiled across the devices. While this increases NoC complexity, the data condensers mitigate the NoC's transfer burden, preventing congestion in larger configurations. Additionally, the proposed throttle-aware offloading method analyzes and schedules the complex expert distribution, optimizing resource usage and minimizing bottlenecks. This enables efficient handling of larger models and expert configurations while maintaining performance. By programming the timing parameters, this method can be applied to different NPU-PIM platforms.

V. Conclusion

In this paper, we presented PIMoE for efficient MoE Transformer deployment. By leveraging all-bank parallelism, PIMoE effectively addresses memory-bound challenges in MoE FFN computations. The throttle-aware task offloading mitigates workload imbalances between PIM and NPU. Additionally, the near-memory-controller data condenser improves data transfer efficiency by compressing sparse weight parameters from DRAM. Our experimental results show that PIMoE achieves 4.5× speedup and 13.7× greater energy efficiency compared to the A100, and 1.4× speedup over a state-of-the-art MoE platform. These results demonstrate the potential of PIMoE for improving performance and energy efficiency in large-scale MoE Transformer systems, providing a solid foundation for future developments in this area.

REFERENCES

[1] T. B. Brown, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[4] M. Artetxe, S. Bhosale, N. Goyal, T. Mihaylov, M. Ott, S. Shleifer, X. V. Lin, J. Du, S. Iyer, R. Pasunuru *et al.*, "Efficient large scale language modeling with mixtures of experts," *arXiv preprint arXiv:2112.10684*, 2021.

[5] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International conference on machine learning*. PMLR, 2022, pp. 18 332–18 346.

[6] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[7] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5547–5569.

[8] M. Seo, X. T. Nguyen, S. J. Hwang, Y. Kwon, G. Kim, C. Park, I. Kim, J. Park, J. Kim, W. Shin *et al.*, "Ianus: Integrated accelerator based on npu-pim unified memory system," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 545–560.

[9] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "Attacc! unleashing the power of pim for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 103–119. [Online]. Available: https://doi.org/10.1145/3620665.3640422

[10] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 722–737. [Online]. Available: https://doi.org/10.1145/3620666.3651380

[11] Z. Yu, S. Liang, T. Ma, Y. Cai, Z. Nan, D. Huang, X. Song, Y. Hao, J. Zhang, T. Zhi *et al.*, "Cambricon-llm: A chiplet-based hybrid architecture for on-device inference of 70b llm," *arXiv preprint arXiv:2409.15654*, 2024.

[12] T. Kim, K. Choi, Y. Cho, J. Cho, H.-J. Lee, and J. Sim, "Monde: Mixture of near-data experts for large-scale sparse models," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3655951

[13] S. Yun, K. Kyung, J. Cho, J. Choi, J. Kim, B. Kim, S. Lee, K. Sohn, and J. H. Ahn, "Duplex: A device for large language models with mixture of experts, grouped query attention, and continuous batching," *arXiv preprint arXiv:2409.01141*, 2024.

[14] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[15] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim *et al.*, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 350–352.

[16] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim *et al.*, "A 1ynm 1.25 v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 1–3.

[17] SAIT, "Pimsimulator," https://github.com/SAITPublic/PIMSimulator, 2024.

[18] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1573–1586, 2022.

[19] Y. Hilewitz, Z. J. Shi, and R. B. Lee, "Comparing fast implementations of bit permutation instructions," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 2. IEEE, 2004, pp. 1856–1863.

[20] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[21] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2015, pp. 162–163.

[22] X. Lin, H. Tian, W. Xue, L. Ma, J. Cao, M. Zhang, J. Yu, and K. Wang, "Flame: Fully leveraging moe sparsity for transformer on fpga," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3656507

[23] S. Kim, S. Kim, K. Cho, T. Shin, H. Park, D. Lho, S. Park, K. Son, G. Park, S. Jeong *et al.*, "Signal integrity and computing performance analysis of a processing-in-memory of high bandwidth memory (pim-hbm) scheme," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 11, pp. 1955–1970, 2021.

YAP: Yield Modeling and Simulation for Advanced Packaging

Zhichao Chen, Puneet Gupta
Department of Electrical and Computer Engineering, University of California, Los Angeles
zhichaochen@ucla.edu, puneetg@ucla.edu

***Abstract*—Three-dimensional integration technologies present a promising path forward for extending Moore's law, facilitating high-density interconnects between chips and supporting multi-tier architectural designs. Cu-Cu hybrid bonding has emerged as a favored technique for the integration of chiplets at high interconnect density. This paper introduces YAP, a yield model for wafer-to-wafer (W2W) and die-to-wafer (D2W) hybrid bonding process. The model accounts for key failure mechanisms that contribute to yield loss, including overlay errors, particle defects, Cu recess variations, excessive wafer surface roughness, and Cu density. We also develop an open-source yield simulator and compare the accuracy of the near-analytical yield model with the simulation results. The results demonstrate that YAP achieves virtually identical accuracy while offering over 10,000x faster runtime. YAP enables the co-optimization of packaging technologies, assembly design rules, and overall design methodologies. We used YAP to examine the impact of bonding pitch, compare W2W and D2W hybrid bonding for varying chiplet sizes, and explore the benefits of tighter process controls, such as improved particle defect density.**

***Index Terms*—yield modeling, hybrid bonding, wafer-to-wafer (W2W), die-to-wafer (D2W), 3D integration, chiplet, roughness, pad recess, peeling stress, particle defects, overlay, warpage/bow.**

I. Introduction

As we approach the physical and economic limits of scaling in two-dimensional integrated circuits, Three-Dimensional Integrated Circuits (3D-ICs) have emerged as a promising solution to sustain the momentum of Moore's Law. By stacking multiple layers of devices vertically, 3D-ICs offer significant benefits such as reduced interconnect lengths, enhanced performance, lower power consumption, and higher integration density. A pivotal technology that enables 3D-ICs is hybrid bonding (HB) which facilitates reliable fine pitch interconnections, making it highly suitable for high-bandwidth memory, logic-memory integration, and advanced sensor applications [1]–[4]. The two popular HB variants are Wafer-to-wafer (W2W) which allows for a higher alignment accuracy and die-to-wafer (D2W) bonding which allows for better leveraging of pre-tested known-good-die [1].

Accurate, predictive yield modeling for advanced packaging is essential is key to identifying potential failure mechanisms early in the technology development cycle, enabling system-technology co-optimization, and guiding the development of the chiplet interconnect repair strategy [6]. The overall assembly yield model for advanced integration is determined by multiple components, including chiplet yield, hybrid bonding yield, and through-silicon via (TSV) yield. Though full system yield with simplified underlying models has received a lot of attention [7]–[10], models of the yield of hybrid bonding have largely been overly simplistic. [7], [8] propose system-level yield models specific to 3D stacked ICs, However, [7] does not extend its yield model to include the bonding process, and [8] assumes the bonding yield as a constant, which is also done by [10] when building an assembly yield for a chiplet system. None of these studies considers the complex physical failure mechanisms or derives a concrete yield model for the HB process. Such modeling becomes even more important as the industry strives to rapidly scale down the bonding pitches in the next decade.

Failure mechanisms for hybrid bonding are complex which makes analytical modeling of yield challenging. This paper proposes YAP, a physical mechanism-driven near-analytical yield model. To the best of our knowledge, YAP is the first yield model tailored for the HB process, which provides a detailed analysis and modeling framework to predict and optimize bonding yield. The main contributions are summarized as follows:

- To the best of our knowledge, we present the first yield model specifically for the HB process. This model incorporates critical failure mechanisms contributing to yield loss, including overlay errors, particle defects, Cu recess variations, surface roughness of wafers and dies, and excessive Cu density.
- We construct a yield simulator based on the distribution parameters of various failure mechanisms to validate the proposed yield model and assess its predictive accuracy.
- We perform case studies examining various factors that impact yield, compare the yield performance of W2W and D2W HB approaches, and demonstrate the necessary process control for a high yield performance.

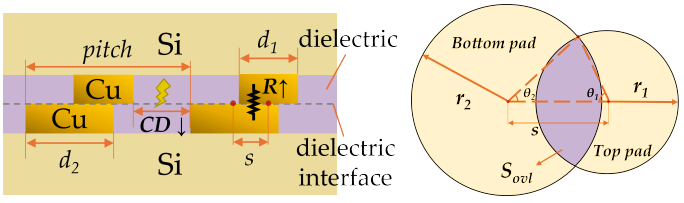
The remainder of the paper is organized as follows: Section II introduces key failure mechanisms of HB processes. Section III proposes modeling methodologies for W2W HB and extends it to D2W HB. Section IV describes the experiment setting, compares the results from the near-analytical model and the simulator, and conducts case studies on critical yield-influencing factors. Finally, Section V concludes the paper and gives suggesting directions for future research.

II. Overview of Failure Mechanisms of Hybrid Bonding

This section provides an overview of the primary failure mechanisms in HB processes, including overlay errors, Cu recess variations, and particle defects. Each mechanism contributes to yield loss when bonding parameters are inadequately controlled, and understanding these mechanisms is critical for optimizing the process yield.

A. Overlay Errors

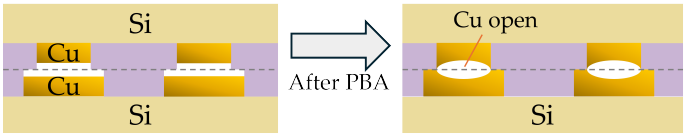
Ensuring the quality of the Cu connection formed during the HB process is critical for the electrical properties of the design. Misalignment of Cu pillars of the top and bottom wafers inevitably exists due to the robot arm calibration error, the warpage induced by thermal stress difference, etc. The misalignment influence on yield loss is increasingly pronounced as we are shrinking the pad size and using the sub-micron pitch [11]. As shown in Fig.1, the excessive misalignment will



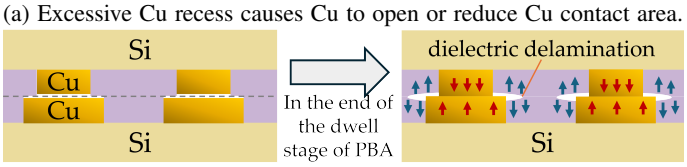
(a) Misalignment leads to resistance increase and dielectric breakdown.

(b) Contact area calculation given the misalignment.

Fig. 1: Failure mechanism of Overlay errors.

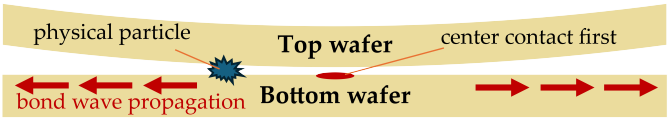


(a) Excessive Cu recess causes Cu to open or reduce Cu contact area.

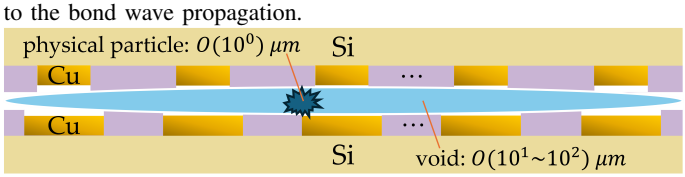


(b) Insufficient Cu recess may cause dielectric delamination, depending on the dielectric bonding strength, Cu density, and annealing temperature, etc.

Fig. 2: Failure mechanism of Cu recess variations.



(a) The presence of particle will form a main void and a void tail due to the bond wave propagation.



(b) Void formation can fail the dielectric and Cu bonding. A particle of a few microns can form a main void of hundreds of microns [38].

Fig. 3: Failure mechanism of particle defects.

decrease the contact area (S_{ovl}) of the Cu interface, hence increasing the contact resistance and triggering electromigration-related defects [12]. Also, the probability of dielectric breakdown will increase due to the reduced critical distance (CD), causing thinner insulating film between the upper and lower pads of adjacent pillars [2]. It is defined as the pitch being p, the pad being circular, and the top and bottom pad diameters being $d_1 = 2r_1$ and $d_2 = 2r_2$. The critical distance between two perfectly aligned Cu pillars is defined as $CD = p - d_2$, i.e., the distance between two neighboring pads. In some designs, the top pad size is set smaller than bottom pad size to increase the misalignment tolerance [28]. Overlay errors are typically classified as pad-level random errors and systematic errors.

B. Cu Recess Variations

The CMP process will result in Cu recess effects, forming a concave shape on the Cu pad surface. As presented in Fig. 2a, excessive Cu recess will decrease the bonding quality or even incur Cu connection failure after post-bond annealing (PBA) [17], [18]. Meanwhile, Cu protrusion and insufficient Cu recess can degrade the yield. Excessive wafer surface roughness decreases the effective contact area of the dielectric during low-temperature bonding, which lowers the density of covalent bonds after PBA, thereby weakening the bonding strength and energy per unit area at the dielectric interface [18]–[20]. In fine-pitch designs, high Cu density and insufficient Cu recess can incur an undesirable peak peeling stress at the dielectric interface at the end of the annealing dwell stage [21]–[24]. As can be seen in Fig. 2b, if the dielectric interface bonding is unable to withstand the peeling stress, dielectric delamination or cracking can occur, leading to bonding failure [25]–[27]. In summary, to achieve a high yield, especially for a chiplet with a large number of Cu pads, a well-controlled Cu recess variation of the top and bottom pad within a range determined by Cu pattern density, surface roughness, etc. is necessary.

C. Particle Defects

The HB process requires high-standard cleanliness to prevent the presence of physical particles, which will incur void formation at the bonding interface [1], [38]. In addition to physical particles, gas condensation during the bonding process can incur edge voids in the wafer bevel region. The outer edge region is usually removed by a sawing process and silicon dies or chips located inward from the wafer edge will not be influenced, therefore, there will be no yield impact [37]. In the defect model, the emphasis is on the yield loss resulting from the particle-induced void formation. As presented in Fig. 3, in the W2W HB process, due to the bond wave propagation, the presence of a particle at the bonding interface will result in a *main void* with a trailing *void tail* extending radially [38].

III. YAP Yield Model

In this section, we introduce our yield modeling methodology for W2W HB and then extend the model to D2W HB. To validate our derived model, we compare the modeling results with the simulation results across 300 parameter sets. The workflow of simulation and model validation is illustrated in Fig. 4. The detailed experiment setting will be discussed in Section. IV.

A. Overlay Model

Our proposed overlay model estimates the yield loss caused by the Cu pad misalignment. We assume the bonding misalignment is normally distributed with zero mean and process-specific σ_1 [13]. Then, the possibility of survival (POS) of one single pad can be calculated as

$$POS_{ovl,pad} = \frac{1}{\sigma_1\sqrt{2\pi}} \int_{-\delta}^{\delta} e^{-\frac{u^2}{2\sigma_1^2}} \mathrm{d}u \tag{1}$$

where u denotes the random overlay error of the top and bottom pad, and δ represents the maximum allowed overlay error to ensure the pad's survival. δ should be determined by the contact area constraints and the critical distance constraints. We assume s as the systematic overlay error of a Cu connection caused by three distortion components: *translation*, *rotation*, and *magnification* [14]. The translation and rotation mainly originate from the equipment precision limitations, and the magnification mainly comes from the wafer warpage/bow induced by thermal expansion mismatch of multiple materials [21]. We define the translation errors as T_x, T_y in x, y directions, respectively, and the rotation error as α. Typically, the warpage of bonded wafers can range from a few micrometers to over 100 μm, which can be optimized to $\sim$10 μm through run-out compensation [16]. Let B represent the warpage of the bonded wafer. It can be observed that the magnification factor E is linearly correlated to B [15], [16]. We can build a linear model to characterize the magnification factor E as

$$E = k_{mag} \cdot B \tag{2}$$

where k_{mag} is the fitting parameter of the model. The parameter is related to the Cu pad depth, Cu area density, bonding process temperature, etc [21]. We model the systematic misalignment $\Delta x, \Delta y$ in x, y directions respectively by

$$\begin{cases} \Delta x(x,y) = T_x - \alpha \cdot y + E \cdot x, \\ \Delta y(x,y) = T_y + \alpha \cdot x + E \cdot y. \end{cases} \tag{3}$$

The systematic overlay error s at the location (x, y) is by

$$s(x,y) = \sqrt{[\Delta x(x,y)]^2 + [\Delta y(x,y)]^2} \tag{4}$$

As Fig. 1b shows, the contact area of two Cu pads is by

$$S_{ovl} = \begin{cases} \pi r_1^2, & s < r_2 - r_1 \\ \theta_1 r_1^2 + \theta_2 r_2^2 - s r_1 \sin\theta_1, & r_2 - r_1 \le s \le r_1 + r_2. \\ 0, & s > r_1 + r_2 \end{cases} \tag{5}$$

If we assume that, to ensure the pad survival, the contact area should exceed k_{ca} times the surface area of the top pad interface, i.e., $S_{ovl} > k_{ca}\pi r_1^2$, and the critical distance CD should be greater than k_{cd} times the ideal critical distance, i.e.,

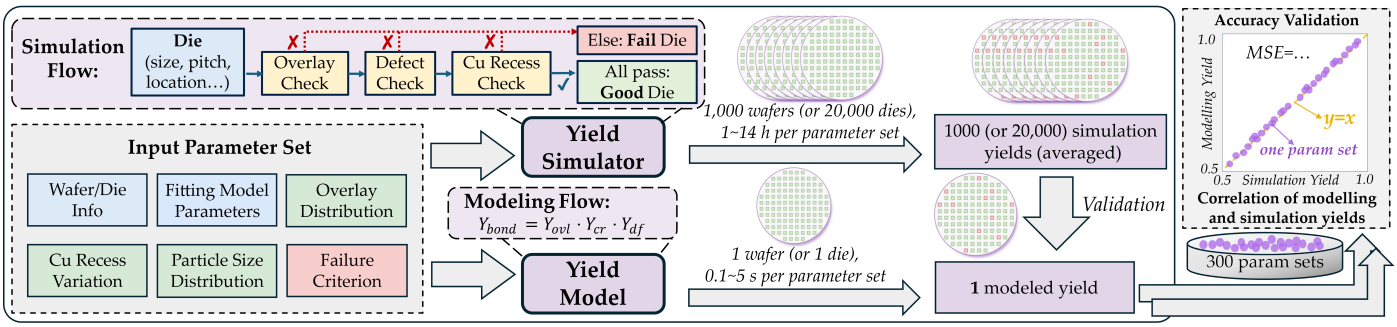


Fig. 4: Simulation workflow and the validation of modeling yield on various input parameter sets.

$CD > k_{cd}(p - d_2)$, then δ can be given by

$$\delta = \min \left\{ \frac{\theta_1 r_1^2 + \theta_2 r_2^2 - k_{ca}\pi r_1^2}{r_1 \sin\theta_1}, \right. \\ \left. (1 - k_{cd})p - \frac{1}{2}d_1 + \left(k_{cd} - \frac{1}{2}\right) d_2 \right\} \tag{6}$$

Given above, the POS of a die with N pads is given by

$$POS_{ovl,die} = \frac{1}{\sigma_1\sqrt{2\pi}} \min_{i\in[1,N]} \left\{ \int_{-\delta - s_i}^{\delta - s_i} e^{-\frac{u^2}{2\sigma_1^2}} \mathrm{d}u \right\} \tag{7}$$

where s_i denotes the systematic overlay residue of the i-th pad in one die. Assuming one wafer has M dies, the overlay die yield of the wafer can be calculated as

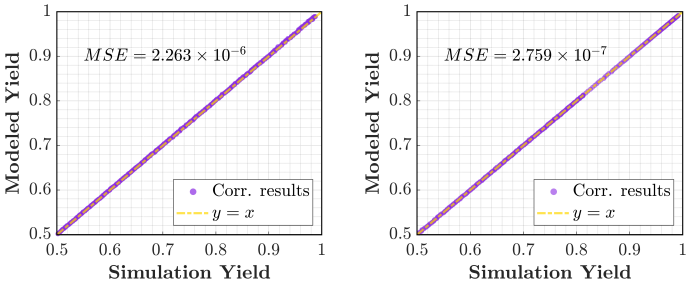
$$Y_{ovl,W2W} = \frac{1}{M} \sum_{j=1}^{M} POS_{ovl,die,j} \tag{8}$$

We vary input parameters for both the model and simulator, including factors such as translation error, rotation error, warpage, die size, and others. The 300 comparison results (purple points) and the mean squared error (MSE) in Fig. 5a demonstrate that our model aligns closely with the simulation data, confirming its reliability and accuracy.

B. Cu Recess Model

We can assume the pad height after the CMP process is normally distributed according to [28], [30]. Let the dielectric surface be zero reference. The height will be negative for the recessed pad and positive for the protruded pad. It is evident that the sum of the top and bottom pad heights also follows a normal distribution. Let h represent the sum of pad heights of two corresponding pads. The mean value of this distribution is denoted by μ_h, and the variance by σ_h^2. It is noticed that h should be clamped within a range of (ζ_-, ζ_+) so that the Cu bonding failure and dielectric delamination can be avoided. Below the calculation of ζ_- and ζ_+ is discussed.

a) Calculation of ζ_-: Based on the observation in [30]–[32], the height difference due to Cu expansion during annealing is linearly correlated to the annealing temperature. The *lower bound* ζ_- of the total Cu heights required to form a qualified Cu bonding area is determined by the sum of Cu expansion after PBA so that the gap between two pads caused by the recess is adequately filled with Cu.



(a) Overlay model. (b) Cu recess model.

Fig. 5: Correlation results of the overlay model and the Cu recess model with the simulation data for W2W HB.

b) Calculation of ζ_+: The critical condition for delamination is that the sum of heights equals ζ_+. It is noticed that surface roughness will decrease the effective contact area of two surfaces. We use the asperity-based roughness model proposed by [19], [33] to calculate the normalized effective contact area $A_b^*(\sigma_z, R_z, E_d, w)$ given roughness and bonding parameters such as the standard deviation of asperity height σ_z, asperity cap radius R_z, Young's modulus of contact surface material E_d, and fully contact bonding energy w [1]. The maximum tolerable peeling stress σ_{tol} to avoid delamination can be given by

$$\sigma_{tol} = A_b^*(\sigma_z, R_z, E_d, w) \times \sqrt{\frac{2E_d w}{t_d}} \tag{9}$$

where t_d denotes the thickness of the surface material [35]. As the ambient temperature changes during the PBA, the thermal expansion mismatch between metal and dielectric will incur various stresses on the bonding interface [27]. Among different interfaces, dielectric-dielectric (e.g. SiO_2-SiO_2) interface is more inclined to delaminate due to relatively lower bonding strength and higher peeling stress at the end of the annealing dwell stage [22], [25]. For simplification, we use a fitting model to evaluate dielectric interface peeling stress dependence by

$$\sigma_{peel} = k_{peel} \cdot D_{Cu} \cdot (h - h_0) \tag{10}$$

where D_{Cu} denotes the Cu pattern density, h_0, k_{peel} are fitting parameters, and k_{peel} is related to the annealing temperature, pad shape, pad arrangement, pad structure, etc. [22], [23], [27]. To prevent delamination, one should have

$$\sigma_{tol} \geq \sigma_{peel} \Rightarrow h \leq h_{peel} \tag{11}$$

Additionally, given the Cu protrusion after CMP will incur delamination, the *upper bound* of the sum of heights is by

$$\zeta_+ = \min\{0, h_{peel}\} \tag{12}$$

To summarize, the POS of this pad during PBA is given by

$$POS_{cr,pad} = \frac{1}{\sqrt{2\pi\sigma_h^2}} \int_{\zeta_-}^{\zeta_+} e^{-\frac{(h-\mu_h)^2}{2\sigma_h^2}} \mathrm{d}h \tag{13}$$

The die yield, as the POS of a die with N pads with respect to the factor of the Cu recess variations is given by

$$Y_{cr,W2W} = POS_{cr,die} = POS_{cr,pad}^N \tag{14}$$

We vary input parameters of Cu recess, pitch, roughness, etc. to validate the Cu recess model. Fig. 5b presents the correlation of the proposed Cu recess model with the simulation data.

C. Defect Model

The relationship between a particle's properties and void size is complex. Due to the significant discrepancy between theoretical and experimental results, it is more practical to develop a fitting model to predict the void size given the particle information. Additionally, for a particle of a specific size and material, it is shown in [3], [38], that the main void size and the void tail length are linearly correlated with the particle's location and the square root of particle thickness. We model

[1]To model the interaction of two rough surfaces, the surface roughness σ_z and Young's modulus E_d need to be normalized [34].

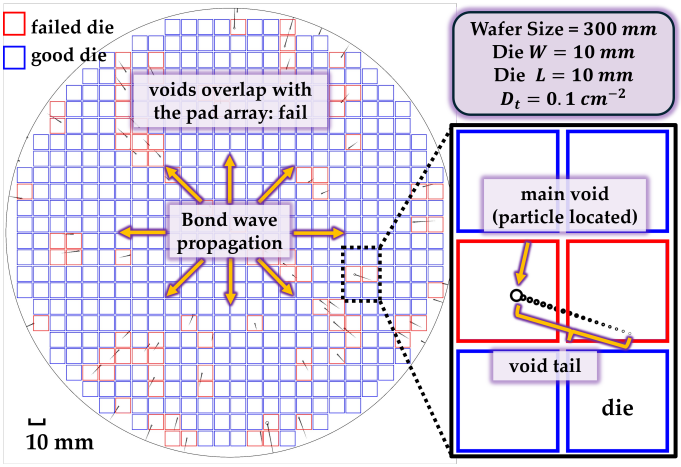


Fig. 6: Visualization of the void formation simulation.

the size r_{mv} of the main void that is located with the distance L $(0 \leq L < R)$ from the wafer center by

$$r_{mv} = (k_r L + k_{r_0}) t^{1/2} \tag{15}$$

where t is the particle thickness. Similarly, the void tail length l can be modeled by

$$l = k_l L t^{1/2} \tag{16}$$

where k_r, k_{r_0}, k_l are fitting parameters. Fig. 6 visualizes the simulation of the void formation, which is similar to the scanning acoustic microscopy images of voids in [38]. Since the average void tail length on the wafer can achieve a few millimeters, more than 10 times the scale of the main void size (a few hundred μm in W2W HB, the defect shape can be reasonably simplified as a line as illustrated in Fig. 7. Additionally, a die is considered to have failed if the void tail overlaps the pad array area, as the void size is typically much larger than the pitch ($\leq$ 10 μm).

We assume the thickness distribution of particle defects as $D(t)$. A typical form of $D(t)$ can be [39]

$$D(t) = D_t \cdot \frac{(z-1) \cdot t_0^{z-1}}{t^z}, \qquad t > t_0 \tag{17}$$

where t_0 is the smallest particle thickness, and D_t is the total number of particles of all thicknesses per unit area on the die. The curve shaping parameter z is typically assumed to be between 2 and 3 [40], [41]. By Eq. 16, 17, the distribution of void tail length is given by

$$f_l(l) = \begin{cases} \dfrac{2D_t(z-1)l}{z k_l^2 R^2 t_0}, & l \leq k_l R t_0^{1/2} \\ \dfrac{2D_t(z-1)(k_l^2 R^2 t_0)^{z-1}}{z l^{2z-1}}, & l > k_l R t_0^{1/2} \end{cases} \tag{18}$$

where R denotes the radius of the wafer. The comparison of the derived $f_l(l)$ and the simulating distribution is shown in Fig. 8a, confirming the formula's accuracy. The critical area calculation method is presented in Fig. 7. The average critical area regarding the line defect with a length l across all directions, i.e., $0 < \phi < 2\pi$, can be given by

$$A(l) = ab + \frac{2}{\pi}(a+b)l \tag{19}$$

Hence, the average number Λ of particle-induced void tail defects that will fail a die can be given by

$$\Lambda = \int_0^\infty A(l) f_l(l) \mathrm{d}l = D_t ab + \frac{8D_t(z-1)}{3\pi(2z-3)} \cdot (a+b) k_l R t_0^{1/2} \tag{20}$$

Using the Poisson yield model [43], the yield with respect to the particle-induced void formation is given by

$$Y_{df,W2W} = \exp(-\Lambda) \tag{21}$$

We vary input parameters of particle defect density, die size, wafer size, etc. to validate the defect model. Fig. 8b demonstrates the correlation of the defect yield with the simulation results.

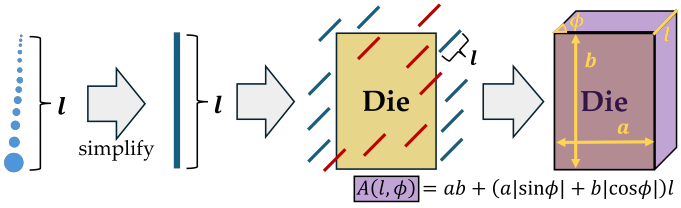
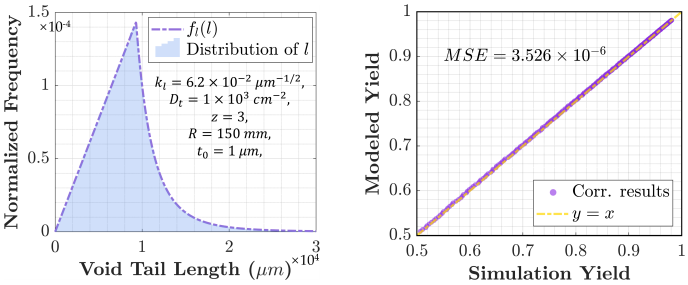


Fig. 7: Critical area calculation. Die size: $a \times b$. Void tail length: l. Void tail direction: ϕ.



(a) Void tail length distribution. (b) Defect model.

Fig. 8: Correlation results of the void tail length distribution and the defect model with the simulation data for W2W HB.

D. Overall Bonding Yield Model

To develop the overall bonding yield model for W2W hybrid bonding, we assume the overlay error, Cu recess variations, and particle defects have independent impacts on the die yield. By combining the Eq. 8, 14, 21, the assembly yield is

$$Y_{W2W} = Y_{ovl,W2W} \cdot Y_{cr,W2W} \cdot Y_{df,W2W} \tag{22}$$

E. D2W Hybrid Bonding Yield Models

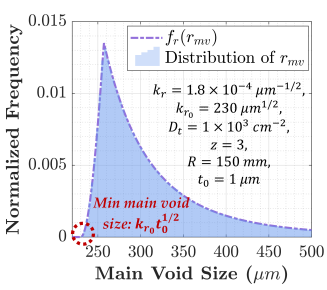
Here we extend the yield model for D2W HB. We assume that the Cu expansion mechanism during PBA in W2W HB remains consistent in the D2W HB case. However, the yield terms regarding the overlay error and particle defects should be revised to account for the differences in bonding approaches.

1) Overlay Model: In D2W hybrid bonding, the systematic overlay error independently happens die-to-die. Given that the die is much smaller than the wafer, the same marker alignment errors at the die edge result in a larger rotation error α and magnification error E than that on average in W2W HB. The overlay yield can be represented by the POS of the die as

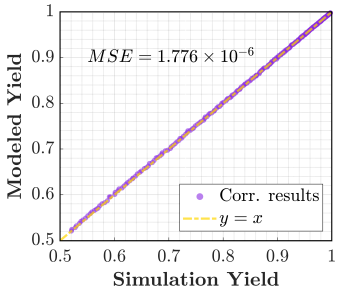
$$Y_{ovl,D2W} = \frac{1}{\sigma_1\sqrt{2\pi}} \min_{i\in[1,N]} \left\{ \int_{-\delta-s_i}^{\delta-s_i} e^{-\frac{u^2}{2\sigma_1^2}} \mathrm{d}u \right\} \tag{23}$$

2) Defect Model: Since the die scale is much smaller than the wafer's, the formation of the void tail is unlikely to occur. Therefore, we only consider the main void-induced failure in the D2W defect model. Combining Eq. 15, 17, the PDF of the main void size r_{mv} can be given by

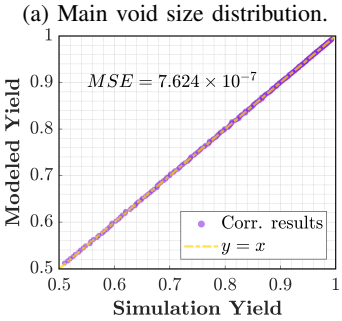
$$f_r(r_{mv}) = \begin{cases} \dfrac{D_t(z-1)t_0^{z-1}}{k_r^2 R^2} \times [\dfrac{2r_{mv}}{z t_0^z} + \dfrac{2k_{r_0}^{2z}}{z(2z-1)r_{mv}^{2z-1}} \\ - \dfrac{2k_{r_0}}{(z-\frac{1}{2})t_0^{z-\frac{1}{2}}}], \quad k_{r_0} t_0^{1/2} < r < (k_r R + k_{r_0}) t_0^{1/2}, \\ \dfrac{2D_t(z-1)t_0^{z-1}(k_r R + k_{r_0})^{2z-2}}{r_{mv}^{2z-1}} - \\ \dfrac{2D_t(z-1)^2 t_0^{z-1}}{k_r^2 R^2 r_{mv}^{2z-1}} \times [\dfrac{(k_r R + k_{r_0})^{2z} - k_{r_0}^{2z}}{z} \\ - \dfrac{2k_2(k_r R + k_{r_0})^{2z-1} - 2k_{r_0}^{2z}}{z - \frac{1}{2}} + \\ \dfrac{k_{r_0}^2(k_r R + k_{r_0})^{2z-2} - k_{r_0}^{2z}}{z-1}], \quad r \geq (k_r R + k_{r_0}) t_0^{1/2}. \end{cases} \tag{24}$$



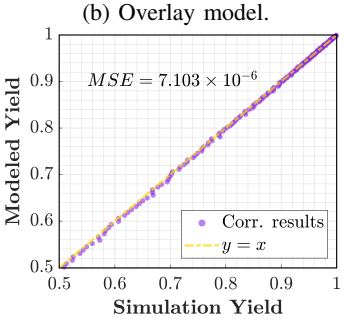
(a) Main void size distribution.



(b) Overlay model.



(c) Cu recess model.



(d) Defect model.

Fig. 9: Correlation results of main void size distribution and the yield model with the simulation data for D2W HB.

where R is the effective radius of the die, i.e., $R = (ab/\pi)^{1/2}$, aiming to remain the average number of particles within the die area. The good alignment between $f_r(r_{mv})$ and the simulated distribution is shown in Fig. 9a. To simplify the critical area calculation, we assume the defect and pad shape are square. The side lengths of voids and top pads are defined as $2r_v, 2r_1$, respectively. The critical area of the main void is given by

$$A(r_v) = \begin{cases} 4N(r_v + r_1)^2, & 2(r_v + r_1) \le p \\ [a + 2(r_v + r_1)] \cdot [b + 2(r_v + r_1)], & 2(r_v + r_1) > p \end{cases} \tag{25}$$

By Eq. 24, 25, the average number Λ of particle-induced main void defects that will fail a die is given by

$$\Lambda = \int_0^\infty A(r_{mv}) f_r(r_{mv}) \mathrm{d}r_{mv} \tag{26}$$

Using the Poisson yield model [43], the yield with respect to the particle-induced void formation for D2W HB is given by

$$Y_{df,D2W} = \exp(-\Lambda) \tag{27}$$

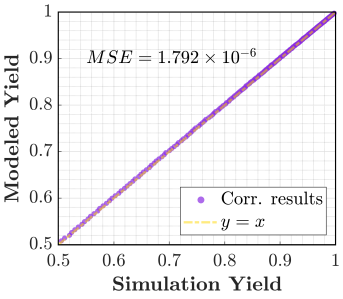
3) Overall Bonding Yield Model: Similarly, we assume the overlay error, Cu recess variations, and particle defects influence the die yield independently for D2W HB. Fig. 9b, 9c, 9d show the correlation results of three yield terms, respectively. By combining Eq. 14, 23, 27, the bonding yield is by

$$Y_{D2W} = Y_{ovl,D2W} \cdot Y_{cr,D2W} \cdot Y_{df,D2W} \tag{28}$$

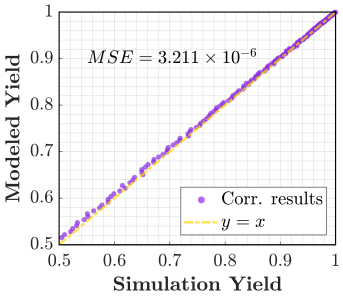
IV. Experimental Results

To validate the derived model, we developed a simulator focusing on Cu-SiO_2 hybrid bonding process. Table I presents the baseline model parameters, with additional details available in our code. These are the parameter values used in our experiments unless otherwise stated.

Fig. 4 outlines the simulation workflow. Overlay error, Cu recess, and particle data are sampled from their respective distributions. Particles are uniformly distributed across the wafer or die, and the void tails are generated based on the fitting model from [38], simulating bond wave propagation (Fig. 6). A die survives only if Cu pillar misalignment is within the safe range (*Overlay Check*), no void overlaps the Cu contact (*Defect Check*), and no dielectric delamination or Cu connection failure occurs (*Cu Recess Check*). The simulation results are closer to the actual conditions with less approximation compared to the model. However, to obtain an accurate yield prediction, simulation requires 1000 wafer samples (20,000 die samples)



(a) Yield of W2W HB.



(b) Yield of D2W HB.

Fig. 10: Correlation of bonding yield with the simulation.

for W2W (D2W) HB, taking 12.2 hours (0.5 hours) on a single CPU (AMD Ryzen 9 8945HS). In contrast, the yield model achieves virtually identical accuracy shown in Fig. 10 in 0.5 s (0.07 s) for W2W HB, offering over *10,000x runtime improvement* and enabling its usage in yield optimization and pathfinding optimization loops.

Below we discuss a few example case studies using the YAP model indicating its strengths in system-technology co-optimization. We vary particle defect density (0.01 cm^{-2}, 0.1 cm^{-2}), pitch (1 μm, 6 μm), and chiplet sizes (10 mm^2, 50 mm^2, 100 mm^2) in the experiment. The yield breakdown and the overall bonding yield are reported in Fig. 11 (W2W setup) and Fig. 12 (D2W setup).

A. Impact of Particle Defect Density

The HB process demands strict elimination of particles at the bonding interface. Fig. 11, 12 show that for the relaxed bonding pitch (6μm), bonding yield is completely defect-limited. Furthermore, W2W HB is more sensitive to particle contamination due to void tail formation during bond wave propagation, resulting in a larger critical area per die. As can be seen from the results, a 10X improvement in defect density (e.g., by following a more stringent cleanroom standard) ensures near-perfect bonding yield for both W2W and D2W for all chiplet sizes.

TABLE I: Baseline parameters in yield modeling and simulation

Parameter	Value
Pad pitch [27]	6 μm
Bottom, Top pad size [27]	3 μm, 2 μm
Die size [47]	10 mm $\times$ 10 mm
Wafer size	300 mm
Random misalignment [46]	5 nm (10 nm)*
System x,y translation [46]	5 nm (10 nm)*
System rotation [46]	0.1 μrad (0.05 μrad)*
Bonded wafer warpage [16]	10 μm (3 μm)*
System magnification [16]	0.9 ppm (0.07 ppm)*
Particle defect density [47]	0.1 cm^{-2}
Minimum particle thickness [38]	1 μm
Shaping factor (z) in Eq. 17 [40], [41]	3
Bottom/Pad recess [22], [28]	10.0 nm (1.0 nm)*
Roughness (σ_z) [20], [45]	1 nm
Adhesion energy (SiO_2-SiO_2) [18], [25]	1.2 J/m^2
Young's modulus (SiO_2) [22], [27]	73 GPa
Dielectric thickness [44]	1.5 μm
contact area constraint k_{ca} in Eq. 6 [11]	0.75
critical distance constraint k_{cd} in Eq. 6 [2]	0.75
k_{mag} in Eq. 2 [16]	0.09 m^{-1}
k_{peel} in Eq. 10 [22]	6.55$\times$ 10^{15} N·m^{-3}
h_0 in Eq. 10 [22]	75 nm
k_r in Eq. 15 [38]	1.8$\times 10^{-4}$ $\mu m^{-1/2}$
k_{r0} in Eq. 15 [38]	230 $\mu m^{1/2}$
k_l in Eq. 16 [38]	6.2$\times 10^{-2}$ $\mu m^{-1/2}$

Note: Distribution parameters with * are shown in Mean (Std.).

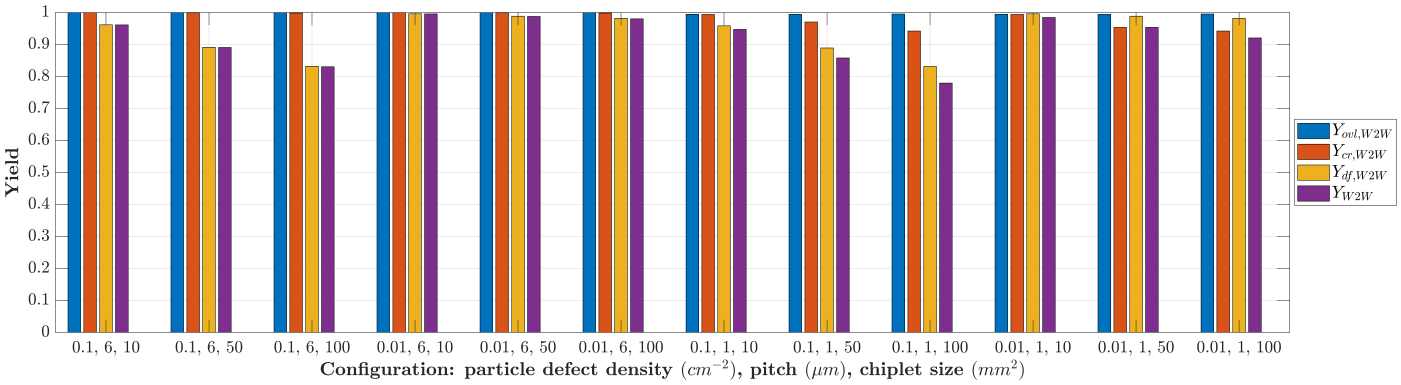


Fig. 11: W2W case studies for various configurations.

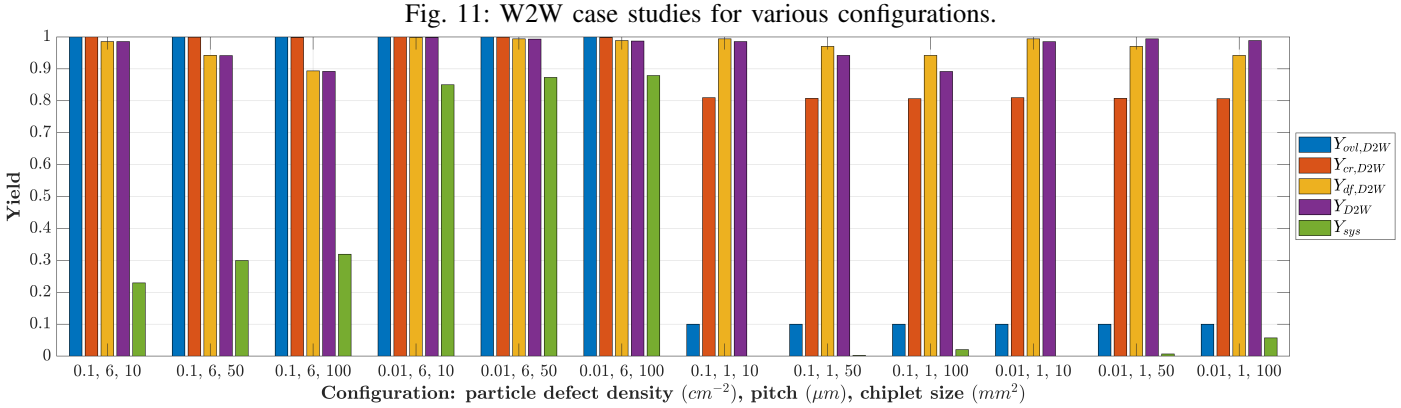


Fig. 12: D2W case studies for various configurations.

B. Impact of Bonding Pitch

In this experiment, the bottom pad size is set to half of the corresponding pitch. As shown in Fig. 11, 12, reducing the pitch from 6 μm to 1 μm leads to a noticeable yield decrease across various chiplet sizes, with the effect being more pronounced in D2W HB. In D2W HB, the primary yield loss stems from increasing overlay errors, as smaller pitches heighten sensitivity to Cu pillar misalignment. For overlay errors, the error α and E resulting from the marker alignment errors scale inversely with the maximum edge distance from the center, which is smaller for a chiplet in D2W HB than for a full wafer. For W2W HB, chiplets closer to the wafer center are more likely to survive, therefore achieving a higher $Y_{ovl,W2W}$. Meanwhile, reducing the pitch significantly increases the number of pads, amplifying the yield's sensitivity to Cu recess variations. For W2W HB increased yield loss is mainly driven by a decrease in Y_{cr} when bonding pitch is decreased. Defect yield, remains largely unaffected, as void size significantly exceeds the pitch, leaving the critical area unchanged.

Another observation is that W2W bonding fares far better than D2W bonding at finer pitches and the difference is even more pronounced at low defect densities. This primarily stems from the ease of high-accuracy alignment for the W2W case.

C. Analyzing Yield Limiters with Varying Chiplet Size

The bonding yield drops with increasing chiplet size for both D2W and W2W bonding, primarily driven by worsened copper recess yield (due to more number of pads per die) and defect yield.

D2W HB can be employed in 2.5D integration to assemble large chiplet systems. Therefore, just looking at a single chiplet yield can be misleading. Though our focus is not full system assembly yield modeling, we add a system yield (Y_{sys} in Fig. 12) calculated simply as $Y_{D2W}^{\#chiplets}$, in absence of any redundancy, for a nominal system size of 1000 mm^2. This approach reflects the cumulative probability of all chiplets bonding successfully in the system. Increasing the chiplet size, which reduces the number of chiplets required for system assembly, helps mitigate the compounding effect of Y_{D2W} losses. Interestingly, even though Y_{D2W} decreases as the chiplet size increases, Y_{sys} remains higher.[2]

V. Conclusion

This work presents YAP: a comprehensive yield modeling framework for W2W and D2W hybrid bonding in advanced packaging. YAP models overlay errors, particle-induced void defects, and Cu recess variations and is validated against a physics-inspired yield simulator[3]. The proposed YAP yield model accurately predicts bonding yield across various chiplet sizes, pitches, and pad configurations and achieves a 10,000-fold speed improvement over direct simulations while maintaining negligibly small mean square error. YAP enables fast system-technology co-optimization. Case studies using it highlight key differences between W2W and D2W HB, offering insights into yield-limiting failure mechanisms motivating future technology development and system architecture choices.

Our future work directions include incorporating YAP into a comprehensive system assembly yield model; extending YAP to model other forms of fine-pitch bonding such as thermal-compression bonding; evaluating some of the approximations made in YAP in the context of ultra-fine pitch bonding processes; and developing fault tolerance and yield improvement techniques leveraging YAP.

Acknowledgement

The authors thank Alexander Graening and Krutikesh Sahoo at UCLA and Nicolas Pantano at IMEC for their inputs during the early stages of the work. This work was supported in part by CHIMES, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program.

[2]Note that this does not account for any worsened yield of a larger chiplet. A more complete system yield model can be found, for example, in [10] albeit with an oversimplified bonding yield model.

[3]The code of the yield model and simulator is available open-source at `https://github.com/nanocad-lab/YAP`.

REFERENCES

[1] Elsherbini, A., Jun, K., Vreeland, R., Brezinski, W., Niazi, H., Shi, Y., Yu, Q., Qian, Z., Xu, J., Liff, S. & Others Enabling hybrid bonding on Intel process. *2021 IEEE International Electron Devices Meeting (IEDM)*. pp. 34-3 (2021)

[2] Ikegami, Y., Onodera, T., Chiyozono, M., Sakamoto, A., Shimizu, K., Kagawa, Y. & Iwamoto, H. Study of Ultra-Fine 0.4 μm Pitch Wafer-to-Wafer Hybrid Bonding and Impact of Bonding Misalignment. *2024 IEEE 74th Electronic Components And Technology Conference (ECTC)*. pp. 299-304 (2024)

[3] Lau, J. Recent Advances and Trends in Cu–Cu Hybrid Bonding. *IEEE Transactions On Components, Packaging And Manufacturing Technology*. **13**, 399-425 (2023)

[4] Lee, S., Jee, Y., Park, S., Lee, S., Hwang, B., Jo, G., Lee, C., Park, J., Jang, A., Jung, H. & Others A study on memory stack process by hybrid copper bonding (HCB) technology. *2022 IEEE 72nd Electronic Components And Technology Conference (ECTC)*. pp. 1085-1089 (2022)

[5] Lujan, A. Cost and Yield Analysis of Die-to-Wafer Hybrid Bonding. *2022 International Conference On Electronics Packaging (ICEP)*. pp. 129-130 (2022)

[6] Marinissen, E., Pancholi, V., Chuang, P. & Keim, M. IEEE Std P3405: New Standard-under-Development for Chiplet Interconnect Test and Repair. *2024 IEEE 42nd VLSI Test Symposium (VTS)*. pp. 1-11 (2024)

[7] Singh, E. Analytical modeling of 3D stacked IC yield from wafer to wafer stacking with radial defect clustering. *2014 27th International Conference On VLSI Design*. pp. 26-31 (2014)

[8] Xu, Q., Jiang, L., Li, H. & Eklow, B. Yield enhancement for 3D-stacked ICs: Recent advances and challenges. *17th Asia And South Pacific Design Automation Conference*. pp. 731-737 (2012)

[9] Campbell, D. Yield modeling of 3D integrated wafer scale assemblies. *2010 Proceedings 60th Electronic Components And Technology Conference (ECTC)*. pp. 1935-1938 (2010)

[10] Graening, A., Pal, S. & Gupta, P. Chiplets: How small is too small?. *2023 60th ACM/IEEE Design Automation Conference (DAC)*. pp. 1-6 (2023)

[11] Zhang, B., Chew, S., Stucchi, M., Dewilde, S., Iacovo, S., Witters, L., Webers, T., Van Sever, K., De Vos, J., Miller, A., Beyer, G. & Beyne, E. Scaling Cu/SiCN Wafer-to-Wafer Hybrid Bonding down to 400 nm interconnect pitch. *2024 IEEE 74th Electronic Components And Technology Conference (ECTC)*. pp. 312-318 (2024)

[12] Moreau, S., Bouchu, D., Jourdon, J., Ayoub, B., Lhostis, S., Frémont, H. & Lamontagne, P. Recent Advances on Electromigration in Cu/SiO_2 to Cu/SiO_2 Hybrid Bonds for 3D Integrated Circuits. *2023 IEEE International Reliability Physics Symposium (IRPS)*. pp. 1-7 (2023)

[13] Ghaida, R., Gupta, M. & Gupta, P. Framework for exploring the interaction between design rules and overlay control. *Journal Of Micro/Nanolithography, MEMS, And MOEMS*. **12**, 033014-033014 (2013)

[14] Armitage Jr, J. & Kirk, J. Analysis of overlay distortion patterns. *Integrated Circuit Metrology, Inspection, And Process Control II*. **921** pp. 207-223 (1988)

[15] Okudur, O., Iacovo, S., Kang, S., Gonzalez, M. & Beyne, E. Simulations of Wafer-to-Wafer Bonding Dynamics and Deformation Mechanisms. *2024 IEEE 10th ESTC*. pp. 1-5 (2024)

[16] Kang, S., Iacovo, S., D'havé, K., Van Huylenbroeck, S., Okudur, O., Alexeev, A., Plach, T., Probst, G., Ding, T., Wimplinger, M. & Others Investigation of Distortion in Wafer-to-wafer Bonding with Highly Bowed Wafers. *2024 IEEE 74th Electronic Components And Technology Conference (ECTC)*. pp. 386-393 (2024)

[17] Ren, H. Process Development and Process Window Investigation of Copper-Silicon Dioxide Die-to-Wafer (D2W) Hybrid Bonding. (University of California, Los Angeles, 2021)

[18] Chidambaram, V., Leong, Y. & Ren, Q. Wafer Level Fine-Pitch Hybrid Bonding: Challenges and Remedies. *2020 IEEE 22nd Electronics Packaging Technology Conference (EPTC)*. pp. 459-463 (2020)

[19] Gui, C., Elwenspoek, M., Tas, N. & Gardeniers, J. The effect of surface roughness on direct wafer bonding. *Journal Of Applied Physics*. **85**, 7448-7454 (1999)

[20] Dubey, V., Wünsch, D., Gottfried, K., Fischer, T., Helke, C., Hasse, M., Hanisch, A., Hofmann, L., Reuter, D., Wiemer, M. & Others Impact of Dielectric Types on Surface Topography for Wafer-Level Hybrid Bonding. *2024 IEEE 10th Electronics System-Integration Technology Conference (ESTC)*. pp. 1-5 (2024)

[21] Ji, L., Che, F., Ji, H., Li, H. & Kawano, M. Modelling and characterization on wafer to wafer hybrid bonding technology for 3D IC packaging. *2019 IEEE 21st EPTC*. pp. 87-94 (2019)

[22] Ji, L., Che, F., Ji, H., Li, H. & Kawano, M. Wafer-to-wafer hybrid bonding development by advanced finite element modeling for 3-D IC packages. *IEEE Transactions On Components, Packaging And Manufacturing Technology*. **10**, 2106-2117 (2020)

[23] Wang, H., Chen, H., Xiang, J. & Yang, X. Research on simulation of Cu/SiO_2 hybrid bonding process and interface failure mechanism by Finite Element Analysis. *2023 24th International Conference On Electronic Packaging Technology (ICEPT)*. pp. 1-7 (2023)

[24] Beilliard, Y., Estevez, R., Parry, G., McGarry, P., Di Cioccio, L. & Coudrain, P. Thermomechanical finite element modeling of Cu–SiO_2 direct hybrid bonding with a dishing effect on Cu surfaces. *International Journal Of Solids And Structures*. **117** pp. 208-220 (2017)

[25] Fujii, N., Furuse, S., Yoshioka, H., Ogawa, N., Yamada, T., Hirano, T., Saito, S., Hagimoto, Y. & Iwamoto, H. Bonding Strength of Cu-Cu Hybrid Bonding for 3D Integration Process. *ECS Transactions*. **112**, 3 (2023)

[26] Le, X. & Choa, S. Assessment of the Risk of Crack Formation at a Hybrid Bonding Interface Using Numerical Analysis. *Micromachines*. **15**, 1332 (2024)

[27] Zhao, G., Zeng, Y. & Zhao, Y. Simulation and Experimental Analysis of Thermomechanical Stress Around Interconnects for W2W Hybrid Bonding. *2024 IEEE 10th ESTC*. pp. 1-6 (2024)

[28] Kim, S., Fodor, F., Heylen, N., Iacovo, S., De Vos, J., Miller, A., Beyer, G. & Beyne, E. Novel Cu/SiCN surface topography control for 1 μm pitch hybrid wafer-to-wafer bonding. *2020 IEEE 70th Electronic Components And Technology Conference (ECTC)*. pp. 216-222 (2020)

[29] Beyne, E., Kim, S., Peng, L., Heylen, N., De Messemaeker, J., Okudur, O., Phommahaxay, A., Kim, T., Stucchi, M., Velenis, D. & Others Scalable, sub 2μm pitch, Cu/SiCN to Cu/SiCN hybrid wafer-to-wafer bonding technology. *2017 IEEE International Electron Devices Meeting (IEDM)*. pp. 32-4 (2017)

[30] De Messemaeker, J., Witters, L., Zhang, B., Tsau, Y., Fodor, F., De Vos, J., Beyer, G., Croes, K. & Beyne, E. New Cu "Bulge-Out" Mechanism Supporting SubMicron Scaling of Hybrid Wafer-to-Wafer Bonding. *2023 IEEE 73rd Electronic Components And Technology Conference (ECTC)*. pp. 109-113 (2023)

[31] Lin, H., Tran, D., Chiu, W., Chang, H. & Chen, C. In-situ measurement of thermal expansion in Cu/SiO_2 hybrid structures using atomic force microscopy at elevated temperatures. *Applied Surface Science*. **662** pp. 160103 (2024)

[32] Lin, H., Chiu, W., Chang, H. & Chen, C. Observation of Thermal Expansion Behavior of Nanotwinned-Cu/SiO_2 & Regular-Cu/SiO_2 Hybrid Structure via In-Situ Heating AFM. *2024 IEEE 74th Electronic Components And Technology Conference (ECTC)*. pp. 816-820 (2024)

[33] Maugis, D. On the contact and adhesion of rough surfaces. *Journal Of Adhesion Science And Technology*. **10**, 161-175 (1996)

[34] Rieutord, F., Moriceau, H., Beneyton, R., Capello, L., Morales, C. & Charvet, A. Rough surface adhesion mechanisms for wafer bonding. *ECS Transactions*. **3**, 205 (2006)

[35] Hutchinson, J. & Suo, Z. Mixed mode cracking in layered materials. *Advances In Applied Mechanics*. **29** pp. 143-145 (1991)

[36] Nagano, F., Inoue, F., Phommahaxay, A., Peng, L., Chancerel, F., Naser, H., Beyer, G., Uedono, A., Beyne, E., De Gendt, S. & Others Origin of Voids at the SiO_2/SiO_2 and SiCN/SiCN Bonding Interface Using Positron Annihilation Spectroscopy and Electron Spin Resonance. *Ecs Journal Of Solid State Science And Technology*. **12** (2023)

[37] Kim, Y., Nguyen, T. & Choa, S. Enhancement of the bond strength and reduction of wafer edge voids in hybrid bonding. *Micromachines*. **13**, 537 (2022)

[38] Nagano, F., Iacovo, S., Phommahaxay, A., Inoue, F., Chancerel, F., Naser, H., Beyer, G., Beyne, E. & Gendt, S. Void formation mechanism related to particles during wafer-to-wafer direct bonding. *ECS Journal Of Solid State Science And Technology*. **11**, 063012 (2022)

[39] Glang, R. Defect size distribution in VLSI chips. *IEEE Transactions On Semiconductor Manufacturing*. **4**, 265-269 (1991)

[40] Stapper, C. Modeling of defects in integrated circuit photolithographic patterns. *IBM Journal Of Research And Development*. **28**, 461-475 (1984)

[41] Bruls, E. & Others Characterization of defects in integrated circuits: resources, models and applications. (1992)

[42] Lu, T., Hsu, K., Hsu, C., Hsu, C. & Wu, Y. Effect of Cu Film Thickness on Cu Bonding Quality and Bonding Mechanism. *Materials*. **17**, 2150 (2024)

[43] Koren, I. & Koren, Z. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proceedings Of The IEEE*. **86**, 1819-1838 (1998)

[44] Chidambaram, V., Lianto, P., Wang, X., See, G., Wiswell, N. & Kawano, M. Dielectric materials characterization for hybrid bonding. *2021 IEEE 71st ECTC*. pp. 426-431 (2021)

[45] Dubey, V., Wünsch, D., Gottfried, K., Wiemer, M., Fischer, T., Schermer, S., Dittmar, N., Helke, C., Haase, M., Ghosal, S., Hanisch, A., Bonitz, J., Luo-Hofmann, J., Hofmann, L., Lykova, M., Stoll, F., Vogel, K. & Schulz, S. Impact of Dielectric and Copper Via Design on Wafer-to-Wafer Hybrid Bonding. *2023 IEEE 73rd ECTC*. pp. 795-799 (2023)

[46] Mitsuishi, H., Mori, H., Maeda, H., Ushijima, M., Kamashita, A., Okada, M., Aramata, M., Shiomi, T., Sakamoto, S., Takahata, K. & Others 50 nm overlay accuracy for wafer-to-wafer bonding by high-precision alignment technologies. *2023 IEEE 73rd Electronic Components And Technology Conference (ECTC)*. pp. 1664-1671 (2023)

[47] Patel, D., Xie, M., & Koch, J. Hybrid Bonding Process Flow – Advanced Packaging Part 5. https://semianalysis.com/2024/02/09/hybrid-bonding-process-flow-advanced/ (2024)

SuperCopyback: Revisiting Copyback on Modern High-Performance NAND Flash-based SSDs

Dong Huang, Bo Ding, Wei Tong✉*, Dan Feng*✉
Key Laboratory of Information Storage System, Engineering Research Center for Data System and Technology, Ministry of Education
Wuhan National Laborotory of Optoelectronics, Huazhong University of Science and Technology
{*haltz,boding,tongwei,dfeng*}*@hust.edu.cn,* ✉*Corresponding Authors*

***Abstract*—NAND flash-based SSDs have emerged as a critical storage solution due to their exceptional performance and cost-effectiveness. However, the sequential write limitation of NAND flash blocks necessitates garbage collection (GC) to reclaim space occupied by stale data. Nevertheless, the extensive data migration involved in GC significantly impacts performance and Quality of Service (QoS) of SSDs. To mitigate this issue, copyback has been proposed as a means to accelerate GC by eliminating off-chip data movements. Specifically, copyback reads data into on-plane latches and immidiately re-writes it into another page on the same plane.**

However, in the case of modern high-performance SSDs, copyback is rarely utilized due to the following challenges: (1) Copyback operates at the page-level and thus fails to effectively reclaim invalid data within the context of subpage-level mapping; (2) Copyback eliminates off-chip data movements, preventing data pages from being read out for Redundant Array of Independent NAND (RAIN) parity computation, thereby compromising SSD reliability. In this study, we introduce SuperCopyback as a solution that efficiently addresses these issues for modern SSDs. Firstly, we propose a Multiple-Read-One-Write (MROW) copyback approach through lightweight latch circuit modifications to enable subpage-level copyback implementation; additionally, we propose an orchestrated GC method to effectively utilize MROW copyback. Furthermore, we present a novel copyback-based RAIN scheme that conceals data pages readout latency in write operations and relocates parities to support efficient copyback. The experimental results on both synthetic and real traces demonstrate that SuperCopyback achieves a performance comparable to an ideal scenario where data movement of 4KB takes only 1ns.

***Index Terms*—NAND Flash Memory, Solid State Disk, Copyback.**

I. Introduction

NAND flash-based Solid-State Disks (SSDs) have emerged as the predominant storage medium for many applications due to their significantly superior performance compared to hard disks, primarily attributed to the inherent abundant parallelism [1]–[4]. Typically, an SSD comprises multiple channels, each of which is shared by multiple chips. Each chip consists of multi-planes and multi-plane commands can be used for simultaneous read/write operations on all planes [19]–[23]. Multiple blocks from different chips are organized into a superblock, with pages at identical offsets from these blocks grouped as superpages. The utilization of superpage-based management not only maximizes the exploitation of abundant parallelism and simplifies data management, but also facilitates the cross-chip Redundant Array of Independent NAND (RAIN) stripe organization, which significantly enhances the reliability of SSDs [5]–[8].

Due to the inherent sequential write constraint of flash blocks, Garbage Collection (GC) is essential for reclaiming space occupied by invalid data in order to accommodate incoming writes [9]–[11]. Specifically, GC entails the selection of a candidate block, migration of valid data to another available free block, and subsequent erasure of the candidate block. The performance, endurance, and Quality of Service (QoS) of SSDs are significantly influenced by GC due to its extensive data migration.

Copyback is an effective strategy for accelerating garbage collection (GC) by eliminating off-chip data movements through channels. Specifically, copyback reads data to on-plane data latches (i.e., page register) and then immidiately re-writes it to another page on the same plane. The effectiveness of copybacks have been validated by previous research [9], [10], [12]. This optimization significantly enhances GC performance, particularly in scenarios where a channel is shared by multiple high-performance NAND chips.

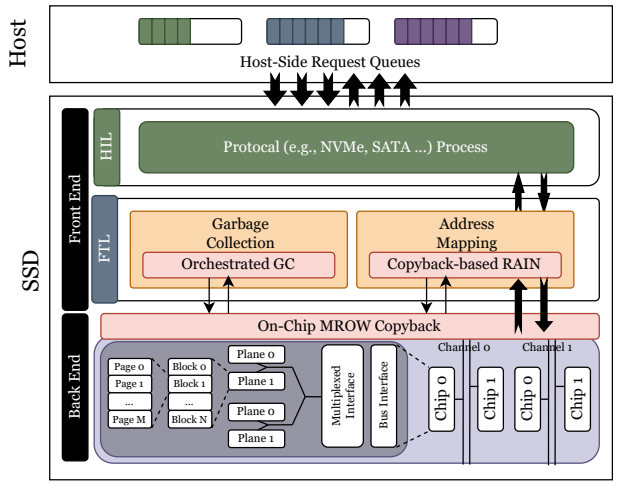


Fig. 1: Overview of SSD Archiecture and SuperCopyback

However, modern SSDs present notable challenges when employing copyback. ***Challenge #1 partial-valid page copyback:*** In the context of modern SSDs, pages typically possess a larger size (e.g., 16KB) compared to the host's I/O granularity of 4KB. To mitigate the resulting write amplification caused by this mismatch, SSDs employ subpage-level mapping, resulting in partial-validity within a page where certain subpages are valid while others remain invalid [13], [14]. As copyback operations occur at the page level, SSDs exclusively programme invalid data or resort to normal GC. ***Challenge #2 Parity Computation:*** Modern SSDs employ RAIN as a means to ensure data reliability in cases where ECC fails to correct errors [15], [16] or die-level failures occur [6], [8], thereby mitigating potential risks. Since the parity write operation requires XOR computation on other data pages in the stripe, the RAIN scheme cannot be used during copyback due to the elimination of off-chip data movements. This dilemma presents a tradeoff between performance and reliability.

In this paper, we propose ***SuperCopyback*** to tackle the challenges. As shown in Figure 1, SuperCopyback includes an efficient subpage-level copyback called ***Multiple-Read-One-Write (MROW) copyback***. MROW copyback is implemented via lightweight hardware modifications (i.e., a few transistors used to act as switches). The core of MROW is that multiple partial-valid pages can be merged into a full-valid page on the chip. We also propose an ***orchestrated GC scheme*** implmented in GC unit to use MROW copyback efficiently. Furthermore, SuperCopyback includes a novel ***copyback-based RAIN scheme*** implemented in address mapping unit. Copyback-based RAIN scheme relocates parities to avoid copyback stalls and hide the data readout latency in write operations. Our experimental results on both synthetic and real traces demonstrate that compared to an SSD with GC using off-chip data movements, SuperCopyback can achieve up to 1.346x throughput (1.263x on average), meanwhile significantly enhancing QoS by reducing 28.4% 99-th percentile latency on average, which is comparable to an ideal

scenario where data movement of 4KB takes only 1ns. In summary, we make the following contributions in this paper:

- We highlight the challenges associated with implementing copyback operations in modern SSDs, including limitations in the context of subpage-level mapping and potential conflicts with RAIN scheme.
- We propose SuperCopyback, which introduces a subpage-level copyback operation called MROW copyback, implemented through lightweight hardware modifications. Additionally, we present an efficient approach for utilizing MROW copybacks. Furthermore, we propose a novel copyback-based RAIN scheme to enable RAIN functionality within the context of MROW copybacks.
- We conducted comprehensive experiments to demonstrate the advantages of SuperCopyback. The experimental results reveal that SuperCopyback increases up to 34.6% throughput and reduces 28.4% 99-th percentile latency on average, compared to an SSD with GC using off-chip data movements.

II. Background

A. SSD Overview

Typically, an SSD is structured into three layers, as depicted in Figure 1 [1], [17]. The front-end encompasses host-interface logic (HIL) and flash translation layer (FTL). HIL receives and responds to requests from the host based on protocols (e.g., NVMe, SATA). FTL operates on a microprocessor and processes I/O requests while sending/receiving flash operations to/from the backend. FTL serves as the core of SSD firmware and is responsible for address mapping, garbage collection, wear leveling, etc. The backend consists of a hierarchically organized NAND flash array with multiple channels which transfer data between the front-end and back-end. Each channel is connected to multiple flash chips, each composed of multiple planes. A plane comprises numerous NAND blocks, with each block containing many pages. Page is the minimum read and program unit while block is the minimum unit of erase. Pages in a block must be sequentially programmed, causing GC problem.

B. GC and Copyback

Due to the sequential programming nature of NAND blocks, FTL typically employs a method of data writes by appending data to the block and recording the page location in a mapping table. Stale data pages are marked as invalid, and to reclaim these pages, FTL performs GC that involves migrating valid data from one superblock (referred to as victim) to another superblock before erasing the victim superblock. A physical page not only carries data but also metadata including Logical Page Address (LPA) and ECC (Error Correction Code) parity. During GC, the controller reads both the data and metadata, corrects errors using ECC parity, allocates a new address to LPA and updates the mapping table; finally, it writes the corrected data into NAND flash memory.

The impact of GC on I/O performance is widely acknowledged [9]–[11], [18], prompting NAND flash vendors to propose the use of copyback as a means to accelerate GC. Copyback involves reading a page to the plane register and immediately re-writing it on the same plane, thereby eliminating off-chip data movement through channels, as shown in Figure 2. While copyback holds significant potential, its current applicability in modern SSDs is limited, as discussed in Section III.

C. Characteristics of Modern NAND Flash-based SSD

Modern high-performance NAND Flash-based SSDs typically exhibit abundant parallelism through the utilization of multiple channels, chips, and planes. In order to fully harness this parallelism, SSDs commonly employ superpage-based data management. This approach involves grouping blocks at identical offsets from each plane to form a superblock, and then further grouping pages at the identical offset from each block to form a superpage. Allocation of pages from superpages is carried out in a round-robin manner. Superpage-based management can maximize the utilization of parallelism while also simplifying data management [5]–[8]. To fully exploit plane-level parallelism, independent multi-plane read is proposed to read multiple pages simultaneously from different offsets on planes [19]–[21]; for writes, buffer-based multi-plane write is proposed [22], [23]. In this paper, these approaches are also used to accelerate SSD performance.

Furthermore, superpage also facilitates the generation of RAIN parity, a technique commonly employed to enhance SSD reliability [6], [8], [24], [25]. Similar to RAID, RAIN organizes data pages into stripes and calculates a parity for each stripe using XOR. In the event of data page corruption due to ECC failure or chip-level failure, RAIN reads all other pages in the stripe and performs an XOR operation to recover the corrupted data page. A single superpage can be divided into one or multiple stripes. As flash block does not allow in-place updates, once generated, the parity is never updated, bringing in minimal overhead. Currently, enterprise-level SSDs typically employ a RAID-5 style RAIN to achieve full read performance [8], [24], [25], which is also used in this paper.

In addition, modern SSDs typically use 16KB pages, representing a fourfold increase from the 4KB pages employed for host I/O requests. This disparity results in significant write amplification and performance degradation. To address this issue, state-of-the-art high-performance SSDs implement subpage-level mapping [13], [14], [26]. Subpages are aggregated in the buffer and subsequently written to NAND memory once they can form a complete physical page. In our study, we adopt 4KB subpages while maintaining physical pages at 16KB.

III. Motivation

Modern solid-state drives (SSDs) commonly employ subpage-level mapping and superpage-based management techniques to optimize performance and ensure reliability. However, incorporating copyback into such SSDs presents unique challenges. In this section, we discuss these challenges and review the existing works.

A. Challenges

As shown in Figure 2, the first challenge arises from the limitation that conventional copyback operations can only be performed at a page level and cannot handle partial-valid pages effectively. While recent advancements like SOML enable reading of subpages across different pages within a plane, they are constrained by block decoder limitations preventing target locations from sharing same block [27]. As a result, these techniques cannot be utilized for GC reads. If invalid subpages are skipped, this would result in accelerated depletion of available storage space, leading to increased write amplification and diminished overall performance and lifespan of SSDs.

Second, as RAIN parity necessitates a XOR operation on all data pages within a cross-chip stripe, the use of copyback inevitably compromises reliability since it eliminates off-chip data transfer. Consequently, a trade-off between performance and reliability arises with the coexistence of either copyback or RAIN.

B. Existing Works

Copyback can accelerates data movement during GC. However, traditional copyback methods are unable to rectify bit errors. To address this issue, FastGC [10] and rcFTL [9] proposed a threshold-based copyback scheme to prevent uncorrectable errors. This strategy is also employed in this paper. The limitations in their design is

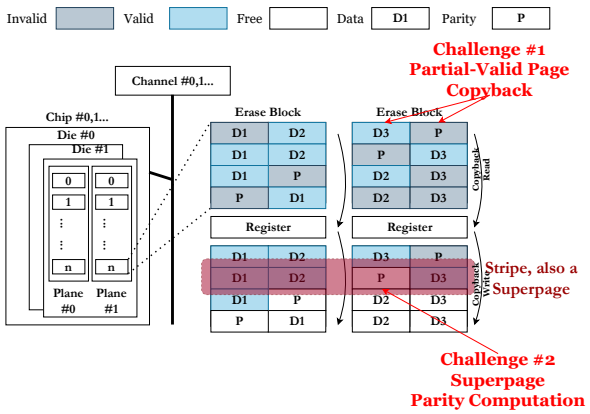


Fig. 2: The challenges in superpage-based SSDs.

that they are for conventional SSDs without superpage management, subpage-level mapping and RAIN protection, making them inadequate to address the aforementioned challenges. DecoupledSSD [18] interconnects channels to mitigate DRAM bus contention, although it still necessitates off-chip data movements during GC. Other interconnection-based approaches [28], [29] address channel contention by enhancing path diversity but at the expense of substantial architectural and hardware modifications. Besides, these approaches are based on conventional SSDs and therefore cannot effectively address the unique challenges posed by superpage-enabled SSDs. For instance, in superpage-enabled SSDs, garbage collection is executed simultaneously across all chips, which presents a significant challenge for path reservation in the interconnect network. Additionally, due to the absence of an intermediate buffer, the target chip's register must be idle to accept data; otherwise, the source chip can only wait, resulting in a sacrifice of chip-level parallelism. Furthermore, even if a dedicated path is always available, there still exists latency in data transfer through the links. These difficulties make using interconnect network during GC for superpage-enabled SSDs challenging. Hence, we focus on using copyback to accelerate GC from another perspective.

To the best of our knowledge, SuperCopyback represents a pioneering effort in enabling efficient copyback on modern SSDs to mitigate the detrimental impact of off-chip data movements during GC. We believe that this makes SuperCopyback novel and a significant contribution to the field.

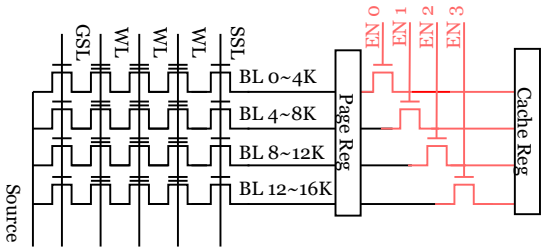
IV. Design

A. Overview

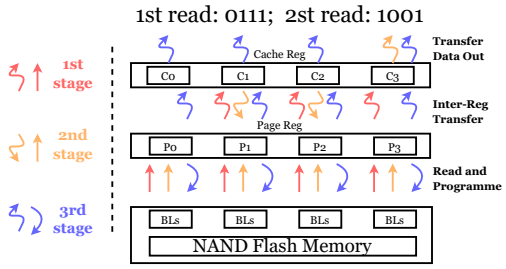
SuperCopyback enables efficient copyback on modern SSDs, as shown in Figure 1. First, it implments on-chip MROW copyback by lightweight hardware modifications. MROW copyback supports not only full-valid page copyback but also partial-valid page copyback. Then, an orchestrated GC is implemented in GC unit to exploit MROW copyback to accelerate GC. In the end, to ensure sufficient reliability when using copyback, a copyback-based RAIN scheme is implemented in the address mapping unit, which hides the data readout latency in program operations during MROW copyback and relocates parity pages to avoid copyback stalls.

B. Subpage Copyback

We propose a subpage-level copyback called Multiple-Read-One-Write (MROW) copyback. The core idea of MROW is that it first uses independent multi-plane read to read pages from all planes to the plane registers. Then, MROW performs a second read for each



(a) Subpage-level inter-register transfer.



(b) Steps of a MROW copyback.

Fig. 3: Implementation of MROW copyback.

plane to read another page to replace the invalid subpages in the first read page.

Two-Latches Meachanism: To enable on-chip merge of read data, we implement a lightweight subpage-level inter-register transfer. We exploit the existing two-latches mechanism in flash chip [3], [30]. Conventionally, a bit-line is connected to a data latch; and a data latch is connected to a cache latch. In a read operation, data bits are sensed through bitlines to data latches; and then transferred to cache latch if cache latches are empty. If possible, channel controller transfers data out from cache latches; meanwhile, chip can still read bits into data latches, accelerating reads through a pipelined style. The writes can also be accelerated in a similar style.

Subpage-level Inter-Register Transfer: To implement subpage-level inter-register transfer, we divide the bitlines, data latches and cache latches in a plane into multiple groups at subpage granularity, as shown in Figure 3a. For each group, we add a transistor and connect it to the *EN* inputs of all cache latches. Then, for target groups, we enable the transistor so the cache latches can store bits from data latches; for other groups, we disable the transistor so the corresponding cache latches can hold the original bits. Similarly, another four transistors can be connected to data latches to enable a symmetric cache-to-data subpage-level inter-register transfer. Since a latch typically only takes a few cycles to store a bit, the latency overhead is negligible. With subpage-level inter-register data transfer, we can read two partial-valid pages, merge the valid subpages into one full-valid page and programme it into NAND cells. If two pages have valid subpages at the same offset (we call it conflict subpages), we read out the one in the first read and write it as the normal GC such as the read out latency can be hided by the second read. MROW can also support full-valid page read by eliminating the first read.

Metadata Readout: To update the mapping table, the metadata of pages must be transferred out. Fortunately, we observe that page program latency is significantly higher than the data transfer latency. Hence, hiding data transfer latency during program operations is possible. Specifically, the chip awaits for the end of reads in copyback

(two reads for partial-valid pages or one read for full-valid pages) and the end of conflict subpages readout (see the example below for details); then, the controller sends copyback write command to the chip; simultaneously, the chip informs the controller that the data on cache register is ready for readout.

Command Support: The existing copyback commands can be leveraged to support the MROW copyback by adding four bits to each plane address in the command. These bits are used to indicate the validity of subpages. Hence, the inter-register transfer can be properly enabled to implement MROW copyback.

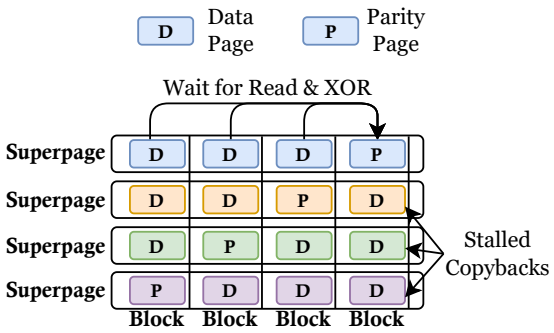
Illustrative Example: Figure 3b illustrates an example, where a page has four subpages. Assume the first read page has three valid subpages (the last three ones, tagged as 0111) and the second read page has two valid subpages (the first and last one, tagged as 1001). The page register and cache register are divided into four groups (P0 P4, C0 C4). In the first stage, the controller sends the first copyback read command to read the first page into page register same as a normal read operation; when read ends, the last three valid subpages are transferred from page register (P1-P3) to cache register (C1-C3). In the second stage, the controller sends the second copyback read command and readout data from C3 if possible. Once read operation ends, valid data are transferred from C1, C2 to P1, P2 to form a full-valid page. In the third stage, all data are copied from page register to cache register; then the data in page register are programmed into NAND cells and the data in cache register are transferred out when channel becomes idle. Notice that for a full-valid page, the first read is eliminated and MROW can be simplified by starting from the second stage.

C. Orchestrated GC Based on Subpage Copyback

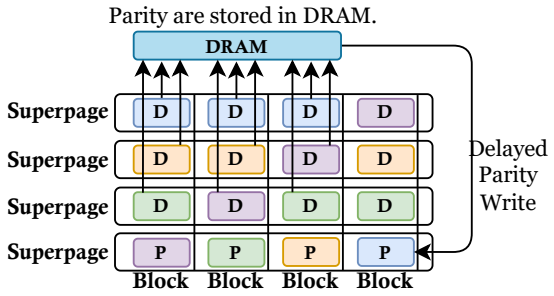
We employ a greedy algorithm to orchestrate copybacks in GC, aiming for the fewest readouts in the second stage. It is important to note that the copyback write operation must wait for the completion of readouts in the second stage. Given that read latency is comparable to data transfer latency, it is not always possible for read operations to effectively conceal data transfer latency. This motivates us to seek ways to minimize readout occurrences in the second stage of MROW copyback.

In order to adhere to the maximum number of successive copybacks, we record the executed copyback counts as page metadata and implement a threshold-based copyback scheme, similar to previous studies. Put simply, if the copyback counts exceed the pre-defined threshold, the subpage is migrated via conventional GC. Additionally, we use one bit in DRAM to record the copyback feasibility of each subpage (1 indicating that this subpage can be migrated through copyback), allowing for indication of the subpages' copyback feasibility in a page with a 4-bits bitmap. As the copyback count metadata is read during GC, the copyback feasibility bit can be updated by comparing it with the copyback counts against the threshold. Prior to performing GC, all valid pages of a plane are grouped into 16 lists corresponding to various 4-bits bitmap values.

For the target chip, we select either one full-valid page or two partial-valid pages on each plane to execute MROW copybacks. To begin, we iterate through the list starting with a bitmap value of *1111* (representing full-valid pages in copyback). If this is not feasible, we search for viable matches from the list *0001* to *1110*. Specifically, for a list with a bitmap of *abcd*, we can identify its corresponding optimal matches (i.e., no readout required in the second stage) within the list *1111-abcd*. In case the list *1111-abcd* is empty, we explore other lists with complementary bitmaps *a'b'c'd'* (i.e., where *a'b'c'd' or abcd = 1111*). If such a match cannot be found, normal GC procedures are employed as fallback.



(a) Conventional RAIN layout based on RAID-5.



(b) Copyback-based RAIN layout.

Fig. 4: Comparison between conventional RAIN and copyback-based RAIN.

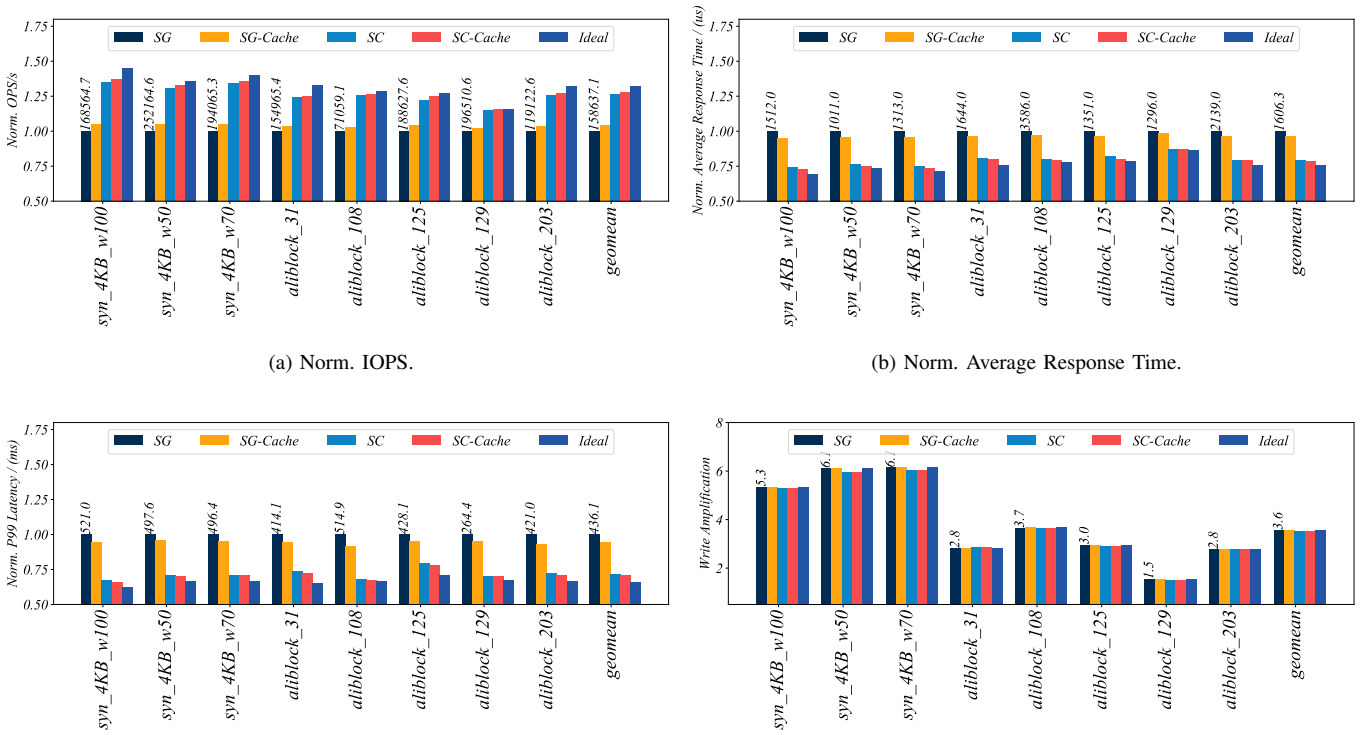
In addition, to maintain superpage management after GC operations, we impose an upper limit on the maximum number of copybacks per chip, which is $\frac{N_{validsubpages}}{Size_{superpage}}$. Once this limit is reached, GC on that chip reverts back to normal mode. Such a limit can also prevent potential delays in GC due to an imbalance in the distribution of valid subpages across different chips. While rare, this scenario remains a possibility and warrants consideration.

The subpage in the second read, if valid but not feasible for copyback, may overwrite the corresponding subpage in the first read designated for copyback. To prevent such exceptional cases, we ensure that the copyback feasibility bitmap matches the validity bitmap (i.e., all valid subpages in the second read are feasible for copyback). Additionally, in order to fully exploit plane-level parallelism through multi-plane operations, if no feasible match is identified on any plane, the corresponding chip also reverts to normal GC. It is worth noting that independent multi-plane read, a prevalent technique in modern NAND chips [19], [20], [31], is used to eliminate the identical page offset requirement in a multi-plane read. This enables full utilization of plane-level parallelism during copyback reads and eliminates the need to align read pages on all planes during MROW copyback. Multi-plane program during copyback is also used as conventional superpage-enabled SSDs.

D. Parity Computation and Storage

In this section, we elaborate on the implementation of efficient RAIN schemem in copyback-based GC.

The computation of parity, which involves XOR computation on all data pages in the stripe, necessitates the controller to read both metadata and data during copybacks. We can observe that the latency associated with these transfers is considerably lower than



(a) Norm. IOPS.

(b) Norm. Average Response Time.

(c) Norm. 99-th Percentile Latency.

(d) Write Amplification.

Fig. 5: Performance Comparison between Different SSDs. The results are normalized to SG.

program latency. To mitigate the impact of data transfer overhead, we strategically execute it during write operations. For a 1.6GB/s channel, transferring 64KB data consumes 40μs while program latency typically is higher than 300μs even for modern high-performance flash chips. Hence, we can effectively conceal this transfer latency by performing data transfers concurrently with program operations. As illustrated in Figure 3b, during the 3rd stage of MROW copyback, not only do we transfer page metadata but also the entire page data.

Furthermore, we propose an optimized RAIN layout to mitigate copyback stalls. In the conventional RAID-5 based RAIN layout, both data and parity are stored within the same superpage (refer to Figure 4 where different colors mean different stripes). However, during GC, the process of writing parity is delayed due to the necessity of reading and computing other data pages using XOR operation in the stripe. The subsequent data copyback are stalled since NAND blocks allow only sequential page writes. To address this challenge, our proposed approach relocates all parity pages to the end of NAND blocks.

As depicted in Figure 4, similar to conventional SSDs, data pages are read into DRAM and controller uses XOR to compute parity. However, instead of immediately writing the parity, we temporarily store it in DRAM until all data pages have been written. This strategy ensures that copybacks are not stalled by parity writes, enabling uninterrupted high-speed copyback operations. Notice that RAIN layout remains fixed, which means that once a page is corrupted, the controller can find other pages in its stripe based on the pre-defined layout. Besides, the parity page still resides in the same plane as original (i.e., only its page offset is changed). This eliminates needs for additional mapping and ensures recovery from page-level or chip-level failures as fast as original RAIN scheme.

E. Overhead

The SuperCopyback technique introduces a acceptable overhead. In the case of MROW, this overhead primarily arises from the modifications made to the control circuits of the data and cache latches, involving only a few transistors as a switch. Considering that each plane register comprises hundreds of thousands of data/cache latches, each composed of tens of transistors, the additional overhead is negligible.

In the case of orchestrated garbage collection (GC), the primary source of overhead arises from the DRAM utilization for an additional copyback feasibility bitmap. Considering that each subpage (4KB in this study) only requires one bit, the total memory overhead for a 1-TB SSD amounts to 32MB. It is worth noting that this bitmap can be stored at the end of the superblock and loaded into DRAM when this particular superblock is chosen as the victim for GC, similar to how a bitmap indicating page validity is stored in usual SSDs [13]. Consequently, if implemented in such a manner, the memory overhead becomes negligible (e.g., 64KB for a 2GB superblock). The main overhead of the proposed RAIN scheme arises from the buffer used for temporarily storing delayed parity writes. In our configuration, the memory overhead amounts to 64 MB ($\frac{2GB(superblocksize)}{32(stripesize)}$). For larger stripes, the memory consumption is reduced.

Typically, a 1-TB high-performance SSD (especially enterprise-level SSD) contains more than 1-GB DRAM, making the overall DRAM overhead acceptable from our perspective. It is worth noting that the consumed DRAM can also be substituted with cost-effective memory options such as PCM and SLC NAND or host memory through HMB technology to enhance cost-effectiveness. This substitution does not impact performance since PCM or SLC NAND still offer significantly faster speeds compared to TLC NAND, and parity/bitmap writes only constitute a minimal proportion of total I/O.

TABLE I: SSD Parameters

# of channels	4	# of chips	8
# of blocks	based on trace footprint	# of planes	4
RAIN Stripe	32	# of pages	1024
Channel BW	1.6GB/s	Page/Subpage Size	16KB/4KB
Read Latency	45 μs	Program Latency	320 μs
Block Erase Latency	3 ms	Over-Provision	15%

TABLE II: Workload Statistics

Trace	Requests	Average Size(KB)	Read Ratio (%)	Footprint (GB)
syn_4KB_w100	13107200	4	0	128
syn_4KB_w70	13107200	4	30	128
syn_4KB_w50	13107200	4	50	128
aliblock_31	3959045	9.22	0.29	54.6
aliblock_108	3249302	24.37	47.21	39.8
aliblock_125	8980886	9.21	19.07	58.2
aliblock_129	4285829	16.11	21.69	99.9
aliblock_203	4722134	12.40	2.25	30.4

In addition, in the event of a power failure, the bitmaps and parity can be reloaded or recomputed, eliminating the need for capacitors or batteries in DRAM to ensure persistence.

V. Evalution

A. Experimental Environment

We use MQSim, which is a widely-used simulator [17], to model a high-performance TLC SSDs. The detailed configuration of the SSD is listed as Table I. SSD performs GC when there is only one free superblock. We adjust the device capacity based on the footprint of each trace to avoid too large over-provison space. We extend MQSim to support subpage-level mapping and RAID-5 based RAIN scheme. We compare the following schemes: (1) SG, a superpage-based SSD using normal GC. (2) SG-Cache, an SG SSD with cache read/program using on-chip cache latches, (3) SC, an SSD using SuperCopyback that the maximum number of successive copybacks is 8 for GC, (4) SC-Cache, SC SSD with cache read/write for non-copyback IO, (4) Ideal, an SSD same as SG-Cache but its 4KiB data transfer overhead during GC is 1ns, which is used to emulate the ideal scenario.

We select both synthetic and real traces to evaluate the performance benefits. The synthetic traces are generated 4KiB uniform random workloads with different write ratios (50%, 70%, 100%) with 50GB size in total on a 128GB SSD. The real traces are widely-used block traces from Alibaba Cloud [32]. The characteristics of trace are listed as Table II. We ignore the I/O timestamps to make traces intensive. To avoid too much over-provision space, we adjust the device capacity based on trace foorptints.

B. Performance

In this section, we assess the performance advantages of SuperCopyback in terms of throughput, average response time, tail latency at the 99th percentile and write amplification (Figure 5). The following observations can be made. Firstly, in comparison to SG, SC achieves a maximum of 1.346x (34.6%) higher IOPS with an average improvement of 1.263x (26.3%); it reduces the average response time by up to 25.7% with an average reduction of 20.9%; and it decreases p99 latency by up to 32.8% with an average decrease of 28.4%. These results demonstrate that SuperCopyback not only significantly enhances throughput but also improves the QoS for SSDs. Furthermore, in comparison to SG-Cache, SC-Cache also demonstrates comparable performance benefits; similarly, when compared to SC, SC-Cache exhibits marginal performance advantages. This implies that

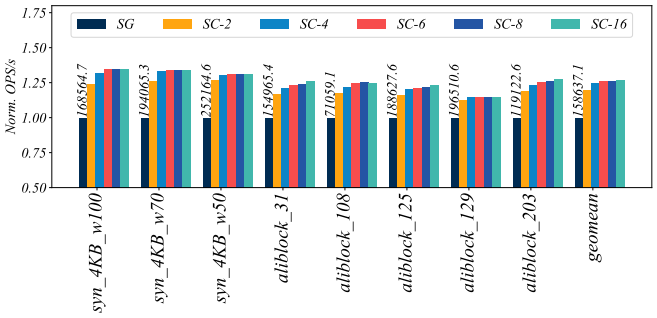


Fig. 6: The Impact of Copyback Threshold.

SuperCopyback can still derive advantages from cache read/write operations during the execution of user I/O requests; however, compared to cache read/write activities, SuperCopyback significantly enhances GC performance. Moreover, the SC-Cache achieves a performance close to Ideal (96.9% IOPS of Ideal), indicating that SuperCopyback effectively and efficiently mitigates the detrimental effect associated with data movements during GC. In the end, the write amplification of all schemes are nearly identical, indicating that SuperCopyback does not exert any adverse impact on the endurance of the SSD; the performance benefits of SuperCopyback also exhibit a positive correlation with write amplification, as higher write amplification leads to more severe GC and consequently increased data movements, thereby exacerbating its detrimental effects.

C. The Impact of Copyback Threshold

In this section, we assess the impact of varying copyback thresholds (i.e., the maximum number of consecutive copybacks for a subpage before ECC becomes unable to correct errors) on system performance. We explore copyback thresholds ranging from 2 to 16 (SC-2 to SC-16), and present the corresponding experimental results in Figure 6. The experimental results demonstrate that even with a copyback threshold of 2, SuperCopyback still exhibits significant performance benefits (1.198x IOPS). Furthermore, as the copyback threshold increases, the performance advantages of SuperCopyback also increase. When the copyback threshold reaches 6, SC achieves comparable performance to that obtained under higher copyback thresholds. This implies that in order to fully exploit SuperCopyback, it is only necessary for the ECC to ensure that at least 6 copybacks can be performed before ECC failures occur. We believe this is possible according to the results in prior works [9], [10].

VI. Conclusion

In this study, we propose SuperCopyback to mitigate the adverse impacts of off-chip data movements. SuperCopyback implements MROW copyback to support subpage-level copyback via lightweight hardware modifications; in addition, an orchestrated GC scheme is proposed to exploit MROW copyback. Furthermore, a copyback-based RAIN scheme is introduced to support efficient copyback, achieving both high reliability provided by RAIN and high performance provided by copyback. The experimental results on both real and synthetic traces demonstrate that SuperCopyback can significantly enhance the performance of modern SSDs.

VII. Acknowledgements

This work was supported by National Key R&D Program of China under Grant 2023YFB4502100 and also supported by the NSFC under Grant No.62172178, No.61821003.

References

[1] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in *Proceedings of the international conference on Supercomputing*, 2011, pp. 96–107.

[2] H. Wan, X. Sun, Y. Cui, C.-L. Yang, T.-W. Kuo, and C. J. Xue, "Flashembedding: storing embedding tables in ssd for large-scale recommender systems," in *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2021, pp. 9–16.

[3] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-cosmos: In-flash bulk bitwise operations using inherent computation capability of nand flash memory," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 937–955.

[4] X. Sun, H. Wan, Q. Li, C.-L. Yang, T.-W. Kuo, and C. J. Xue, "Rm-ssd: In-storage computing for large-scale recommendation inference," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 1056–1070.

[5] S.-H. Tseng, T.-Y. Chen, and M.-C. Yang, "Are superpages super-fast? distilling flash blocks to unify flash pages of a superpage in an ssd," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 630–642.

[6] Y. Zhou, F. Wu, W. Huang, and C. Xie, "Livessd: A low-interference raid scheme for hardware virtualized ssds," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1354–1366, 2020.

[7] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The {CASE} of {FEMU}: Cheap, accurate, scalable and extensible flash emulator," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018, pp. 83–90.

[8] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Improving 3d nand flash memory lifetime by tolerating early retention loss and process variation," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 3, pp. 1–48, 2018.

[9] D. Hong, M. Kim, J. Park, M. Jung, and J. Kim, "Improving ssd performance using adaptive restricted-copyback operations," in *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2019, pp. 1–6.

[10] F. Wu, J. Zhou, S. Wang, Y. Du, C. Yang, and C. Xie, "Fastgc: Accelerate garbage collection via an efficient copyback-based data migration in ssds," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[11] C. Ji, L. Wang, Q. Li, C. Gao, L. Shi, C.-L. Yang, and C. J. Xue, "Fair down to the device: A gc-aware fair scheduler for ssd," in *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2019, pp. 1–6.

[12] N. Shahidi, M. T. Kandemir, M. Arjomand, C. R. Das, M. Jung, and A. Sivasubramaniam, "Exploring the potentials of parallel garbage collection in ssds for enterprise storage systems," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 561–572.

[13] Y. Zhou, E. Xu, L. Zhang, K. Karkra, M. Barczak, W. Gao, W. Malikowski, M. Kozlowski, Ł. Łasek, R. Lu *et al.*, "Csal: the next-gen local disks for the cloud," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 608–623.

[14] D. Lee, D. Hong, W. Choi, and J. Kim, "Mqsim-e: An enterprise ssd simulator," *IEEE Computer Architecture Letters*, vol. 21, no. 1, pp. 13–16, 2022.

[15] Q. Li, H. Dang, Z. Wan, C. Gao, M. Ye, J. Zhang, T.-W. Kuo, and C. J. Xue, "Midas touch: Invalid-data assisted reliability and performance boost for 3d high-density flash," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 657–670.

[16] S. Wang, F. Wu, Z. Lu, Y. Zhou, Q. Xiong, M. Zhang, and C. Xie, "Lifetime adaptive ecc in nand flash page management," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1253–1556.

[17] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "{MQSim}: A framework for enabling realistic studies of modern {Multi-Queue}{SSD} devices," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018, pp. 49–66.

[18] J. Kim, M. Jung, and J. Kim, "Decoupled ssd: Rethinking ssd architecture through network-based flash controllers," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[19] D. Huang, D. Feng, Q. Liu, B. Ding, W. Zhao, X. Wei, and W. Tong, "A read latency variation aware independent read scheme for qlc ssds," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[20] N. Shibata, K. Kanda, T. Shimizu, J. Nakai, O. Nagao, N. Kobayashi, M. Miakashi, Y. Nagadomi, T. Nakano, T. Kawabe *et al.*, "A 1.33-tb 4-bit/cell 3-d flash memory on a 96-word-line-layer technology," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 178–188, 2019.

[21] B. Kim, S. Lee, B. Hah, K. Park, Y. Park, K. Jo, Y. Noh, H. Seol, H. Lee, J. Shin, S. Choi, Y. Jung, S. Ahn, Y. Park, S. Oh, M. Kim, S. Kim, H. Park, T. Lee, H. Won, M. Kim, C. Koo, Y. Choi, S. Choi, S. Park, D. Youn, J. Lim, W. Park, H. Hur, K. Kwean, H. Choi, W. Jeong, S. Chung, J. Choi, and S. Cha, "28.2 a high-performance 1tb 3b/cell 3d-nand flash with a 194mb/s write throughput on over 300 layers i," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 27–29.

[22] C. Gao, L. Shi, C. J. Xue, C. Ji, J. Yang, and Y. Zhang, "Parallel all the time: Plane level parallelism exploration for high performance ssds," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2019, pp. 172–184.

[23] C.-Y. Liu, Y. Lee, W. Choi, M. Jung, M. T. Kandemir, and C. Das, "Gssa: A resource allocation scheme customized for 3d nand ssds," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 426–439.

[24] S. Shadley, "NAND flash media management through RAIN." [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technical-marketing-brief/brief_ssd_rain.pdf

[25] R. Pletka, I. Koltsidas, N. Ioannou, S. Tomić, N. Papandreou, T. Parnell, H. Pozidis, A. Fry, and T. Fisher, "Management of next-generation nand flash to achieve enterprise-level endurance and latency targets," *ACM Transactions on Storage (TOS)*, vol. 14, no. 4, pp. 1–25, 2018.

[26] M. Kang, W. Lee, and S. Kim, "Subpage-aware solid state drive for improving lifetime and performance," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1492–1505, 2018.

[27] C.-Y. Liu, J. B. Kotra, M. Jung, M. T. Kandemir, and C. R. Das, "Soml read: Rethinking the read operation granularity of 3d nand ssds," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 955–969.

[28] J. Kim, S. Kang, Y. Park, and J. Kim, "Networked ssd: Flash memory interconnection network for high-bandwidth ssd," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 388–403.

[29] R. Nadig, M. Sadrosadati, H. Mao, N. M. Ghiasi, A. Tavakkol, J. Park, H. Sarbazi-Azad, J. G. Luna, and O. Mutlu, "Venice: Improving solid-state drive parallelism at low cost via conflict-free accesses," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–16.

[30] C. Gao, X. Xin, Y. Lu, Y. Zhang, J. Yang, and J. Shu, "Parabit: processing parallel bitwise operations in nand flash memory based ssds," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 59–70.

[31] A. Khakifirooz, S. Balasubrahmanyam, R. Fastow, K. H. Gaewsky, C. W. Ha, R. Haque, O. W. Jungroth, S. Law, A. S. Madraswala, B. Ngo *et al.*, "30.2 a 1tb 4b/cell 144-tier floating-gate 3d-nand flash memory with 40mb/s program throughput and 13.8 gb/mm 2 bit density," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 424–426.

[32] J. Li, Q. Wang, P. P. Lee, and C. Shi, "An in-depth analysis of cloud block storage workloads in large-scale production," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2020, pp. 37–47.

ZenLeak: Practical Last-Level Cache Side-Channel Attacks on AMD Zen Processors

Han Wang[†], Ming Tang[†‡], Quancheng Wang[†], Ke Xu[†], Yinqian Zhang[§]
[†]Key Laboratory of Aerospace Information Security and Trusted Computing,
Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China
{han.wang, m.tang, wangquancheng, kexuwhu}@whu.edu.cn
[§]Southern University of Science and Technology, yinqianz@acm.org

***Abstract*—While Last-Level Cache (LLC) side-channel attacks often target inclusive caches, directory-based attacks on non-inclusive caches have been demonstrated on Intel and ARM processors. However, the vulnerability of AMD's non-inclusive caches to such attacks has remained uncertain, primarily due to challenges in reverse-engineering cache addressing, constructing eviction sets, and evicting private cache lines.**

This paper addresses these challenges and demonstrates the feasibility of conducting LLC side-channel attacks on AMD's non-inclusive caches. We first reverse-engineer the cache addressing functions for the L2 set index, L3 slice, and L3 set index. Leveraging this insight, we construct the first eviction sets on AMD processors. We then introduce the first LLC side-channel attack on AMD's Zen series CPUs. The effectiveness of our approach is validated by attacking OpenSSL's AES T-table.

I. Introduction

Encryption is essential for data confidentiality, but side-channel attacks exploit physical characteristics of cryptographic implementations—such as execution time [1]–[3] and power consumption [4]—to extract secret information like cryptographic keys. These attacks leverage subtle hardware implementation details without software vulnerabilities.

The LLC has become a key target for side-channel exploits, with the Prime+Probe attack [1] being particularly notable. This method is especially concerning because it does not require the attacker to share a core or memory with the victim, lowering prerequisites and increasing the likelihood of success.

Prime+Probe attacks have traditionally targeted Intel's inclusive LLCs. Techniques for inclusive caches are not directly applicable to non-inclusive caches. In non-inclusive caches, evicting data from L3 does not necessarily evict corresponding data from L1 and L2, reducing attack effectiveness. Specifically, an attacker on one core cannot evict data from another core's private cache (Figure 1).

It has been demonstrated that in non-inclusive cache architectures, Intel's directory structure can be exploited to conduct conflict-based cache attacks on the LLC [5]. This approach enabled the first cross-core Prime+Probe attack on non-inclusive caches by exploiting the shared directory (Figure 2).

In contrast, this method is ineffective on AMD's non-inclusive cache architecture due to fundamental differences in directory structures. Intel uses a shared, capacity-constrained directory across all cores, whereas AMD assigns a core-partitioned shadow tag [6] in the L3 cache for each L2 cache entry. Consequently, AMD's cache directories prevent attackers from inferring a victim's cache activities through a shared directory.

To date, no LLC-based cache attacks without shared resources have been successfully carried out against AMD's non-inclusive caches. This raises a critical research question:

> **Research Question**: Is AMD's non-inclusive cache architecture inherently secure and immune to last-level cache side-channel attacks?

Challenge: Addressing this question involves three key challenges. *First*, AMD's cache addressing structure is undocumented, complicating the mapping of memory addresses to specific cache sets and slices. *Second*, no methods exist for constructing eviction sets on AMD's non-inclusive caches, which are essential for cache side-channel attacks. *Third*, cache lines cannot be evicted from private caches on other cores, limiting cross-core attacks.

Experiment Setup: All experiments were conducted on an AMD Ryzen 9 5900X (Zen 3). Starting with the Zen series, AMD has employed non-inclusive caches [7]. While addressing functions may vary between CPUs, the overall cache structure remains consistent, allowing our reverse engineering and attack methods to be adapted across the AMD Zen lineup.

Contribution: Our contributions are threefold. *First*, we comprehensively reverse-engineer the cache addressing functions for the L2 set index, L3 slice, and L3 set index on AMD's Zen series processors, revealing how physical addresses map to cache sets. Our methods are architecture-independent and not limited to AMD CPUs. *Second*, building on this reverse engineering, we propose the first effective algorithm for constructing eviction sets on AMD processors. *Third*, we introduce a method for cross-core eviction of private cache lines, enabling LLC side-channel attacks on

[‡]Corresponding author. This work was supported in part by the National Key R&D Program of China (No. 2022YFB3103800).

[§]Yinqian Zhang is in part supported by National Natural Science Foundation of China under grant No. 62361166633, National Key R&D Program of China under grant No. 2023YFB4503902, Shenzhen Science and Technology Program under grant No. JSGG20220831095603007

AMD's non-inclusive caches. We successfully demonstrate this approach by attacking OpenSSL's AES T-table, revealing potential vulnerabilities in these systems.

Disclosure: We have responsibly disclosed our findings to the AMD PSIRT team. AMD has acknowledged our work.

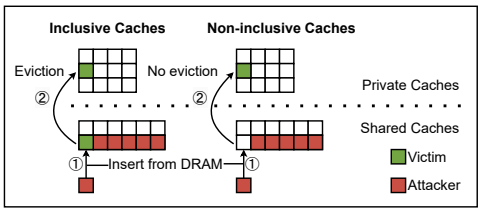


Fig. 1: Eviction attempts in (non-)inclusive cache architectures. In inclusive caches, inserting a new cache line into the LLC (step ①) evicts the corresponding line from the private cache (step ②). In non-inclusive caches, the victim cache line remains in the private cache despite the attacker's insertion.

II. Background and Related Work

LLC Side-Channel Attacks: Prime+Probe [1] introduced the concept of performing cache attacks on the LLC without shared memory. An attacker first fills specific cache sets with their data (prime) and then measures access latencies after the victim's execution (probe) to reveal memory access patterns. This method applies only to inclusive caches. Yan et al. [5] conducted directory-based attacks on Intel non-inclusive caches by targeting the Extended Directory (ED). By constructing eviction sets mapped to the same cache slice and the ED set, they force the victim's cache line from the private cache to the shared LLC, allowing detection of victim accesses through increased latencies. A similar attack has been performed on ARM processors [8]. Evict+Spec+Time [9] targets speculative attacks on Intel and AMD processors but replaces evict with the flush instruction in their implementation. This changes the threat model, as flush requires shared memory between attacker and victim, unlike eviction. On non-inclusive caches, flush and evict differ fundamentally, as eviction involves constructing sets and removing private cache lines—issues not fully explored in prior work.

Hardware Defenses Hardware approaches such as cache randomization [10] and partitioning [11] have not yet been widely adopted in commercial processors, leaving systems vulnerable to cache side-channel attacks. AMD has recommended software-based mitigation strategies. However, achieving universal and efficient protection remains challenging. Applying these techniques [12], [13] to arbitrary software is inherently complex and imposes a substantial burden on developers to safeguard sensitive data across diverse microarchitectures.

Reverse Engineering of Cache Addressing Functions: Initial efforts to reverse-engineer Intel LLC slice functions were presented in prior work [14], [15]. Specifically, two methods were proposed [15]: one utilizing the Performance Monitoring Unit (PMU) and another using latencies of the flush instruction. Subsequent research reverse-engineered the L2 set index of the Apple M1 processor using an eviction set [16]. An automated method for reverse-engineering nonlinear functions was also introduced in later studies [17]. However, these approaches have limitations when applied to AMD's non-inclusive caches, as discussed in Section III.

Construction of Eviction Sets: In inclusive caches, addresses are iteratively removed from a set, and if removing one stops evicting the target address, it is part of the eviction set [1]. Several improvements [18]–[20] have accelerated this pruning process. Various algorithms [5], [21] have also been proposed for Intel's non-inclusive caches; however, due to a lack of understanding of AMD's cache addressing, these algorithms have not been implemented on AMD processors. Specific details are discussed in Section IV.

III. Reverse Engineering

In this section, we address the first challenge: reverse engineering the cache addressing structure of AMD processors.

A. Cache Slicing

Two methods for reverse-engineering cache slicing on Intel CPUs were proposed: one based on Performance Monitoring Units (PMUs) and another on timing measurements of the flush instruction [15]. However, the PMU-based method is not applicable on AMD platforms due to the lack of relevant PMUs (e.g., LLC_LOOKUP events).

The principle of the timing-based method is that performing a flush operation on the slice closest to the core typically takes the least time. We found that this method cannot be directly applied to AMD CPUs because the time differences for flush operations across different slices are insufficient to provide conclusive results, as demonstrated in previous research [17]. We conducted a detailed analysis to understand the underlying reasons and subsequently enhanced the method.

Our analysis revealed that the execution time of the first 1,500 flush operations on each core is relatively short, but time increases significantly afterward (Figure 3). If we follow the original algorithm and frequently switch cores, the flush operations remain within the shorter time range, leading to insufficient differentiation between slices.

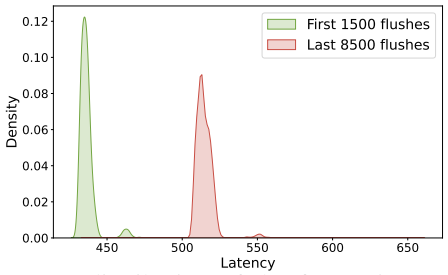


Fig. 3: Latency distribution of the first 1,500 and last 8,500 flushes on AMD cores.

To address this issue, we modified the algorithm to incorporate a warm-up phase and complete all address tests on each core before switching. This adjustment improves the reliability of the results. Although Gerlach et al. [17] proposed an eviction-based method to collect data on AMD CPU, it

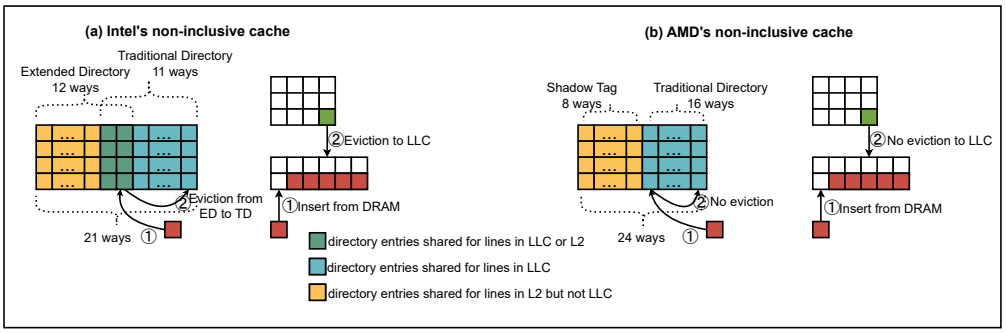


Fig. 2: Intel and AMD use distinct directory structures for non-inclusive caches. (a) Intel's shared directory allows LLC eviction through directory conflicts. (b) AMD's per-L2-line shadow tags prevent LLC eviction through the directory.

requires 23 hours to gather sufficient data for analyzing the slice-addressing hash function, whereas our method requires only 30 minutes.

On the Ryzen 9 5900X processor, we faced a unique challenge: the processor has six physical cores but eight cache slices in one CCX (CPU Complex). During our timing measurements, we observed that if a core flushes an address and the time is the shortest, we increment a count corresponding to that core-slice mapping. For certain addresses, we noticed that the counts between two cores were approximately equal, indicating that these addresses correspond to the extra two slices, as they do not uniquely map to a single core.

With the mapping between addresses and slice indexes established, we iterated through memory addresses, flipping the i-th bit (starting from bit 6, as bits 0–5 correspond to the cache line offset) in each address (`addr1`) to generate a new address (`addr2`). By comparing their index bits, we incremented the corresponding position in the count matrix if a difference was detected. This process tracks how flipped address bits affect index bits, revealing their relationship, as shown in Table I.

The threshold is calculated as:

$$\text{Threshold} = \left\lfloor \frac{N_{\text{addr}}}{2^B} \right\rfloor$$

where N_{addr} is the total number of addresses, and B denotes the number of address bits involved in hashing. This threshold determines whether a specific address bit contributes to the hash computation. The same formula is used to determine the involvement of address bits in Section III-B and III-C.

TABLE I: Correspondence between physical address bits and slice value bits.

Slice Bit	b6	b7	b8	b9	b10	b11	b12
0	23162	0	0	0	0	0	0
1	7216	11412	22290	6	8	13	10
2	11234	22448	97	7	14	19	16

Analyzing the count matrix, we derived the corresponding hash function. Previous studies [8], [15], [17] indicate that cache-addressing hash functions predominantly use XOR operations, so we focused exclusively on XOR-based functions in the analysis. Unlike previous work using an AMD Ryzen 9 5900HX [17], we found no bits involved in CCX selection on the Ryzen 9 5900X.

The specific hash function is as follows:

$$\{s_2, s_1, s_0\} = \{b_6 \wedge b_7,\ b_6 \wedge b_7 \wedge b_8,\ b_6\}$$

B. L2 Index Addressing

It is commonly assumed that cache set index bits correspond to the base-2 logarithm ($\log_2$) of the number of sets, as in Intel processors [5]. However, we found that on AMD processors, L2 and L3 cache indexing may involve the most significant bits (MSBs) of the address, contrary to this assumption.

This prompted us to investigate which bits contribute to L2 cache addressing on AMD processors. Similar patterns exist on other architectures; for instance, previous work showed that the Apple M1 CPU also uses high-order address bits for cache indexing [16]. Their method involved constructing eviction sets—a technique unavailable for AMD processors before our work. Moreover, our eviction set construction algorithm relies on insights gained from reverse-engineering these addressing functions. Similarly, the reverse engineering of snoop filter addressing on ARM processors relied on specific PMU events [8], which are not available on AMD processors. Our method does not depend on architecture-specific knowledge, making it broadly applicable to other architectures.

We conducted experiments to identify the bits involved in L2 cache indexing. Sixteen nodes were added to a linked list with physical addresses spaced at regular intervals. By adjusting the interval between addresses, we ensured that the physical addresses differed by only four bits (see Figure 4). If these bits do not contribute to the L2 set index computation, access latency increases due to multiple addresses mapping to the same cache set, exceeding the 8-way associativity. As shown in Figure 5, when the interval was set to 2^{16} and 2^{17}, access latency increased; when the interval was increased to 2^{18}, latency decreased, indicating that the addresses no longer mapped to the same cache set. We deduced that bits 6–15 and bits 21–27 of the physical address collectively determine the L2 cache index on AMD processors.

To further investigate the impact of higher-order address bits, we reverse-engineered the L2 cache addressing function using Algorithm 1. We first established two addresses:

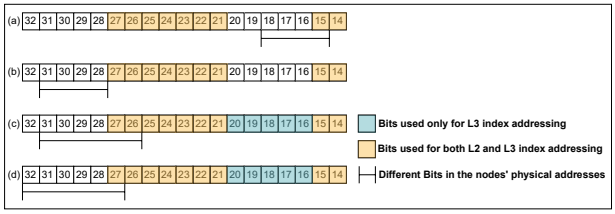


Fig. 4: Using a sliding window to control the bits of physical address differences among nodes in a linked list.

Addr1: Defined by a base address, an offset (gap), and varying cache sets.

Addr2: Defined by the base address and different cache sets.

Algorithm 1: Algorithm for determining the relationship between the L2 set index and the physical address.

Input : Memory region `mem`, range from `begin` to `end`, step size `gap`, number of nodes `num`
Output: Results of each test

```
base ← starting address of mem;
for offset from begin to end step gap do
    for j from 0 to number_of_L2_set do
        Addr1 ← base + offset + (j ≪ 6);
        AppendNodes(Addr1, num/2);
        for k from 0 to number_of_L2_set do
            count ← 0;
            Addr2 ← base + (k ≪ 6);
            AppendNodes(Addr2, num/2);
            ShuffleList();
            for trial from 0 to 9 do
                avg_time ← TraverseList() ;
                if avg_time exceeds threshold then
                    count ← count +1;
            if count >5 then
                same_set ← 1;
```

For each address, we constructed linked lists containing eight nodes, each separated by a gap of 0x10000. This setup ensured that the nodes within each list were identical in the bits relevant to the L2 cache index.

We measured the access latency by traversing these linked lists. If the combined latency exceeded a predefined threshold, it indicated that both lists mapped to the same L2 cache set, confirming how bits 21-–27 affected bits 6—15.

This analytical approach mirrors the method used in our cache-slicing analysis. After mapping addresses to L2 set indexes, we flipped specific address bits to generate new addresses and recorded the relationship between differing index bits and address bits, producing a count matrix (Table II).

Using the count matrix, we derived a hash function that employs XOR operations. Given the address bits:

$$b_i = (\text{addr} \gg i)\,\&1 \quad \text{for } i \in [6, 15] \cup [21, 27]$$

Each bit of the L2 index is defined as follows:

$$index_i = \begin{cases} b_6 & \text{if } i = 0, \\ b_{i+6} \oplus index_{i-1} & \text{if } 0 < i < 3, \\ b_{i+6} \oplus b_{i+18} \oplus index_{i-1} & \text{if } 3 \le i \le 10. \end{cases}$$

C. L3 Index Addressing

Reverse engineering the index addressing for the L3 cache follows the same principles as for the L2 cache but involves differences in identifying specific bits in the calculation. A linked list of 64 nodes was constructed, with physical addresses spaced at regular intervals. By adjusting the intervals, we ensured that the physical addresses differed by exactly six bits, as shown in Figure 4. Cache lines were evicted from private caches before each traversal, as detailed in Section V-A. As depicted in Figure 5, the results showed that with a 2^{26} interval, the 64 nodes spanned four L3 sets, each containing 16 cache lines. In contrast, with a 2^{27} interval, the nodes occupied only two L3 sets, exceeding the associativity limit and increasing access latency. These observations indicate that L3 set indexing is determined by bits 9–27.

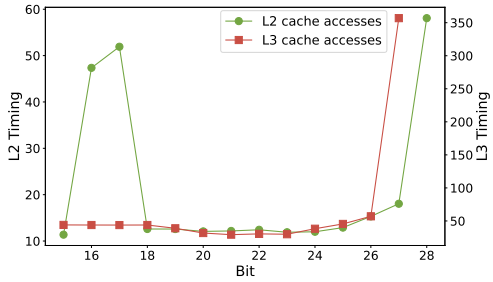


Fig. 5: Relationship between the interval in physical addresses of linked list nodes and memory access timing.

Similar to reversing the L2 set index, given the address bits:

$$b_i = (\text{addr} \gg i)\,\&1 \quad \text{for } i \in [9, 27]$$

Each bit of the L3 index is defined as follows:

$$index_i = \begin{cases} b_9 \oplus b_{21} & \text{if } i = 0, \\ b_{i+9} \oplus b_{i+21} \oplus index_{i-1} & \text{if } 0 < i \le 7, \\ b_{i+9} \oplus index_{i-1} & \text{if } 7 < i \le 12. \end{cases}$$

IV. Eviction Set Construction

An eviction set is a group of memory addresses that map to the same cache set; in cache attacks, accessing these addresses forces the eviction of specific cache lines through the cache replacement policy. In this section, we analyze why previous eviction set construction algorithms fail on AMD Zen processors and introduce a new method based on reverse-engineering cache addressing.

Firstly, algorithms designed for inclusive caches are not effective on AMD's non-inclusive cache architecture. An eviction set construction algorithm for inclusive caches [1] iteratively removes addresses from a candidate set U and observes whether the target address x still experiences eviction. If removing an address x' prevents eviction of x, then x' is necessary for the eviction set. However, on non-inclusive caches like those in AMD processors, some addresses in the minimal eviction set U may reside only in private caches and not in the LLC. They cannot conflict with x in the LLC, which results in false negatives when constructing eviction sets [5].

TABLE II: Correspondence between the physical address bits and L2 set index bits.

Index Bit	b6	b7	b8	b9	b10	b11	b12	b13	b14	b15	b16	b17	b18	b19	b20	b21	b22	b23	b24	b25	b26	b27
0	130946	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	65410	130818	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	32640	65280	130562	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	16256	32512	65024	130050	0	0	0	0	0	0	0	0	0	0	0	130050	0	0	0	0	0	0
4	8064	16128	32256	64514	129026	0	0	0	0	0	0	0	0	0	0	64512	129026	0	0	0	0	0
5	3968	7936	15872	31746	63490	126978	0	0	0	0	0	0	0	0	0	31744	63490	126978	0	0	0	0
6	1920	3840	7680	15362	30722	61442	122882	0	0	0	0	0	0	0	0	15360	30722	61442	122882	0	0	0
7	896	1792	3584	7170	14338	28674	57346	114690	0	0	0	0	0	0	0	7168	14336	28672	57344	114690	0	0
8	384	768	1536	3073	6145	12289	24577	49153	98306	0	0	0	0	0	0	3072	6144	12288	24576	49153	98305	0
9	128	256	512	1024	2048	4096	8192	16384	32769	65538	0	0	0	0	0	1024	2048	4096	8192	16384	32768	65536

Secondly, existing algorithms for non-inclusive caches also face limitations on AMD processors. The L2 occupy set was introduced to address false positives in non-inclusive caches [5]. It evicts addresses from private caches to ensure that all candidate addresses reside in the LLC and can conflict with x. It is effective on Intel's non-inclusive caches because flipping the 16th bit of the physical address allows the L2 occupy set and the L3 eviction set to be mapped to different slices, preventing false positives. However, our reverse-engineering of AMD's cache addressing revealed that all cache lines in the same L2 cache set are mapped to the same LLC slice, making this method inapplicable to AMD architecture.

Through reverse-engineering in Section III, we determined that in Zen 3 processors, the L2 set index is derived from bits 6—15 and 21—27 of the physical address, LLC slice addressing is determined by bits 6—8, and L3 set indexing is based on bits 9—27. Addresses within the same L2 set and slice must also belong to the same L3 set. Therefore, we can adapt the L2-driven Candidate Address Filtering method [21] for AMD processors.

To construct the L3 eviction set, we first identify the L2 eviction set EV'. After obtaining the L2 eviction set EV', we filter the address set U to select those addresses that share the same L2 set as the target address, using them as candidates for the L3 eviction set. The L3 eviction set EV is then constructed by ensuring that all candidate addresses are evicted into the L3 cache, avoiding false negatives. Finally, we construct the L3 eviction set from these candidates. The whole process is shown in Algorithm 2.

In this approach, the L2 eviction set serves as the occupy set. Since it is a subset of the full eviction set, this structure prevents false positives and enables effective eviction set construction on AMD's non-inclusive caches.

Algorithm 2: Eviction set construction algorithm

Input : Target address x, Address set U
Output: Eviction set EV
1 $EV' \leftarrow$ `FindEV`$(x, U, L2_Threshold, \varnothing)$;
2 $U' \leftarrow$ `CheckConflict`(U, EV');
3 $EV \leftarrow$ `FindEV`$(x, U', L3_Threshold, EV')$;

In our experiments, the `FindEV` function used the $O(n^2)$ algorithm, proposed in previous work [1], but more efficient algorithms can be applied. Figure 6 shows the cumulative success rates: 98% after 7 attempts on 1GB pages and 78% after 8 attempts on 4KB pages.



Fig. 6: Cumulative success rate of eviction set construction on 1GB and 4KB pages.

V. Side Channels

A. Eviction from Private Caches

It was observed that a context switch on Intel processors may flush cache data to memory [22]. This allows an attacker to evict a victim's cache lines to memory by inducing a context switch, for example, by sending a signal. If both processes share the same user ID, no special privileges are required to send a signal to a process running on a different core.

However, our experiments on AMD CPUs revealed different behavior. As shown in Figure 7, we measured the cache latency of accessing cache lines from the same L2 and L3 cache sets, both under normal conditions and after the process received a signal. Under normal conditions, when the number of entries was below 24 (the combined associativity of the L2 and L3 caches), accesses hit the cache, resulting in low latency. When the number of entries exceeded 24, cache misses occurred, leading to a significant increase in latency.

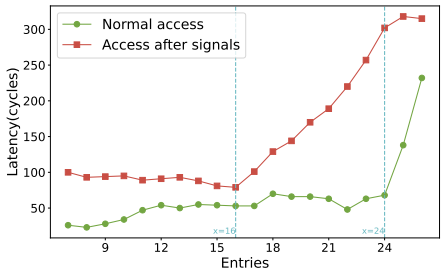


Fig. 7: Comparison of cache access latency under normal conditions and after receiving signals.

Interestingly, after receiving a signal, no significant increase in latency was observed, suggesting that the data remained in the L3 cache. When the number of entries exceeded 16 (the L3 cache associativity), latency increased, indicating that the

L2 set was not involved. This suggests that the signal caused cache lines to be evicted only from private caches.

Thus, we conclude that on AMD platforms, the signal mechanism can be exploited to flush the victim's private caches while retaining data in the L3 cache, presenting a unique opportunity for cache-based side-channel attacks.

B. *Prime+Signal+Probe*

Based on these findings, we propose a cache attack targeting the LLC in AMD's non-inclusive cache architecture. In this attack, the attacker first identifies the victim's target LLC sets, utilizing power spectral density to detect periodic accesses to the LLC sets in the frequency domain [21]. Once the target LLC sets are identified, the attack proceeds as follows.

The attacker constructs an eviction set, either through an eviction set construction algorithm or by selecting cache lines from a 1GB large page. The relevant cache sets are primed to establish a known state. A signal is sent to the victim process, triggering a context switch that evicts the victim's cache lines from private caches. The attacker then probes the cache sets by measuring access latency. By analyzing these latency measurements, the attacker can determine whether the victim accessed specific cache lines. If the victim accessed cache lines containing secret data, the attacker's corresponding cache lines would be evicted, leading to increased access latency. This process is illustrated in Figure 8.

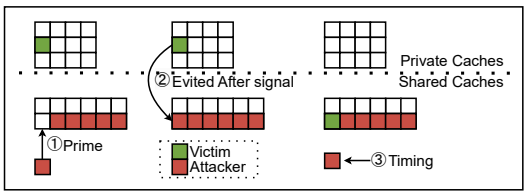


Fig. 8: Overview of the Prime+Signal+Probe attack. The attacker primes the cache set with their data (step ①), and sends a signal to the victim, triggering a context switch that evicts the victim's cache lines from the private L2 cache to the shared LLC if they were accessed by the victim beforehand (step ②). Finally, the attacker probes for increased latency to detect if specific lines were accessed by the victim (step ③).

We selected the AES T-table implementation as our side-channel attack target. Despite countermeasures [23]–[25] that mitigate most cache side channels in AES implementations, the T-table approach remains widely used for examining emerging side-channel attacks compared to earlier methods [3], [26], [27]. Drawing from similar attack scenarios [1], [21], our Prime+Signal+Probe technique also applies to other cryptographic algorithms, including RSA, ElGamal, and ECDSA.

The T-table implementation optimizes AES by converting SubBytes, ShiftRows, and MixColumns into 16 memory lookups from four precomputed tables. In the first round, table accesses are made to entries $T_j[p_i \oplus k_i]$, where $i \equiv j \mod 4$ and $0 \leq i < 16$, allowing attackers to infer possible key-byte values (k_i) when the plaintext (p_i) is known.

We conducted a known-plaintext attack on the AES T-table implementation in OpenSSL. By analyzing whether the victim accessed specific T-table entries, we inferred possible key-byte values. For each round, we recorded the results of 1024 AES encryptions. Figure 9 illustrates a cache template generated from 1024 encryptions, revealing a discernible pattern.

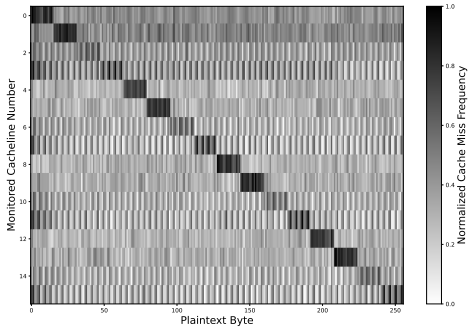


Fig. 9: Cache templates showing the cache miss frequency of the eviction set.

We performed multiple rounds of the attack, applying majority voting to the results. Scenarios included selecting cache lines from the same L3 set on a 1GB page and constructing eviction sets with the proposed algorithms on 1GB and 4KB pages. Figure 10 shows the accuracy of majority voting: on 4KB pages, accuracy reached 100% with zero standard deviation when the vote count exceeded 16, allowing key-byte inference. For 1GB pages, achieving 100% accuracy required a majority vote count of 30 when constructing eviction sets and 22 when selecting cache lines directly.

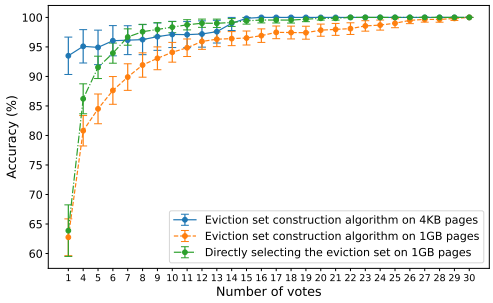


Fig. 10: Accuracy of majority voting on 1GB and 4KB pages.

VI. Conclusion

We analyzed the security of AMD's non-inclusive caches. By reverse-engineering the cache addressing functions, developing an eviction set construction algorithm, and employing signals to evict cache lines from private caches, we successfully execute the last-level cache side-channel attack on AMD processors. Given the challenges of applying software mitigation to arbitrary applications, our findings highlight the need for hardware-based defenses in commercial processors.

REFERENCES

[1] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*, (San Jose, CA), pp. 605–622, IEEE, 2015.

[2] H. Wang, M. Tang, K. Xu, and Q. Wang, "Cache bandwidth contention leaks secrets," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2024.

[3] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pp. 279–299, Springer, 2016.

[4] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 355–371, IEEE, 2021.

[5] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *2019 IEEE Symposium on Security and Privacy (SP)*, (San Francisco, CA, USA), pp. 888–904, IEEE, 2019.

[6] S. Srinivasan and W. L. Walker, "Shadow tag memory to monitor state of cachelines at different cache level," Sept. 11 2018. US Patent 10,073,776.

[7] I. Cutress, "The amd zen and ryzen 7 review: A deep dive on 1800x, 1700x and 1700," *Anandtech, Mar*, vol. 2, 2017.

[8] Z. Kou, S. Sinha, W. He, and W. Zhang, "Attack directories on arm big.little processors," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, (San Diego California), pp. 1–9, ACM, 2022.

[9] S. H. W. Cheng, C. Chuengsatiansup, D. Genkin, D. McNeil, T. Murray, Y. Yarom, and Z. Zhang, "Evict+ spec+ time: Exploiting out-of-order execution to improve cache-timing attacks," *Cryptology ePrint Archive*, 2024.

[10] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "{ScatterCache}: thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium (USENIX Security 19)*, pp. 675–692, 2019.

[11] G. Dessouky, A. Gruler, P. Mahmoody, A.-R. Sadeghi, and E. Stapf, "Chunked-cache: On-demand and scalable cache isolation for security architectures," *arXiv preprint arXiv:2110.08139*, 2021.

[12] S. Weiser, A. Zankl, R. Spreitzer, K. Miller, S. Mangard, and G. Sigl, "{DATA}–differential address trace analysis: Finding address-based {Side-Channels} in binaries," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 603–620, 2018.

[13] G. Doychev and B. Köpf, "Rigorous analysis of software countermeasures against cache attacks," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 406–421, 2017.

[14] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space aslr," in *2013 IEEE Symposium on Security and Privacy*, pp. 191–205, IEEE, 2013.

[15] C. Maurice, N. L. Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering intel last-level cache complex addressing using performance counters," in *Research in Attacks, Intrusions, and Defenses*, vol. 9404 of *Lecture Notes in Computer Science*, pp. 48–65, Springer International Publishing, 2015.

[16] J. Yu, A. Dutta, T. Jaeger, D. Kohlbrenner, and C. W. Fletcher, "Synchronization storage channels ({{{{{S2C}}}}}: Timer-less cache {Side-Channel} attacks on the apple m1 via hardware synchronization instructions," in *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 1973–1990, 2023.

[17] L. Gerlach, S. Schwarz, N. Faroß, and M. Schwarz, "Efficient and generic microarchitectural hash-function recovery," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 3661–3678, IEEE Computer Society, 2024.

[18] P. Vila, B. Köpf, and J. F. Morales, "Theory and practice of finding eviction sets," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 39–54, IEEE, 2019.

[19] Z. Xue, J. Han, and W. Song, "Ctpp: A fast and stealth algorithm for searching eviction sets on intel processors," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 151–163, 2023.

[20] T. Kessous and N. Gilboa, "Prune+ plumtree-finding eviction sets at scale," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 173–173, IEEE Computer Society, 2024.

[21] Z. N. Zhao, A. Morrison, C. W. Fletcher, and J. Torrellas, "Last-level cache side-channel attacks are feasible in the modern public cloud," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 582–600, 2024.

[22] Y. Chen, A. Hajiabadi, L. Pei, and T. E. Carlson, "Prefetchx: Cross-core cache-agnostic prefetcher-based side-channel attacks," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (Edinburgh, United Kingdom), pp. 395–408, IEEE, 2024.

[23] S. Gueron, "Intel advanced encryption standard (aes) new instructions set," *Intel Corporation*, vol. 128, 2010.

[24] R. Könighofer, "A fast and cache-timing resistant implementation of the aes," in *Cryptographers' Track at the RSA Conference*, pp. 187–202, Springer, 2008.

[25] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 1–17, Springer, 2009.

[26] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "{RELOAD+ REFRESH}: Abusing cache replacement policies to perform stealthy cache attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1967–1984, 2020.

[27] J. Lee, F. Sang, and T. Kim, "Prime+ retouch: When cache is locked and leaked," *arXiv preprint arXiv:2402.15425*, 2024.

Sphinx: A High-Performance Hybrid Index for Disaggregated Memory With Succinct Filter Cache

Jingxiang Li[§], Shengan Zheng[‡*], Bowen Zhang[§], Hankun Dong[§], Linpeng Huang[§*]
[§]Shanghai Jiao Tong University [‡]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
[*]Corresponding authors: {shengan, lphuang}@sjtu.edu.cn

***Abstract*—Disaggregated memory (DM) architecture physically separates computing and memory resources into distinct pools interconnected via high-speed networks within data centers, with the aim of improving resource utilization compared to traditional architectures. Most existing range indexes for DM that support variable-length keys are based on adaptive radix trees. However, these indexes exhibit suboptimal performance on DM due to excessive network round trips during tree traversal and inefficient node-based caching mechanisms.**

To address these issues, we propose Sphinx, a novel hybrid index for DM. Sphinx introduces an Inner Node Hash Table to minimize the network round trips during index operations by replacing the sequential tree traversal with parallel hash reads. Sphinx incorporates a Succinct Filter Cache to further minimize network overhead while keeping the computing-side cache small and coherent. Experimental results show that Sphinx outperforms state-of-the-art counterparts by up to 7.3× in the YCSB benchmark.

I. Introduction

Disaggregated Memory (DM) architecture [1]–[5] has gained significant attention in both academia and industry in recent years. DM separates the memory and computation resources into distinct pools. This separation allows for independent scaling of these resources, which addresses the problem of low resource utilization in modern data centers. Since the memory side has minimal computing power, clients on the computing side typically access memory resources on the memory side via high-speed networks such as RDMA [6] and CXL [7], which enables clients to directly read from or write to remote memory without involving remote CPUs.

The Adaptive Radix Tree (ART) [8], [9] is a widely used index structure in many data center applications, such as key-value storage systems and transaction processing systems, due to its efficient support for variable-length keys. However, existing ARTs [9]–[12] exhibit suboptimal performance on DM due to excessive network round trips and the inefficient node-based caching mechanism. Specifically, executing index operations on tree-based indexes like ART requires traversing the hierarchical tree structure [13]. On DM, this traversal involves multiple network round trips between computing-side clients and the memory-side index. Since accessing remote memory remains significantly slower than accessing local memory, these excessive round trips result in high latency and become the primary performance bottleneck during index operations.

Existing ART for DM [10] attempts to reduce tree traversal round trips by caching certain inner nodes on the computing side. However, this node-based caching mechanism is inefficient on DM. Firstly, the computing side of DM has limited memory resources, allowing only a small subset of inner nodes to be cached. As a result, node-based caching fails to effectively reduce network round trips, leading to diminished performance. Furthermore, node-based caching introduces potential cache coherence issues, as remote inner nodes may be modified by other clients, causing inconsistencies between the computing-side cache and the memory-side index.

To address these issues, we propose Sphinx, a hybrid index designed to minimize network overhead during index operations while keeping the computing-side cache small and coherent. To reduce network round trips, Sphinx incorporates an *Inner Node Hash Table* to store the essential metadata of ART inner nodes. This allows clients to directly access inner nodes through reading hash entries in parallel, rather than traversing the tree sequentially. To achieve optimal performance with limited computing-side memory resources, we propose the *Succinct Filter Cache*. The succinct filter cache further reduces the network overhead by minimizing hash entries read to one in most cases. Instead of caching the full content of inner nodes, the succinct filter cache employs a succinct (space-efficient) data structure (e.g., cuckoo filter [14]) to track only the existence of inner nodes. As a result, the succinct filter cache significantly improves memory utilization and preserves coherence when remote inner nodes are modified by other clients.

The contributions of this paper are summarized as follows:

- We thoroughly analyze existing ARTs on DM, demonstrating that they provide inferior performance due to excessive network round trips introduced by traversing the hierarchical tree structure and the unsuitable node-based caching mechanism.
- We propose Sphinx, a hybrid range index supporting variable-length keys on DM. By introducing the *Inner Node Hash Table*, Sphinx minimizes network round trips by replacing the sequential tree traversal with reading hash entries in parallel.
- We propose the *Succinct Filter Cache*, which further reduces the network overhead of Sphinx and enables it to achieve optimal performance with limited computing-side memory resources.
- We conduct extensive evaluations of Sphinx and compare it with the state-of-the-art range indexes for variable-length keys on DM. Experimental results show that

Sphinx outperforms existing systems in terms of both throughput and latency.

II. Background And Motivation

A. Disaggregated Memory Architecture

Disaggregated memory (DM) architecture physically separates computing resources (e.g., CPU cores) and memory resources (e.g., DRAM) into distinct pools within data center, which aims to improve the resource utilization in traditional data center architectures [5], [15], [16]. In a DM setup, Compute Nodes (CNs) contain extensive computing resources but have limited memory. Conversely, Memory Nodes (MNs) are equipped with substantial memory resources but only wimpy computing power. CNs and MNs are interconnected via high-speed networks such as RDMA [6] and CXL [7].

RDMA is a promising solution for disaggregated memory architecture in modern data centers, as it not only offers high bandwidth (e.g., 200 Gbps) and low latency (e.g., 2 μs) [17]–[19] but also provides a set of one-sided operations (`RDMA Read/Write/CAS` (`Compare And Swap`) /`FAA` (`Fetch And Add`)), allowing clients in CNs to directly access remote memory without requiring CPU intervention on the MN side. This capability is particularly beneficial for DM, where minimizing CPU overhead on MNs with limited computing resources is critical.

B. Adaptive Radix Tree

The Adaptive Radix Tree (ART) [8] is a space optimized variant of the radix tree [20]. As shown in the memory nodes of Fig. 1, each inner node in ART represents a unique prefix of a key stored in the tree. This characteristic allows it to efficiently support variable-length keys. To improve space utilization, ART employs inner nodes with different capacities (e.g., 4, 16, 48 and 256) to store their children, allowing them to grow dynamically as new keys are inserted into the tree. To reduce tree height and enhance the performance of index operations, ART incorporates a path compression mechanism to merge multiple inner nodes with only one child into a single inner node (e.g., the inner node marked as *LYR* in Fig. 1) and eliminate inner nodes along the path to a single leaf node (e.g., the path from *LYR* to *LYRICS* in Fig. 1).

Existing ARTs [9], [10], [12] exhibits suboptimal performance on DM due to excessive network round trips. ARTs follow a hierarchical structure [13], in which a node can only be accessed after its address is retrieved from its parent node. In DM architectures, the index resides on the MN side with sufficient memory, which is separated from the clients on the CN side. This separation necessitates excessive network round trips during tree traversal, resulting in high latency and early saturation of network resources.

Existing ART for DM [10] attempts to mitigate this issue by caching certain inner nodes. During index operations, the client first traverses the tree formed by the cached inner nodes locally, then proceed to traverse the remote tree from the deepest node reached in the local traversal. However,

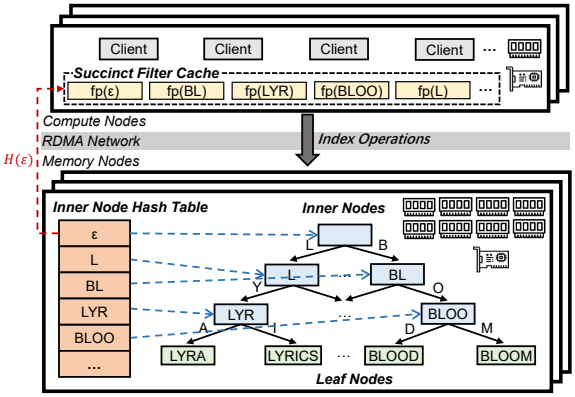


Fig. 1: Overview of Sphinx.

this node-based caching mechanism is inefficient in memory-limited CNs. First, node based caching requires a significant amount of memory to achieve optimal performance, which is impractical for memory-limited CNs. For example, as shown in Sec. V-B, SMART [10], the state-of-the-art ART on DM, still deliver suboptimal performance even with a cache size set to 17.6% of the dataset size. Second, node-based caching can lead to cache coherence issues, as remote ART nodes may be updated by other clients. Although SMART mitigates this issue by preallocating a space of Node-256 for each inner node and employing an associated reverse checking mechanism, it incurs significant memory overhead on the MN side. The overhead ranges from 2.1–3.0× in our experiments, as detailed in Sec. V-D.

III. Design

Based on the above analysis, we propose Sphinx, a hybrid index for variable-length keys on DM, designed to minimize network overhead in index operations while keeping the CN-side cache small and coherent.

The architecture of Sphinx is illustrated in Fig. 1. The ART Nodes of Sphinx are evenly distributed across MNs by consistent hashing [21]. To minimize network round trips during index operations, each MN in Sphinx employs an *Inner Node Hash Table* to store the addresses and lightweight metadata of ART inner nodes allocated to that MN. This reduces network round trips during index operations by replacing the sequential tree traversal with reading hash entries in parallel. To further reduce the network overhead, Sphinx introduces a *Succinct Filter Cache* that efficiently filter out unnecessary hash entry reads, thereby reducing network bandwidth consumption. The succinct filter cache adopts a succinct data structure (e.g., cuckoo filter [14]) to track the existence of inner nodes, rather than storing their full content. This approach not only minimizes memory consumption on the CN side, but also eliminates cache coherence issues that could arise from directly caching the inner nodes.

A. Inner Node Hash Table

To reduce network round trips, Sphinx introduces an MN-side *Inner Node Hash Table* for the ART, designed to break

the serial dependencies between ART nodes. As illustrated in Fig. 1, we construct a hash table for the inner nodes of the ART, where the key is the full prefix of each inner node, while the value stores the node's address and its associated metadata. During index operations, instead of sequentially fetching each inner node through tree traversal, the client is able to read all the hash entries of these inner nodes in parallel. Based on these hash entries, the client then retrieves the deepest inner node with another network round trip.

With the inner node hash table, locating the value corresponding to a key can be optimized in a parallel manner. Specifically, for a given key K (e.g., *LYRICS* in Fig. 1), the client retrieves all hash entries corresponding to the prefixes of K (*LYRICS*, *LYRIC*, ..., L) in parallel to find the deepest inner node corresponding to K. This approach stems from the path compression mechanism [8] used in ART, which prevent the client from directly determining the existence of each inner node, necessitating a read of all possible hash entries (can be further optimized to only read one hash entry in most cases, as described in Sec. III-B). We adopt RACE hashing [22] as the index of the inner node hash table, which allows each hash entry to be retrieved within a single network round trip. By further utilizing the doorbell batching mechanism [23], which allows sending multiple RDMA operations to the NIC in a single batch, reading all these hash entries can be performed in a single round trip.

To further retrieve the value of K with these hash entries, the client then identifies the valid hash entry corresponding to the longest prefix (*LYR*), which represents the deepest inner node. Next, the client retrieves this deepest inner node from MNs using the metadata and address stored in the hash entry. Finally, the client retrieves the target leaf node through this deepest inner node and extracts the value from the leaf node.

The inner node hash table incurs only a slight memory consumption on the MN side. Specifically, it is significantly smaller than the inner nodes themselves, as each hash entry only stores the address of the inner node and associated metadata (e.g., a short fingerprint and node type as shown in Fig. 3), fitting within just 8 bytes. In comparison, an ART inner node consumes 40–2056 bytes of memory. Our experiments in Sec. V-D show that the inner node hash table incurs only 3.3–4.9% additional memory usage on the MN side.

B. Succinct Filter Cache

Although reading inner nodes directly from the inner node hash table reduces network round trips, it does not alleviate bandwidth consumption. Specifically, for a key of length L, the client must read $\Theta(L)$ hash entries to locate the deepest inner node, offering no advantage over tree traversal. Additionally, similar to tree traversal, reading multiple hash entries generates a significant load on network, as the NIC must process a substantial number ($\Theta(L)$) of network packets for each index operation. This high load accelerates the saturation of network resources, ultimately diminishing throughput of the system.

To address the above issue, we propose the *Succinct Filter Cache*, a lightweight per-CN cache based on cuckoo filter [14].

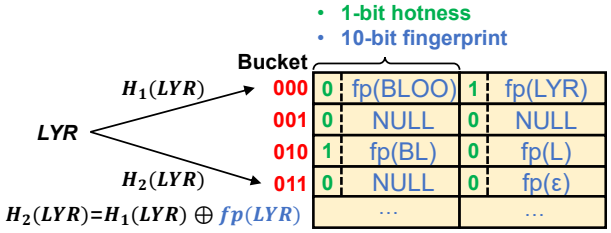


Fig. 2: Succinct Filter Cache.

The succinct filter cache reduces the number of hash entry reads per index operation from $\Theta(L)$ to one in most cases, while maintaining minimal space consumption and preserving cache coherence. The succinct filter cache tracks the existence of inner node prefixes in the remote index using a cuckoo filter, a succinct data structure. This allows the client to locally determine the prefix of the deepest inner node for the key with minimal CN-side memory usage. Consequently, the client only needs to fetch a single hash entry corresponding to the deepest inner node, significantly reducing network overhead associated with reading multiple hash entries. Additionally, the succinct filter cache maintains coherence when remote inner nodes are modified, as it does not store their content.

As illustrated in Fig. 2, the succinct filter cache follows the structure of cuckoo filter. To perform an existence check (i.e., to determine whether a given prefix exists in the succinct filter cache), the client computes the hash values (H_1 and H_2) of the prefix and checks if the corresponding hash fingerprint (fp) is present in the candidate buckets determined by the hash values. During index operations, for a given key K (e.g., *LYRICS*), the client first performs existence checks for all prefixes of K (*LYRICS*, *LYRIC*, ..., L) in the succinct filter cache and identify the longest prefix P that exists in the succinct filter cache (*LYR*). The client then only fetches the hash entry of P from remote inner node hash table. Finally, the client retrieves the deepest inner node and further retrieves the leaf node to extract the value corresponding to K.

In most cases, as described above, an index operation can be performed in three network round trips: reading the hash entry, reading the inner node, and retrieving the leaf node. Due to the probabilistic nature of cuckoo filter, in false-positive cases (<1%), the succinct filter cache may incorrectly indicate that a prefix exists in the remote index when it does not. Nevertheless, both the performance and correctness of index operations remains unaffected by these rare cases. As illustrated in Fig. 3, both the hash entry and the inner node contain metadata that allows the client to avoid traversing the tree starting from an unmatched inner node. For example, the hash entry includes a 12-bit hash fingerprint (fp_2), while the inner node header provides a 42-bit full prefix hash. In the rare scenario where both the fp_2 and the full prefix hash collide, Sphinx may traverse the tree starting from an unmatched inner node U. However, the correctness is still guaranteed, as the client will eventually reach a leaf node to retrieve the corresponding value. At this point, the client computes the common prefix between the key used in the index operation

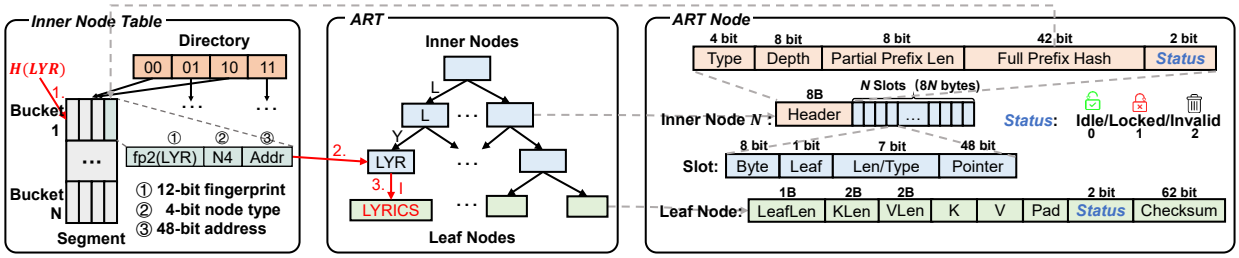


Fig. 3: Detailed data structures of Sphinx

and the key stored in the leaf node. If the common prefix is shorter than the prefix associated with U, the client identifies the false positive and retries the index operation with a shorter prefix from the succinct filter cache. Since such cases are extremely rare ($<$0.01%), the overall performance remains unaffected.

To address scenarios where the dataset size is so large that even tracking the existence of all inner nodes in CNs becomes infeasible, as illustrated in Fig. 2, we further propose a lightweight mechanism to identify hot prefixes in the succinct filter cache. This mechanism is inspired by the second chance page replacement policy [24], which approximates the LRU algorithm using only one additional *hotness* bit per entry. Specifically, when a new prefix is inserted, its *hotness* bit is initialized to 0, indicating it has not been recently used. The *hotness* bit is set to 1 when the entry is accessed, marking it as recently used. During insert operations, if both candidate buckets are full, the client randomly selects an entry with the *hotness* bit set to 0 for replacement. If all entries in both candidate buckets have their *hotness* bits set to 1, the client adopts the cuckoo hashing mechanism [25] to relocate existing entries and make space for the new one. During relocation, the *hotness* bits of all relocated entries are reset to 0, indicating they are eligible for future eviction.

The succinct filter cache is particularly well-suited for DM. First, as demonstrated in prior work [14], using a 10-bit fingerprint per item is sufficient to achieve a low false positive rate ($<$1%) for existence checks, making it significantly more space-efficient than the node-based caching mechanism, which require 40–2056 bytes of memory to cache a single inner node. Second, the succinct filter cache inherently maintains coherence even when inner nodes in the remote memory are modified by other clients. These modifications impact only types, slots, and partial prefixes of inner nodes, leaving their full prefixes unchanged. Consequently, they have no effect on the coherence of the succinct filter cache, which exclusively tracks the existence of full prefixes.

C. Concurrency Control

Sphinx employs a lightweight concurrency control mechanism that enables lock-free read operations and node-grained locks for write operations. To further reduce network overhead in update operations, Sphinx adopts a checksum-based in-place update scheme for leaf nodes.

For concurrency control, as illustrated in Fig. 3, most components (e.g., hash entries, headers and slots) of Sphinx fit within 8 bytes, enabling clients to modify them atomically using the `RDMA CAS` operation. In the inner node hash table, both read and write operations are executed without acquiring locks, as a write operation only affects an 8-byte hash entry and can be performed atomically. For the ART, Sphinx resolves write-write conflicts using a node-grained lock (status) and stores lightweight metadata (e.g., the type and depth of the inner node) in the header of inner nodes to synchronize lock-free reads with a write.

With the consistency maintained in both the ART and the inner node hash table, the correctness of Sphinx is ensured in most cases. The remaining correctness issue arises when a client reads an outdated hash entry while another client is performing a *node type switch*. In ART, a node type switch occurs when a client attempts to insert a new key into a full inner node. In such cases, a new inner node with more slots is allocated, and the original node is replaced by the new one. However, after the original node is removed from the ART, a client may still access it through the inner node hash table before the change propagates to the hash table, resulting in a temporary inconsistency. To address this issue, as shown in Fig. 3, the client sets the status field of the old inner node to *Invalid* following a node type switch. Readers retrieving an inner node from MNs check the status field before use. If the status field is marked *Invalid*, the reader retries the index operation.

For the update operation, if the new value fits within the original leaf node, the client performs an in-place update to reduce network overhead. As shown in Fig. 3, the client begins the in-place update by locking the leaf node, setting its status field to *Locked* using one `RDMA CAS` operation. Sequentially, the client writes the new value, updates the status field to *Idle* and computes the new checksum of the leaf node locally. Finally, the client writes the entire leaf node back to corresponding MN using a single `RDMA Write` operation. To avoid reading partially modified data, all clients verify the checksum of a leaf node before using it. This in-place update scheme eliminates an RDMA operation during an update operation by combining the lock release with the value write, thereby reducing network overhead.

IV. Index Operations

In this section, we present the detailed implementation of the index operations in Sphinx. By leveraging the inner node hash table and succinct filter cache, Sphinx achieves high performance in index operations, primarily by replacing the

sequential tree traversal with a single hash entry read. This substantially reduces network overhead in terms of both round trips and bandwidth consumption.

Search (K). The client first calculates the hash values of all prefixes of K and identifies the longest prefix P exists in the succinct filter cache. As shown in Fig. 3, the client then fetch the corresponding hash bucket from the inner node hash table in one round trip. To avoid additional network round trips in determining the address of buckets through the directory, each CN maintains a local directory cache, which is small enough to fit within limited CN-side memory (typically 2–5% of the succinct filter cache size). The client then iterates through the hash bucket and fetch all the inner nodes whose hash entry has the same 12-bit fingerprint (fp_2) as P. Typically, there is only one candidate hash entry, as discussed in prior works [26]. Finally, the client performs tree traversals starting from the fetched inner nodes, ultimately reaching the leaf nodes, where the value corresponds to K is extracted. In most cases, the tree traversal requires only one additional network round trip, as the succinct filter cache enables the client to determine the deepest inner node of K. In rare cases, the client may fetch an inner node that is not the deepest due to the structural modifications made by other clients to the remote ART. In such case, the client updates the succinct filter cache for any prefixes not present in the cache to maintain its freshness.

Insert (K, V). The client begins by locating the deepest inner node following the search procedure, and then traverses the tree starting from the node until it reaches a leaf node, verifying that the inner node prefix matches K. Subsequently, the client writes the new leaf node with the new key-value pair using an `RDMA write`. If the operation triggers a node type switch or a node split, the client also writes a new inner node with a slot pointing to the newly created leaf node. To ensure consistency, the client acquires node-grained locks on the affected inner node by setting its status field to $Locked$ with an `RDMA CAS`. This lock acquisition is piggybacked onto the write using doorbell batching [23] to reduce network round trips. Once the lock is acquired, The client makes the insertion visible by atomically install the pointer to the new node in the parent node's slot with an `RDMA CAS`, followed by a piggybacked lock release using another `RDMA CAS`.

After the operation is completed, the client updates the succinct filter cache if the operation triggers a node split, where a new inner node with a new prefix is added to the index. The client updates only its local succinct filter cache, while the synchronization of caches on other CNs is deferred until their clients encounter the corresponding inner node during tree traversal, as described in the search procedure. Similarly, the inner node hash table is updated after a node type switch or a node split to reflect changes applied to the ART. This update can be performed atomically using an `RDMA CAS`, as the client modifies only one 8-byte hash entry.

Update (K, V). The client first retrieves corresponding leaf node following the search procedure. The structure of the leaf node is shown in Fig. 3. Similar to prior works [10], [22], the size of the leaf node is aligned to 64 B, and the unit of the $LeafLen$ field is 64 B. If the leaf node has sufficient space to store V (i.e., the required size is within $64 \times LeafLen$ B), the client performs an in-place update as described in Sec. III-C. Otherwise, the client performs an out-of-place update similar to the insert operation.

Delete (K). The client retrieves the leaf node following the search process and sets its status to $Invalid$ using an `RDMA CAS` operation. The client then updates corresponding slot of its parent node to NULL similar to the insert operation.

Scan (K_1, K_2). The client traverses the tree starting from the root node, retrieves all leaf nodes with keys between K_1 and K_2, and adds extracted key-value pairs to the result set.

V. Evaluation

A. Experimental Setup

Testbed. We emulate an RDMA-based DM cluster with three dual-socket machines, each equipped with 128GB of DRAM, two 2.6GHz Intel Xeon Gold 6348 CPUs (each with 28 cores), and one 2×100 Gbps dual-port Mellanox ConnectX-6 NIC. To save machine resources, we run both a CN and an MN on each machine, as in prior works [10], [27]. We manually limit the index cache memory usage on CNs and the CPU core usage on MNs to simulate a DM cluster.

Workloads. We evaluate Sphinx using the YCSB benchmark [28]. Specifically, we use six types of workloads: A (50% read, 50% update), B (95% read, 5% update), C (100% read), D (95% latest read, 5% update), E (95% Scan, 5% insert), and an additional LOAD workload (100% insert). By default, the workloads follow a zipfian key distribution with a skewness factor of 0.99. The size of the value for each key-value pair is set to 64 bytes.

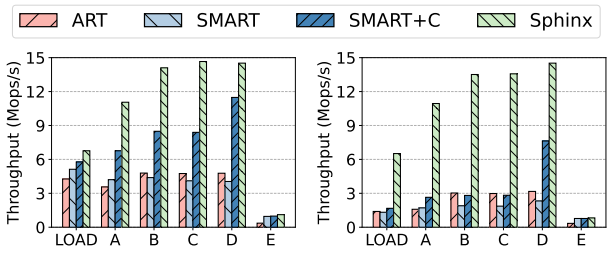
Datasets. Two datasets are used for evaluation: *u64* and *email*. The *u64* dataset consists of 8-byte fixed length integers generated from a uniform distribution, while the *email* dataset comprises publicly available email addresses [29] with sizes ranging from 2 to 32 bytes (an average size of 18.93 bytes).

Comparisons. We compare Sphinx with *SMART* [10], the state-of-the-art range index supporting variable-length keys on DM. Additionally, We port the original *ART* to DM and use it as a baseline. Similar to Sphinx and SMART, *ART* also adopts one-sided RDMA operations to access and modify the index and data stored in remote memory. The CN-side cache size of SMART and Sphinx is set to 20 MB, which is 4.2% of the u64 dataset size and 1.8% of the email dataset size. To further demonstrate the high performance of Sphinx with limited cache size, we also compare Sphinx with *SMART+C*, which has a CN-side cache size of 200 MB (41.7% and 17.6% of the u64 and email datasets, respectively).

B. Overall Performance

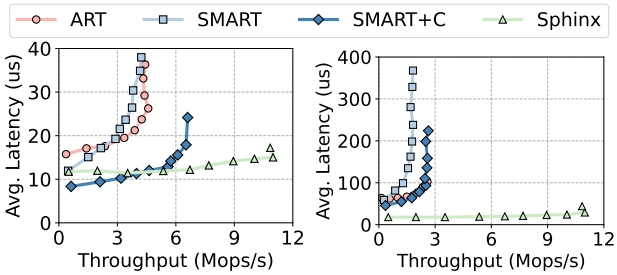
In this section, we evaluate the overall performance of Sphinx with the YCSB benchmark. Fig. 4 displays the throughput of the systems under the YCSB benchmark using the *u64* and *email* datasets.

Base Operations (YCSB A-D, LOAD). As shown in Fig. 4, Sphinx achieves 1.2–3.6× higher throughput on the u64



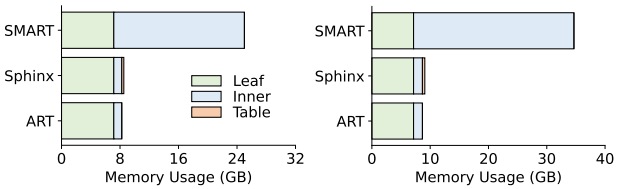
(a) u64 (b) email

Fig. 4: The throughput under the YCSB benchmark.



(a) u64 (b) email

Fig. 5: The scalability in the write-intensive YCSB-A workload.



(a) u64 (b) email

Fig. 6: The memory usage across different datasets

dataset and 1.9–7.3× higher throughput on the email dataset than SMART, SMART+C and ART. By combining the *Inner Node Hash Table* and *Succinct Filter Cache*, Sphinx effectively minimizes network round trips and bandwidth consumption during index operations by replacing the sequential tree traversal with a single hash entry read. This is especially beneficial for the email dataset, where larger key sizes and deeper tree structure exacerbate traversal costs. In contrast, SMART and ART suffer from excessive network round trips during index operations, leading to lower throughput. Furthermore, Sphinx outperforms SMART+C with only 10% of its CN-side cache, demonstrating the efficiency of the *Succinct Filter Cache* in achieving high performance with limited CN-side memory space. Finally, Sphinx maintains consistent performance across both u64 and email datasets, indicating its suitability for variable-length keys.

Range Query (YCSB E). For both the u64 and email datasets, Sphinx, SMART and SMART+C achieve 2.3–3.1× higher throughput than ART, primarily due to the use of the doorbell batching mechanism [23], which hides the latency associated with reading nodes. The performance of Sphinx, SMART and SMART+C is comparable, as the main bottleneck of the workload lies in reading the leaf nodes rather than in tree traversal.

C. Scalability

In this section, we evaluate the scalability of the systems using the YCSB-A workload (50% read, 50% update, skewed) with both the *u64* and *email* datasets. Following prior works [10], [27], [30], we adopt coroutines to hide latency associated with polling the completion queue across all systems, with each coroutine treated as a worker. Fig. 5 presents the throughput-latency curve of the systems under the YCSB-A workload with various numbers of workers in CNs (6–192 workers, evenly distributed across 3 CNs). Sphinx scales well as the number of workers increases in CNs, achieving up to 2.6× higher throughput and 2.2× lower average latency on the u64 dataset, and up to 6.1× higher throughput and 11.7× lower average latency on the email dataset. This performance gain is primarily attributed to the inner node hash table and succinct filter cache, which minimize network overhead in terms of both round trips and bandwidth consumption, thereby preventing premature saturation of network resources.

D. Space Consumption

In this section, we evaluate the MN-side space consumption of the systems across different datasets. We insert 60 million key-value pairs into each system and measure the resulting MN-side memory usage. As shown in Fig. 6, the inner node hash table incurs only a slight increase in memory consumption compared to the original ART, with a 3.3% increase for the u64 dataset and a 4.9% increase for the email dataset. In contrast, SMART consumes 2.1–3.0× more memory than the original ART, as it preallocates a space of Node-256 for each inner node to prevent potential cache coherence problems.

VI. Conclusion

In this paper, we propose Sphinx, a novel hybrid index supporting variable-length keys on DM. Sphinx introduces an *Inner Node Hash Table* to break the serial dependency in ART, offering an opportunity to mitigate the excessive network round trips during tree traversal with parallel hash entry reads. Sphinx further employs a *Succinct Filter Cache* to significantly reduce network bandwidth usage during index operations while keeping the CN-side cache succinct and coherent. Our evaluations demonstrates that Sphinx outperforms existing counterparts by up to 7.3× in YCSB benchmark.

Acknowledgment

We sincerely thank the anonymous reviewers for their constructive comments and suggestions. This work is supported by National Key Research and Development Program of China (No. 2022YFB4500303), National Natural Science Foundation of China (NSFC) (No. 62332012, 62227809, 62302290), the Fundamental Research Funds for the Central Universities, Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0102), and Natural Science Foundation of Shanghai (No. 22ZR1435400).

REFERENCES

[1] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "Tpp: Transparent page placement for cxl-enabled tiered-memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 742–755.

[2] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 649–667.

[3] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, "AIFM: High-Performance, Application-Integrated far memory," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 315–332.

[4] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 417–433.

[5] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee, "Mind: In-network memory management for disaggregated data centers," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 488–504.

[6] InfiniBand Trade Association, "InfiniBand Trade Association," 2024, accessed: 2024-11-19. [Online]. Available: https://www.infinibandta.org/

[7] Compute Express Link, "About CXL™," https://www.computeexpresslink.org/about-cxl, 2024, accessed: 2024-11-06.

[8] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: Artful indexing for main-memory databases," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 38–49.

[9] V. Leis, F. Scheibner, A. Kemper, and T. Neumann, "The art of practical synchronization," in *Proceedings of the 12th International Workshop on Data Management on New Hardware*, 2016, pp. 1–8.

[10] X. Luo, P. Zuo, J. Shen, J. Gu, X. Wang, M. R. Lyu, and Y. Zhou, "SMART: A High-Performance adaptive radix tree for disaggregated memory," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 553–571.

[11] S. Ma, K. Chen, S. Chen, M. Liu, J. Zhu, H. Kang, and Y. Wu, "ROART: range-query optimized persistent ART," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 1–16.

[12] S. K. Lee, K. H. Lim, H. Song, B. Nam, and S. H. Noh, "WORT: Write optimal radix tree for persistent memory storage systems," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017, pp. 257–270.

[13] A. Zeitak and A. Morrison, "Cuckoo trie: Exploiting memory-level parallelism for efficient dram indexing," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 147–162.

[14] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 75–88.

[15] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces," in *Proceedings of the international symposium on quality of service*, 2019, pp. 1–10.

[16] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, "Can far memory improve job throughput?" in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[17] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202–215.

[18] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[19] NVIDIA, "Connectx-6 ethernet smartnic," https://www.nvidia.com/en-sg/networking/ethernet/connectx-6/, 2024, accessed: 2024-11-04.

[20] D. R. Morrison, "Patricia – practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

[21] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, pp. 654–663.

[22] P. Zuo, J. Sun, L. Yang, S. Zhang, and Y. Hua, "One-sided RDMA-Conscious extendible hashing for disaggregated memory," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 15–29.

[23] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 437–450.

[24] F. J. Corbato, *A paging experiment with the multics system*. Massachusetts Institute of Technology, 1968.

[25] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

[26] B. Fan, D. G. Andersen, and M. Kaminsky, "MemC3: Compact and concurrent MemCache with dumber caching and smarter hashing," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX Association, Apr. 2013, pp. 371–384. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/fan

[27] Q. Wang, Y. Lu, and J. Shu, "Sherman: A write-optimized distributed b+ tree index on disaggregated memory," in *Proceedings of the 2022 international conference on management of data*, 2022, pp. 1033–1048.

[28] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[29] Fonxat, "300 million email database," https://archive.org/details/300MillionEmailDatabase, 2018, accessed: 2024-11-05.

[30] X. Wei, R. Chen, and H. Chen, "Fast RDMA-based ordered Key-Value store using remote learned cache," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 117–135. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/wei

An Energy-Efficient High-Utilization Hardware Architecture for Attention Mechanism in Transformer using Balanced Systolic Array and Multi-Row Interleaved Operation Ordering

Haiyang Zhou[1,2], Hongyang Hu[1*], Jinshan Yue[1], Hanghang Gao[1,2], Yuanlu Xie[1], Xiaoxin Xu[1], Chunmeng Dou[1], Ming Liu[3]

[1]*Key Laboratory of Fabrication Technologies for Integrated Circuits, Institute of Microelectronics of Chinese Academy of Sciences, Beijing, China,* [2]*University of Chinese Academy of Sciences, Beijing, China,* [3]*Frontier Institute of Chip and System, Fudan University, Shanghai, China*

[*]*Corresponding author: huhongyang@ime.ac.cn*

***Abstract*—Transformer-based neural networks have achieved remarkable performance. Designing energy-efficient and high-speed accelerators for the attention mechanism, which dominates the energy and latency in Transformers, has become increasingly significant. Existing attention accelerators commonly use algorithm-hardware co-design to achieve higher energy efficiency and speed. However, deeply customized algorithms make these accelerators dependent on a particular application. Therefore, optimizing hardware architecture is crucial for achieving general-purpose acceleration. We observe two limitations in the hardware architecture of existing attention accelerators. First, the widely used input stationary, weight stationary, and output stationary systolic arrays (SAs) can't balance data reuse, register saving, and utilization, which hinders to build more energy-efficient and faster SA-based accelerators. Second, layer-by-layer operation ordering introduces high SRAM access overhead of intermediate results. To address the first limitation, we propose the "Balanced Systolic Array", which improves energy efficiency by 40% compared to conventional systolic arrays and achieves a utilization rate of 99.5%. To address the second limitation, we propose "Multi-Row Interleaved" operation ordering, which reduces the SRAM energy by 31.7%. By integrating two techniques, the proposed attention accelerator achieves a 39% improvement in energy efficiency and a 38% enhancement in throughput×energy efficiency compared to previous works.**

I. Introduction

Transformer-based neural networks have demonstrated superior model performance over RNNs and CNNs in fields such as natural language processing[1][2][3] and computer vision[4]. In Transformers, the attention mechanism dominates the energy and latency[5][6][7]. Therefore, designing energy-efficient and high-speed accelerators for the attention mechanism has become increasingly significant. The computation process of the attention mechanism is shown in Fig.1, where n represents the token length, and d represents the word embedding dimension. The attention score matrix S is obtained by multiplying Q with K^T. After S is dequantized, scaled, processed with Softmax, and quantified, the attention probability matrix P is generated. Finally, P is multiplied by the V to generate the output matrix Z. The data formats for matrix

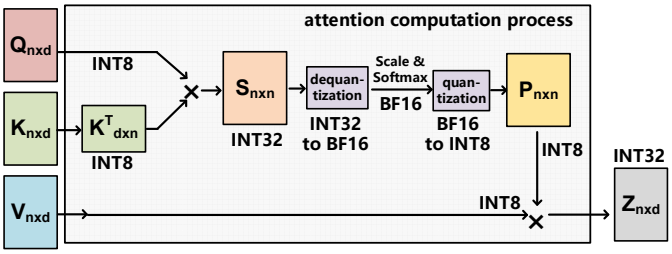


Fig. 1. Computation process of attention mechanism.

multiplication(MM), accumulation, Softmax are INT8, INT32, and BF16, respectively[17]. Existing attention accelerators commonly use algorithm-hardware co-design to achieve higher energy efficiency and speed. [7][9][10][11][12] exploit matrix P sparsity to introduce sparse computing, while [5][6][8] exploit the weak relevances token to introduce approximate computing. However, deeply customized algorithms tie them to a particular application[13]. Therefore, focusing on hardware architecture optimizations alone is vital for general-purpose acceleration.

We observed two hardware architecture limitations in existing attention accelerators, which introduce new challenges. As shown in Fig.2, **the first limitation** is the widely used input systolic (IS), weight stationary (WS), and output stationary (OS) SAs fail to balance data reuse, register saving, and utilization. Data reuse and register saving affect energy efficiency, as they affect the energy of memory[15], and register write energy[16]. Utilization impacts speed, as it affects throughput[13]. Therefore, the limitation restricts to build more energy-efficient and higher-speed SA-based accelerators. Specifically, for data reuse, OS-SA maximizes data reuse by using the outer product for MM[10][18], while IS-SA (or WS-SA) has lower data reuse, as it uses the inner product for MM. For register saving, OS-SA requires registers for accumulation and systolic, whereas IS-SA and WS-SA only need registers for systolic. For utilization, SAs require preloading data (IS-SA or WS-SA), offloading results (OS-SA), and filling pipeline(IS-SA, WS-SA, OS-SA), all of which introduce idle cycles that reduce utilization.

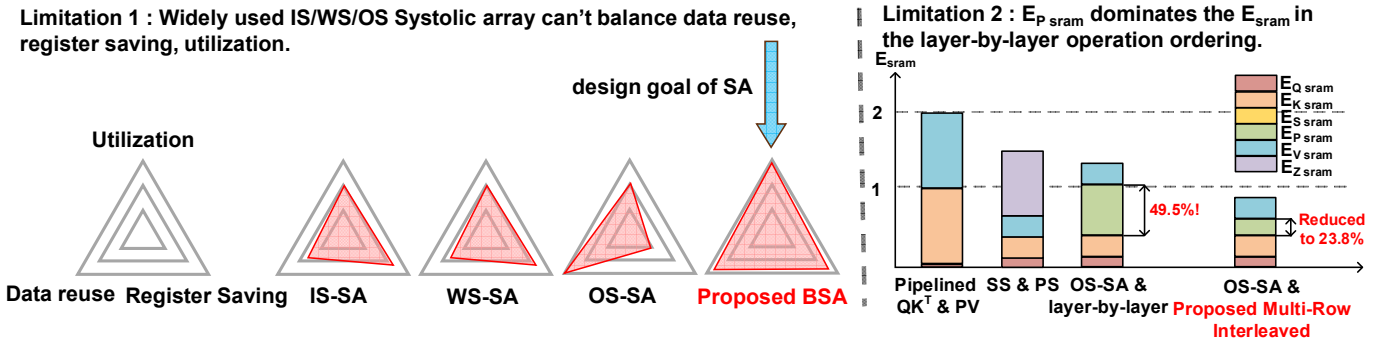


Fig. 2. Two Limitations of hardware architecture in existing attention accelerators.

Therefore, designing an SA that balances data reuse, register saving, and utilization introduces the first challenge.

The second limitation is the layer-by-layer operation ordering introduces significant SRAM access overhead of intermediate results. The right side of Fig.2 shows the total SRAM energy profile of the previous attention accelerator with different architectures(E_{sram}, unit: J, $E_{sram}=E_{Q\ sram}+E_{K\ sram}+E_{S\ sram}+E_{P\ sram}+E_{V\ sram}+E_{Z\ sram}$). The first architecture pipelines $Q\times K^T$ and $P\times V$(pipelined QK^T & PV)[5][6][7]. Although this architecture eliminates SRAM access of intermediate results ($E_{S,P\ sram}=0$), it weakens the spatial reuse of $Q\times K^T$ and $P\times V$ since the multipliers are split into two arrays, resulting in the highest E_{sram}. The second architecture uses reconfigurable SA which configures OS-SA when $Q\times K^T$ and IS-SA when $P\times V$(S stationary & P stationary, SS & PS)[8][10]. Although it not only avoids multiplier splitting but also eliminates SRAM access of intermediate results, the shape of the array for optimal data reuse for IS-SA and OS-SA is different, and IS-SA introduces inner product that weakens data reuse. The third architecture uses OS-SA and layer-by-layer operation ordering to compute $Q\times K^T$ and $P\times V$(OS-SA & layer-by-layer)[12]. Although it not only avoids multiplier splitting but also remains the optimal shape of the array for data reuse, the layer-by-layer operation ordering introduces large energy of SRAM access of intermediate results($E_{P\ sram}$ accounts for 49.5%). Therefore, designing an operation ordering to reduce $E_{P\ sram}$ introduces the second challenge. To address the above challenges, we made three contributions:

- We propose the "Balanced Systolic Array" to address the first challenge, which balances data reuse, register saving, and utilization, further improving energy efficiency and speed of SA.
- We propose a mathematical formula to determine the optimal data reuse for OS-based arrays with the inner-outer-mixed product.
- We propose the "Multi-Row Interleaved" operation ordering to address the second challenge, which significantly reduces $E_{P\ sram}$, further improving energy efficiency for attention accelerators.

The rest of this paper is organized as follows: Section II presents the background on three approaches for matrix multiplication and systolic array. Section III details the proposed attention accelerator architecture, including the "Balanced Systolic Array" and "Multi-row Interleaved" operation ordering. Section IV compares different systolic arrays and architecture of attention accelerators in terms of energy, energy efficiency, and utilization. Section V concludes the paper.

II. Background

A. Approaches for Matrix Multiplication(MM)

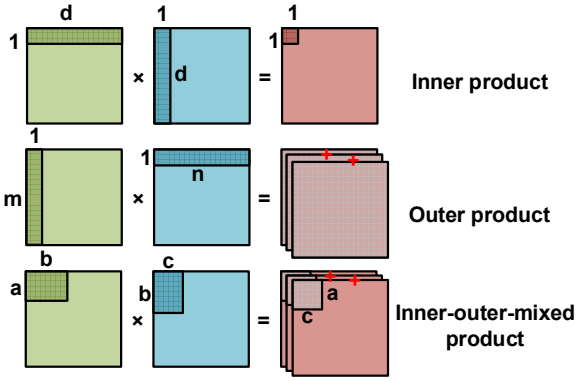


Fig. 3. Three approaches for matrix multiplication computation.

Fig. 3 shows three approaches for MM of an m×d matrix and a d×n matrix. In the output matrix, intensified red represents the result, while gray represents partial sums (psum).

The inner product approach is characterized by obtaining an element of the output matrix by taking the dot product of a row of the m×d matrix with one column of the d×n matrix. In hardware implementation, the inner product is characterized by two or more psums required for one output element to be generated in parallel. Outer product approach which is optimal in input reuse and output reuse, resulting in the highest data reuse[18], where a column of the m×d matrix is multiplied by a row of the d×n matrix to yield a psum matrix. Summing all psum matrices produces the final output matrix. In hardware implementation, the outer product is characterized by the psums required for one output element that are not generated in parallel, and the elements of both m×d matrix and d×n matrix are spatially reused by multiple multipliers. Inner-outer-mixed product approach generates a psum matrix block by multiplying a block from the m×d matrix and a block from the d×n matrix. Summing all corresponding psum matrix blocks yields the final output matrix. In hardware implementation, it has characteristics both of inner product and outer product.

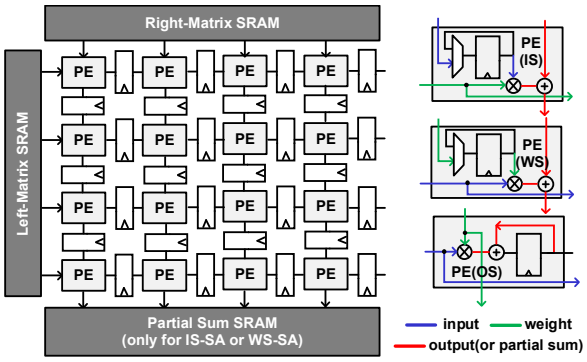


Fig. 4. Typical architecture of systolic array.

B. *Conventional Systolic Array*

Fig.4 shows a typical systolic array(SA) and specific PE structures of IS-SA, WS-SA, and OS-SA. SAs consist of many homogeneous processing elements (PEs), and data is transmitted between PEs through registers, thereby achieving spatial reuse. In IS-SA (or WS-SA), inputs (or weights) need to be preloaded into the registers of the PEs and then latched. After that, weights (or inputs) flow horizontally to the SA, while psums are propagated and summed vertically. The pipeline is filled when weights(or inputs) reach all PEs. In OS-SA, there is no preload process. Inputs flow horizontally to the OS-SA, weights flow vertically to the OS-SA, and each PE accumulates psum locally. Once inputs and weights reach all PEs, the pipeline is filled. The matrix multiplication result is offloaded through accumulators in OS-SA.

In our proposed systolic array, we combine the advantages of good data reuse of the outer product and register saving of the inner product by using the inner-outer-mixed product. We also combine the good register saving of broadcast and the good scalability of systolic. Furthermore, we improve the utilization of the SA by overlapping computation and offload.

III. Proposed Hardware Architecture

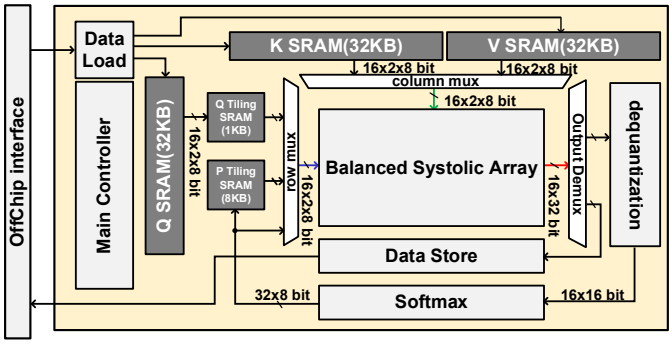


Fig. 5. Top-level of proposed attention accelerator architecture.

A. *Overview*

Fig. 5 shows the top-level architecture of the proposed attention accelerator. The memory mainly includes Q, K, V, Q Tiling, and P Tiling SRAMs. The Q, K, and V SRAMs store the Q, K, and V data, which are loaded by the Data Load module. The computational components consist of a balanced systolic array (BSA) and the Softmax module. The row mux and column mux (col mux) dynamically select the source data to feed into the BSA. The BSA contains 512 multipliers and 512 adders, with INT8 for the multipliers and INT32 for the accumulators. Two input matrices, represented by blue and green lines, are fed into the BSA, and the matrix multiplication results are output along the red line. The $Q \times K^T$ result S is fed into a dequantization module, while the $P \times V$ result Z is output via the Data Store module to the chip's I/O. The Softmax module has 16 exponent function units in BF16 format, which receives the data from the dequantization module and generates the P data. The Q Tiling SRAM and P Tiling SRAM enable data tiling for Q and P data, sourced from Q SRAM and the P data from the Softmax module. The $Q \times K^T$ and $P \times V$ are time-interleaved computed on the BSA.

B. *Proposed Balanced Systolic Array*

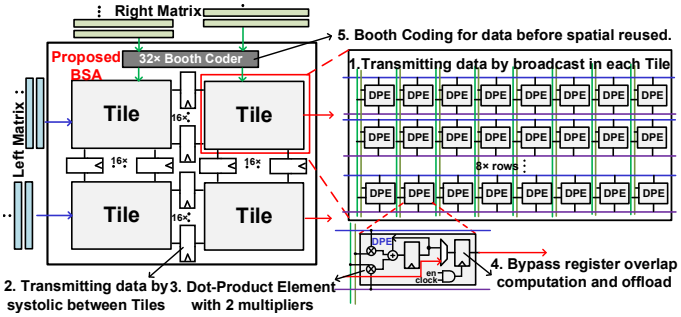


Fig. 6. The structure of the proposed Balanced Systolic Array.

Fig.6 shows the structure of the balanced systolic array which has three features.

Feature1: Inner-Outer-Mixed Product Broadcast Systolic Array (IOM-BSA, labeled 1, 2, 3 in Fig.6). IOM-BSA improves energy efficiency by balancing data reuse and register saving which is a variant of OS-SA. In IOM-BSA, each element of the left (or right) matrix is spatially reused by multipliers in the same row(or in the same column), which achieves outer product to obtain good data reuse. The array is divided into four tiles. In each tile, using broadcast to save registers for systolic and reduce systolic cycles which reduces pipeline filling cycles. Between tile and tile, using registers to transmit data, maintaining the scalability of the systolic array. Each tile contains an 8×8 array of Dot-Product Element (DPE) units. The dot-product operation achieves inner product to save accumulators compared to OS-SA. In summary, the shape of the array is set to 16×16 units, each with two multipliers. We have two considerations for setting the number of multipliers in DPE to 2.

Proposed mathematical formula: In order to determine the optimal data reuse for OS-based arrays with the inner-outer-mixed product, we discuss matrix multiplication $C_{m\times n}=A_{m\times d}B_{d\times n}$, and consider that the data format of the multiplier and accumulator are INT8, INT32, respectively. The SRAM for storing A, B, and Psum (or C) are denoted as A SRAM, B SRAM, and Psum SRAM. Therefore, if an array has

no spatial reuse at all, then it needs to access m×n×d×8 bits from A SRAM(read), m×n×d×8 bits from B SRAM(read), and 2×m×n×d×32 bits from Psum SRAM(write and read). In addition, we denoted the number of rows of the array as a (B is spatially reused a times), the number of multipliers in a unit as b (Psum is spatially reused b times), and the number of columns as c (A is spatially reused c times). Then when performing MM, the access of A SRAM is 8mnd/c bits. The access of B SRAM is 8mnd/a bits. The psum memory access is 0 since output-stationary. Therefore, the total SRAM access is:

$$\frac{8mnd}{a}+\frac{8mnd}{c} \quad (1)$$

Since a×b×c=number of multipliers, substituting abc=512 into the formula(1) leads to a basic inequality, as shown in formula(2).

$$\frac{8mnd}{a}+\frac{mnd}{64}ab \geq \frac{mnd\sqrt{b}}{\sqrt{2}} \left(when\ a=\frac{16\sqrt{2}}{\sqrt{b}}, c=\frac{32}{\sqrt{2}\sqrt{b}}\right) \quad (2)$$

Formula(2) shows that as b increases, data reuse decreases. Therefore, to maintain good data reuse, we prefer to select b=2. At the same time, we can determine the optimal shape of the array for data reuse is 16 rows and 16 columns.

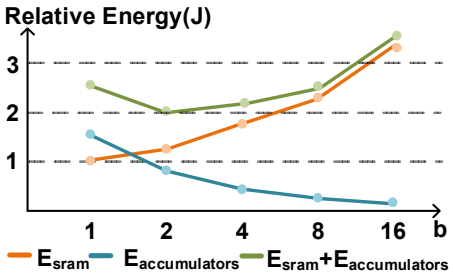


Fig. 7. Energy of SRAM and accumulators versus b.

As shown in Fig.7, to further balance data reuse and register saving, we evaluated the energy curve of SRAM (E_{sram}) and the energy curve of the accumulator ($E_{accumulator}$) as b increases while computing $Q\times K^T$. E_{sram} + $E_{accumulator}$ reaches its minimum value when b = 2. Therefore, we set two multipliers in each DPE based on formula(2) and Fig.7.

Feature 2: Bypass Register (labeled 4 in Fig.6) to balance utilization. Each DPE contains a bypass register with three behaviors: 1) Copying the value of the accumulator in each DPE, 2) Transmitting the bypass register value from the left DPE to the right DPE, and 3) Gating. The offloading task is accomplished through the bypass register by copying the accumulator result to the bypass register, allowing computation and offloading to overlap. Utilization is improved since the latency caused by offloading is hidden. The bypass register only works during the offload phase and is turned off via clock gating at other times. The additional register write energy due to the bypass register is negligible, as offload cycles are much less than computation cycles.

Feature 3: Booth Sharing (labeled 5 in Fig.6). Booth Sharing further improves energy efficiency. In conventional matrix multiplication arrays, each INT8 multiplier typically contains a Booth encoder circuit to implement a high-speed multiplier. We observed that the outer-product-based array shares the same operands across rows and columns, which introduces duplicate Booth coding. Therefore, we encode the operands in each column once and then spatially reuse the encoded operands, reducing the number of Booth encoder circuits. Booth Sharing decreases multiplier energy, improving energy efficiency. Specifically, we reduced the number of Booth encoder circuits from 512 to 32 through Booth sharing.

C. *Proposed Multi-Row Interleaved Operation Ordering*

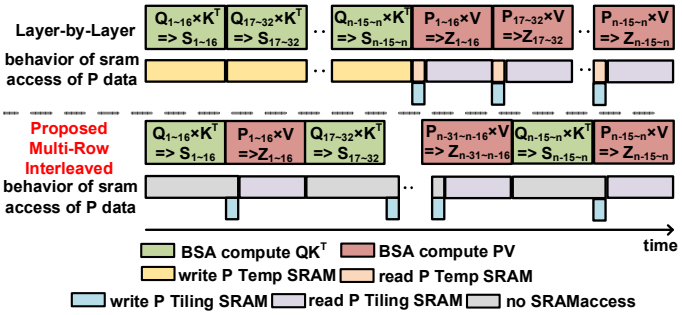


Fig. 8. Comparison between layer-by-layer and proposed Multi-Row Interleaved operation ordering.

Fig.8 shows the proposed Multi-Row Interleaved operation ordering and compares it with layer-by-layer operation ordering, where n is the token length. The P SRAM in Fig.2 includes both the P Temp SRAM and P Tiling SRAM in Fig.8.

Multi-Row Interleaved means that multiple rows of the S matrix and the Z matrix are processed alternately using the same array. Specifically, in the top-level architecture of Fig.5, the BSA is first used to compute the $Q\times K^T$, and the 16 rows of Q are written into the Q Tiling SRAM. The row mux selects the Q data from the Q Tiling SRAM, and the col mux selects the K data from the K SRAM. BSA computes $Q\times K^T$ and outputs the 16 rows of S, which enter the Softmax. When the computation of 16 rows of S is completed, The 16 rows of P from Softmax will also be ready. The row mux selects the P data from Softmax, and the col mux selects the V SRAM. Then, the BSA jumps from the $Q\times K^T$ to the P×V, and P data will be written to the P Tiling SRAM to provide the subsequent data for P Tiling. When the computation of 16 rows of Z is completed, the BSA jumps back from the P×V to the $Q\times K^T$. Therefore, $Q\times K^T$ and P×V are executed interleaved until the entire computation is completed. Multi-Row Interleaved significantly reduces the $E_{P\ sram}$, as it eliminates the access overhead of P Temp SRAM compared to layer-by-layer. Typically, the access overhead of P Temp SRAM is higher than that of P Tiling SRAM due to its larger capacity. Specifically, the capacity of P Temp SRAM and P Tiling SRAM are n×n×8 bits, and 16×n×8 bits, respectively.

IV. Evaluation

A. *Experiment Setup*

We evaluate the proposed hardware architecture and compare it with previous works in 28nm technology. The clock

TABLE I. SETTINGS OF SRAMS IN Q×K^T

SA / SRAM	Q	Q Tiling	K	S
IS-SA	32KB (2048×128)	none	32KB (1024×256)	32KB (512×512)
WS-SA	32KB (1024×256)	none	32KB (2048×128)	32KB (512×512)
OS-SA	32KB (2048×128)	1KB (64×128)	32KB (1024×256)	none
Proposed IOM-BSA	32KB (1024×256)	1KB (32×256)	32KB (1024×256)	none

TABLE II. SETTINGS OF SRAMS IN P×V

SA / SRAM	P	P Tiling	V	Z
IS-SA	256KB (16384×128)	none	32KB (1024×256)	4KB (64×512)
WS-SA	256KB (8192×256)	none	32KB (2048×128)	32KB (512×512)
OS-SA	256KB (2048×128)	8KB (512×128)	32KB (1024×256)	none
Proposed IOM-BSA	256KB (8192×256)	8KB (256×256)	32KB (1024×256)	none

frequency of all architectures is fixed at 500MHz and the power supply voltage is fixed at 0.9V. The energy consumption of the multipliers, adders, SRAMs, and registers is obtained using a 28nm Standard Cell Library, commercial SRAM compiler and Synopsys Design Compiler 2013. The utilization of different architectures is determined by dividing the ideal computing cycles by the actual computing cycles. To compare the energy efficiency fairly, we fixed the number of multipliers and adders to 512, and the data format of the multiplier and accumulator are INT8 and INT32. The evaluation model is BERT-base. For the BERT-based model, the token length of 512 can cover most applications [12], so evaluation under the token length n=512. TABLE I and II shows the settings of SRAMs for evaluating total energy of SRAM (E_{sram}) in different SAs, including capacity and shape. The format of the shape is depth×width. The array shape of IS/WS/OS-SA is set to 32×16[10][14][12], and all SAs use data tiling as much as possible to reduce the E_{sram}.

B. The Evaluation of Proposed BSA

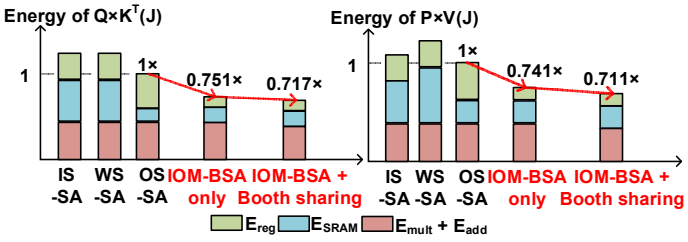


Fig. 9. Evaluation of the energy of BSA.

Fig.9 compared the energy of BSA to conventional SAs in Background. The energy of multiplier E_{mult} and the energy of adder E_{add} are equal since using the same data format and arithmetic circuit. In BSA, IOM-BSA and Booth sharing are techniques to improve energy efficiency. The E_{sram} of IOM-BSA is close to that of OS-SA and better than IS/WS-SA as it takes advantage of outer product. The total register energy E_{reg} of IOM-BSA is better than that of IS/WS/OS-SA since inner product and broadcast significantly reduce the overhead of accumulators and registers for systolic. IOM-BSA achieves a maximum energy reduction of 25.9% (0.741× in Fig.8) compared to the SA with the smallest total energy (E_{sram}+E_{reg}+E_{mult}+E_{add}). Booth sharing further reduces E_{mult}, lowering the total energy by an additional 4%. As the SA scales, larger arrays will amplify the benefits of Booth sharing.

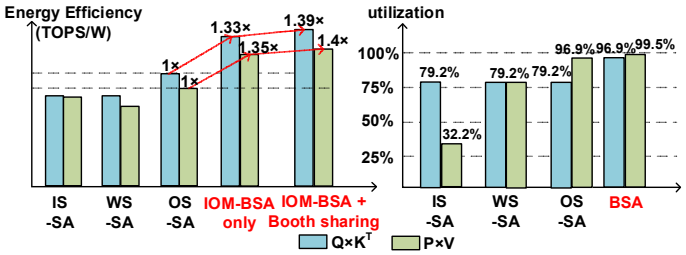


Fig.10. Evaluation of the energy efficiency and utilization of BSA.

The left side of Fig.10 compares the energy efficiency of BSA with conventional SAs. The proposed IOM-BSA achieves up to a 35% improvement in energy efficiency compared with OS-SA, with Booth sharing providing an additional 5% boost. The right side of Fig.9 compares the utilization of BSA with conventional SAs. The high utilization of the proposed BSA is attributed to two factors: 1) tiled broadcast reduces the number of systolic cycles, and 2) the bypass register overlaps offloading and computation, allowing BSA to maintain the highest utilization in both Q×K^T and P×V.

C. Comparison with Previous Hardware Architecture

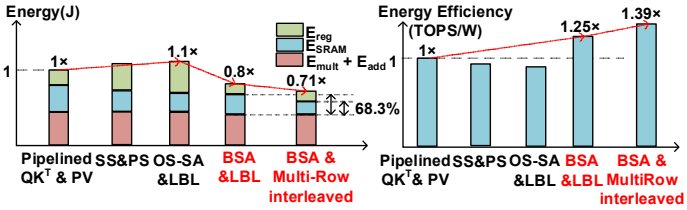


Fig.11. Evaluation of the energy of the proposed architecture.

The hardware architecture and Multi-Row Interleaved operation ordering are evaluated in this section. We compare five attention accelerator hardware architectures: three of them are pipelined QK^T & PV, SS & PS, and OS-SA & layer-by-layer (OS-SA & LBL) in Fig.2. The fourth architecture is BSA & LBL, used to evaluate the improvement brought by BSA, and the fifth architecture is BSA & Multi-Row Interleaved, used to evaluate the improvement brought by Multi-Row Interleaved. In the pipelined QK & PV architecture, we follow the design from [7], where Q×K^T uses an input stationary matrix-vector multiplication(IS-MVM) array with a shape of 64×4, and P×V uses a Broadcast-based outer product array with a shape of 4×64. The array shape of SS & PS is 32×16[8][10]. In the OS-SA & layer-by-layer(LBL), the shape of OS-SA is 32×16[12].

The left side of Fig.11 evaluates the energy of different architectures. OS-SA & LBL has the lowest E_{sram}, as both Q×K^T

and P×V use the outer product, but the overhead from a large number of registers results in the highest total energy. Although the E_{sram} of pipelined QK^T & PV architecture is the highest, it does not incur the overhead of registers for systolic, which greatly reduces E_{reg}, leading to lower total energy compared to SS & PS and OS-SA & LBL. BSA reduces total energy by 20% compared to the pipelined QK^T & PV architecture, which has the lowest energy in previous work. Multi-Row Interleaved eliminates the access overhead of P Temp SRAM, reducing E_{sram} to 68.3% of the E_{sram} of layer-by-layer. BSA & Multi-Row Interleaved reduces the total energy by 29%. The right side of Fig.11 evaluates the energy efficiency of different architectures. BSA improves the energy efficiency by 25%. Multi-Row Interleaved further improves the energy efficiency by 14%.

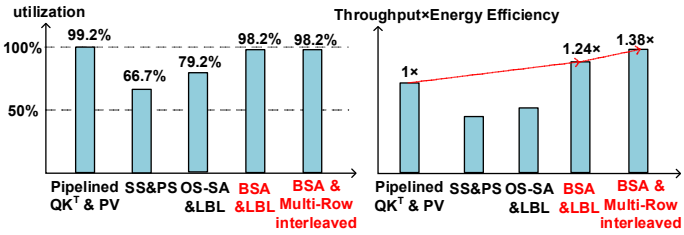


Fig. 12. Evaluation of utilization and throughput×energy efficiency of the proposed architecture.

The left side of Fig. 12 shows the utilization of the five architectures. The pipelined QK^T & PV achieves the highest utilization, reaching 99.2%. SS & PZ has the lowest utilization due to idle cycles of the multiplier caused by pipeline filling, which account for a significant proportion of the P×V. OS-SA & LBL also exhibit low utilization due to the low utilization of the OS systolic array. BSA & LBL can maintain a high utilization of 98.2% due to the high utilization of BSA. Multi-Row Interleaved does not reduce the utilization compared to layer-by-layer while increasing energy efficiency, as it does not introduce idle cycles. Consequently, BSA & LBL architecture and BSA & Multi-Row architecture Interleaved achieve the same high utilization. With improved overall utilization and energy efficiency, the proposed architecture achieves a 38% improvement in throughput×energy efficiency compared to the previous architecture.

V. Conclusion

This paper proposes a hardware architecture for the attention mechanism in Transformer, focusing on optimizing energy efficiency and utilization solely through hardware design. The proposed BSA balances data reuse, register saving, and utilization, achieving up to a 40% improvement in energy efficiency compared to conventional SAs while maintaining 99.5% utilization.The proposed Multi-Row Interleaved reduces E_{sram} by 31.7% and further enhances energy efficiency by 14%. By combining the two techniques, the attention accelerator proposed in this paper achieves a 39% improvement in energy efficiency and a 38% improvement in throughput×energy efficiency compared to previous works.

Acknowledgements This work was supported by the NSFC Grant 62404249, 92364202 and 62488101.

REFERENCES

[1] A. Vaswanet et al.,"Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Devlin et al.,"Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[5] T. J. Ham et al., "A^3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE HPCA*. IEEE, 2020, pp. 328–341.

[6] T. J. Ham et al., "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ISCA*. IEEE, 2021, pp. 692–705.

[7] H. Wang et al., "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 HPCA*. IEEE, 2021, pp. 97–110.

[8] Y. Wang et al., "An Energy-Efficient Transformer Processor Exploiting Dynamic Weak Relevances in Global Attention," *IEEE JSSC*, 2023.

[9] H. You et al., "ViTCoD: Vision Transformer Acceleration via Dedicated Algorithm and Accelerator Co-Design," in *2023 HPCA*. IEEE, 2023, pp. 273-286.

[10] L. Liu et al., "Sanger: A Co-Design Framework for Enabling Sparse Attention using Reconfigurable Architecture," in *2021 MICRO*. pp. 977-991.

[11] G. Shen et al., "SALO: An Efficient Spatial Accelerator Enabling Hybrid Sparse Attention Mechanisms for Long Sequences," in *2022 DAC*. pp. 571-576.

[12] Y. Qin et al., "FACT: FFN-Attention Co-optimized Transformer Architecture with Eager Correlation Prediction," in *2023 ISCA*. pp. 1-14

[13] Z. Wang et al., "COSA:Co-Operative Systolic Arrays for Multi-head Attention Mechanism in Neural Network using Hybrid Data Reuse and Fusion Methodologies," in *2023 DAC*. pp. 1-6.

[14] S. Shanmuga Sundaram et al., "FreFlex: A High-Performance Processor for Convolution and Attention Computations via Sparsity-Adaptive Dynamic Frequency Boosting," *IEEE JSSC*, 2023.

[15] Y. -H. Chen et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE JSSC*, 2017.

[16] X. Chen et al., "Communication Lower Bound in Convolution Accelerators," in *2020 HPCA*. pp. 529-541.

[17] Bita Rouhani et al., "Microscaling Data Formats for Deep Learning," arXiv preprint arXiv:2310.10537, 2023.

[18] Z. Zhang et al., "SpArch: Efficient Architecture for Sparse Matrix Multiplication," in *2020 HPCA*. pp. 261-274.

DyREM: Dynamically Mitigating Quantum Readout Error with Embedded Accelerator

Kaiwen Zhou
Zhejiang University
Hangzhou, China
kaiwenzhou@zju.edu.cn

Liqiang Lu*
Zhejiang University
Hangzhou, China
liqianglu@zju.edu.cn

Hanyu Zhang
Zhejiang University
Hangzhou, China
hyzz@zju.edu.cn

Debin Xiang
Zhejiang University
Hangzhou, China
db.xiang@zju.edu.cn

Chenning Tao
Zhejiang University
Hangzhou, China
tcn@zju.edu.cn

Xinkui Zhao
Zhejiang University
Hangzhou, China
zhaoxinkui@zju.edu.cn

Size Zheng
ByteDance Ltd
Beijing, China
zheng.size@bytedance.com

Jianwei Yin
Zhejiang University
Hangzhou, China
zjuyjw@zju.edu.cn

***Abstract*—Quantum readout error is the most significant source of error, substantially reducing the measurement fidelity. Tensor-product-based readout error mitigation has been proposed to address this issue by approximating the mitigation matrix. However, this method inevitably encounters the dynamic generation of the mitigation matrix, leading to long latency. In this paper, we propose DyREM, a software-hardware co-design approach that mitigates readout errors with an embedded accelerator. The main innovation lies in leveraging the inherent sparsity in the nonzero probability distribution of quantum states and calculating the tensor product on an embedded accelerator. Specifically, using the output sparsity, our dataflow dynamically downsamples the original mitigation matrix, which dramatically reduces the memory requirement. Then, we design DyREM architecture that can flexibly gate the redundant computation of nonzero quantum states. Experiments demonstrate that DyREM achieves an average speedup of $9.6\times \sim 2000\times$ and fidelity improvements of $1.03\times \sim 1.15\times$ compared to state-of-the-art readout error mitigation methods.**

I. Introduction

In the current Noisy Intermediate-Scale Quantum (NISQ) era, various sources of errors hinder the full realization of quantum computing advantage [1], [2]. Among these, readout error is a significant source of noise, with error rates ranging from 1% to 10% on state-of-the-art superconducting quantum processors [2]–[4]. Consequently, mitigating readout error is essential to ensure high fidelity in both NISQ quantum processors and future fault-tolerant quantum systems.

Several mitigation approaches have been proposed, including matrix-based mitigation [5]–[7], machine learning [8], and measurement subsetting [9]. Among these, matrix-based mitigation is widely used, employing matrix-vector multiplication (MVM) to improve measurement fidelity. However, this approach is computationally intensive as the mitigation matrix size grows exponentially with the number of qubits [10]. To address the memory issue, the tensor-product-based approach provides a more general formulation by decomposing the mitigation matrix into a tensor product of sub-mitigation matrices. Each sub-mitigation matrix is independently characterized based on the specific subset of qubits. This formulation avoids constructing and storing the full-dimensional mitigation matrix, making it inherently scalable.

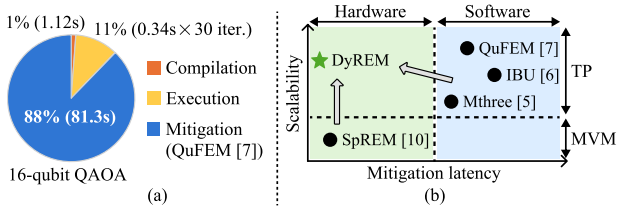


Fig. 1. (a) Breakdown of end-to-end latency for the 16-qubit QAOA on the 156-qubit ibm_fez quantum processor. (b) The tradeoff between scalability and latency for MVM-based method SpREM [10] and tensor-product-based (TP) methods Mthree [5], IBU [6], and QuFEM [7].

However, current tensor-product-based approaches still face limitations, including low accuracy, insufficient flexibility, and high latency. For example, Mthree [5] and IBU [6] calculate the mitigation matrix using 2×2 qubit-independent sub-mitigation matrices. However, these 2×2 sub-mitigation matrices fail to consider qubit crosstalk, severely reducing fidelity. To improve fidelity, QuFEM [7] groups physical qubits and utilizes grouping matrices to approximate mitigation matrix. However, QuFEM lacks the adaptability to varying measured qubits, as its grouping matrices are static and require predetermination. Furthermore, the inherently time-intensive nature of tensor product operations results in significant latency across these approaches. For example, as shown in Figure 1(a), the mitigation time of QuFEM accounts for 88% of the end-to-end latency for a 16-qubit QAOA circuit. Overall, as shown in Figure 1(b), current methods face a tradeoff between scalability and mitigation latency.

In this paper, we propose DyREM, a software-hardware co-design approach that dynamically mitigates readout errors with an embedded accelerator. The key innovation of DyREM lies in exploiting the inherent sparsity in the nonzero probability distribution of quantum states and calculating the tensor product on an embedded accelerator. To adapt to varying measured qubits, our dataflow dynamically downsamples the original mitigation matrix to obtain the required mitigation matrix. Then, we introduce nonzero state-oriented computation

*Corresponding author

to compress and calculate the mitigation matrix efficiently. Finally, we design DyREM architecture to support our dataflow, which can flexibly gate the redundant computation of nonzero quantum states.

The contribution of this paper can be summarized as follows:

- We propose DyREM, a software-hardware co-design approach that dynamically mitigates readout error with an embedded accelerator, leveraging the inherent sparsity in the nonzero probability distribution of quantum states.
- We introduce a dataflow that dynamically downsamples the original mitigation matrix, dramatically reducing the memory requirement. We further design DyREM architecture that flexibly gates the redundant computation of nonzero quantum states.
- We evaluate the effectiveness of DyREM by comparing it with state-of-the-art readout error mitigation methods. The results demonstrate the superiority of DyREM, achieving an average of $9.6\times \sim 2000\times$ speedup and $1.03\times \sim 1.15\times$ fidelity improvement.

II. Background

A. Matrix-Based Readout Error Mitigation

After the measurement operation, the readout error maps the ideal probability distribution P_{ideal} into a noise probability distribution P_{noisy}. This process can be mathematically represented as a linear transformation, where P_{ideal} is multiplied by a noise matrix. To mitigate readout error, we need to multiply P_{noisy} by a mitigation matrix M:

$$P_{miti} = M \cdot P_{noisy} \tag{1}$$

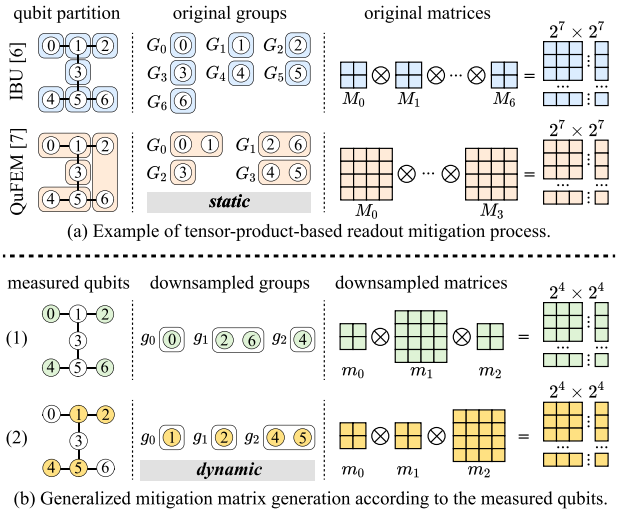
where the size of M is $2^n \times 2^n$ (n refers to the number of qubits). After the mitigation, the probabilities of the ideal states are expected to increase, while those of the noisy probabilities are reduced. However, the size of M scales exponentially with n, exhibiting poor scalability. For example, a 30-qubit mitigation matrix requires 32,768 PB of memory, which is $46.8\times$ the capacity of the world's fastest supercomputer Frontier (700 PB) [11].

B. Tensor-Product-Based Readout Error Mitigation

To address the memory issue, tensor-product-based readout error mitigation has been proposed. The key idea is to approximate the full-dimensional mitigation matrix M using a series of tensor-product operations [5]–[7]:

$$P_{miti} = (M_1 \otimes M_2 \otimes \cdots \otimes M_k) \cdot P_{noisy} \tag{2}$$

where $M_1, M_2, \cdots, M_k$ are sub-mitigation matrices obtained by partitioning the measured qubits into k groups. As illustrated in Figure 2(a), IBU [6] regards each physical qubit as an independent unit and approximates M using a tensor product of seven 2×2 mitigation matrices (i.e., $M_0 \sim M_6$). However, IBU achieves only limited mitigation accuracy because it neglects the measurement crosstalk between qubits.



(a) Example of tensor-product-based readout mitigation process.

(b) Generalized mitigation matrix generation according to the measured qubits.

Fig. 2. (a) Tensor-product-based readout mitigation process. (b) Generalized mitigation matrix generation by dynamically computing the downsampled groups.

To consider crosstalk effects, QuFEM [7] proposes a more general formulation by grouping physical qubits to approximate M accurately. As shown in Figure 2(a), seven qubits on the device are partitioned into four groups (i.e., $G_0 \sim G_3$) by quantifying the interactions between qubits. Each group is then characterized to produce its own mitigation matrix (i.e., $M_0 \sim M_3$). However, the groups and their mitigation matrices are ***static*** as they must be pre-determined before the mitigation process begins.

C. Motivation

In practical quantum circuit execution, the set of measured qubits varies due to differing levels of transpiler optimization, as shown in Figure 2(b). Consequently, the groups of measured qubits are dynamic, rendering the pre-determined mitigation matrices unsuitable for reuse during mitigation.

This limitation motivates us to develop a new approach to enable dynamic readout error mitigation. To this end, we introduce the concept of *downsampled group*, denoted by g_i. We dynamically determine the downsampled group g_i based on the measured qubits and original groups G_j. Then, we calculate the downsampled matrices m_i for g_i by deriving them from the original matrices M_j. Finally, we calculate the mitigation matrix based on the downsampled matrices m_i and perform the MVM step to complete mitigation.

III. DyREM Dataflow

A. Mitigation Dataflow Overview

Figure 3 illustrates an example of DyREM mitigation dataflow. The input consists of measured qubits and noisy distribution. The output is mitigated distribution P_{miti}. The dataflow is divided into three steps. **Step 1.** downsampled groups generation: we partition the measured qubits '①②④⑤' according to the original groups $G_0 \sim G_3$. Then, we obtain four downsampled groups $g_0 \sim g_3$, which serve as the input for matrix downsampling. **Step 2.** matrix downsampling: we categorize the downsampled groups

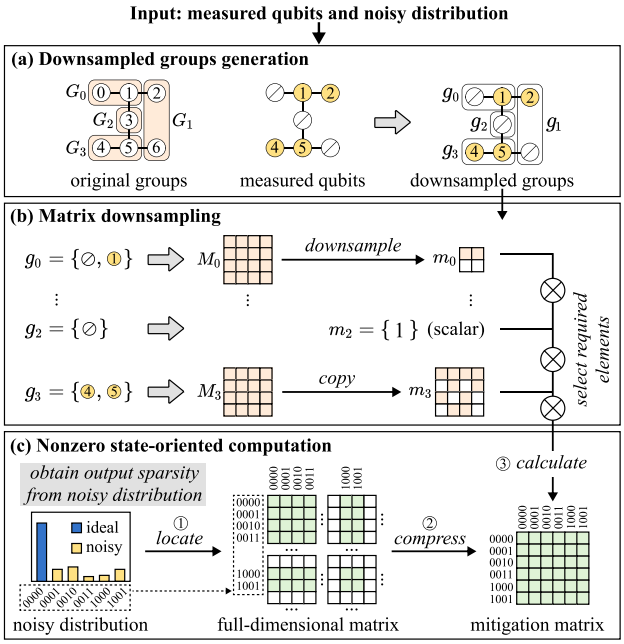


Fig. 3. Example of DyREM mitigation dataflow.

$g_0 \sim g_3$ into three types. Then, we adopt three strategies to generate the downsampled matrices $m_0 \sim m_3$, serving as the input for nonzero state-oriented computation. **Step 3.** nonzero state-oriented computation: we first obtain the output sparsity from the noisy distribution. Then, we compress the full-dimensional matrix based on the output sparsity. Next, we calculate the mitigation matrix by selecting the required elements from the downsampled matrices $m_0 \sim m_3$ and performing multiplication. Finally, we perform matrix-vector multiplication to mitigate the readout error.

In the following sections, we explain the details of **step 2** and **step 3**, respectively.

B. Matrix Downsampling

As depicted in Figure 3(a), some physical qubits are not measured (denoted by ⊘) in circuit execution. Therefore, the downsampled groups $g_0 \sim g_3$ are not identical to the original groups $G_0 \sim G_3$. To consider this variation, we categorize the downsampled groups into three types: (1) unmeasured (e.g., g_2), (2) fully measured (e.g., g_3), and (3) partially measured (e.g., g_0 and g_1). As shown in Figure3(b), we employ tailored strategies to generate downsampled matrices for each type of group. For 'unmeasured' g_2, we set its downsampled matrix to scalar '1', which does not influence the computational result. For 'fully measured' g_3, we directly copy the original matrix M_3 to obtain the downsampled matrix m_3 without additional computation. For 'partially measured' g_0 and g_1, we apply a convolution kernel to downsample the original matrix.

Figure 4 presents the downsampling of the original matrix M_0. Figure 4(a) illustrates the process of deriving the marginal probability distribution (MPD) of variable X from the joint probability distribution (JPD) of two variables X, Y. By taking cross sections of the JPD along the x-axis and performing integration, the probability distribution of variable X is obtained. This process inspires us to use convolution kernels to downsample the original matrix, as the original matrix M_0 essentially represents the JPD of discrete variables (e.g., qubits). Figure 4(b) presents the detailed process. To obtain the convolution kernel, we first generate the valid coordinates based on g_0, where the value at measured qubit ① is fixed at 0, and the value at ⊘ is filled sequentially from '00' to '11'. The values at the valid coordinates are set to 1, while all other positions are set to 0. Then, we extract the maximal sub-coordinate '10', convert it to decimal, and add one to obtain the kernel size. Next, we determine the stride based on g_0 by setting ⊘ to 0 and measured qubit ① to 1. We then convert '01' to decimal to obtain the stride (e.g., 1). Finally, we scan M_0 using the kernel and normalize each row to obtain the downsampled matrix m_0.

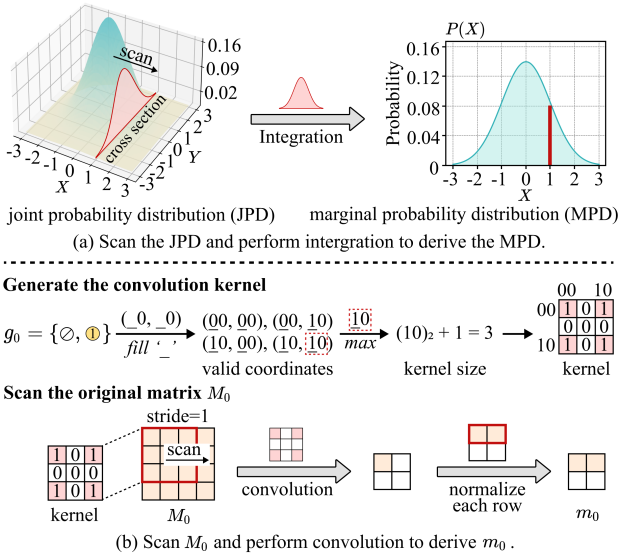


Fig. 4. Inspiration and an example of the matrix downsampling for partially measured groups.

C. Nonzero State-Oriented Computation

To enable on-chip calculation of the mitigation matrix, we propose nonzero state-oriented computation, comprising two steps: mitigation matrix compression and nonzero state similarity detection.

Mitigation matrix compression. A straightforward approach to obtain the mitigation matrix is to compute the tensor product of the downsampled matrices m_0 to m_3. However, this approach is time-consuming and resource-intensive. We observe that the ideal distribution usually lies in the noisy distribution, which only contains a few nonzero values. Therefore, we exploit this output sparsity from the noisy distribution to compress the mitigation matrix. By confining the mitigation process to the noisy space, we reduce the mitigation matrix size from exponential to linear. For example, in Figure 3(c), the mitigation matrix size is reduced from 16×16 to 6×6 based on the output sparsity.

Nonzero state similarity detection. After compression, we cannot simply perform a tensor product of downsampled matrices to calculate the mitigation matrix. Figure 5 shows

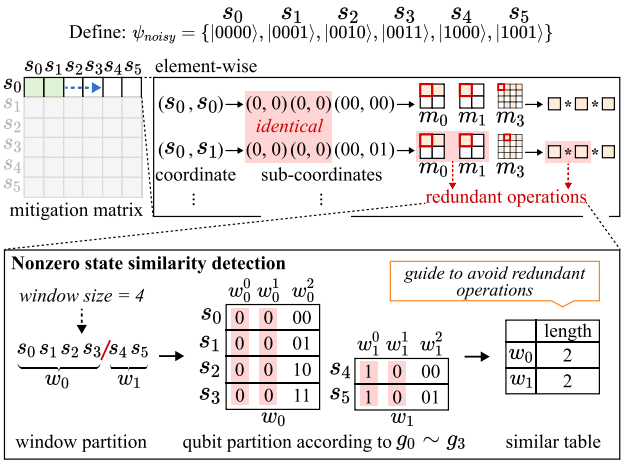


Fig. 5. The element-wise calculation process of mitigation matrix and the illustration of nonzero state similarity detection.

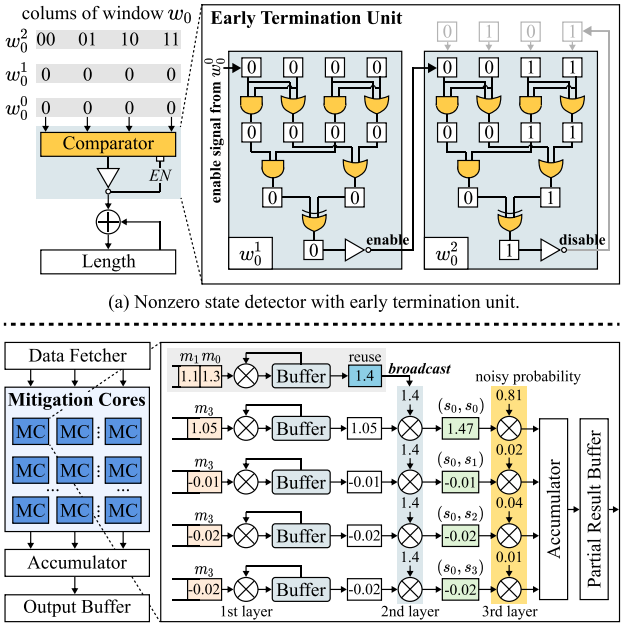
a straightforward approach to compute the mitigation matrix elements individually: for each element, its coordinates are divided into sub-coordinates based on the downsampled groups $g_0 \sim g_3$; the corresponding elements are then extracted from the downsampled matrices $m_0 \sim m_3$ and multiplied cumulatively to obtain the matrix element. However, we find that this approach involves many redundant operations. For example, the first two sub-coordinates of (s_0, s_0) and (s_0, s_1) are identical, which means the extraction and multiplication for (s_0, s_1) are redundant.

To eliminate redundant operations, we introduce nonzero state similarity detection. Specifically, we use a window to partition the noisy states sequentially. Note that we assume the noisy states are ordered according to their binary values. Then, we partition the qubits of noisy states according to the downsampled groups $g_0 \sim g_3$. Within each window, we identify the longest identical segment and obtain its length to construct a similar table. We can generate this table once we receive the noisy distribution. Then, we utilize this table to guide the calculation of the mitigation matrix, thereby eliminating redundant operations. Accordingly, we compute each row of the mitigation matrix window-wise rather than element-wise.

Figure 5 shows the process of computing a similar table for a mitigation matrix. First, we set the window size to 4. Then, the noisy states are partitioned into two windows (i.e., w_0 and w_1). In each window, the noisy states are partitioned into three columns according to the downsampled groups $g_0 \sim g_3$. Clearly, in w_0, the largest identical segment is (w_0^0, w_0^1), while in w_1, it is (w_1^0, w_1^1). Next, we record the length of these two segments in a similar table (i.e., 2 and 2). After applying the nonzero state similarity detection, we reduce the extraction and multiplication operations required for 6×6 mitigation matrix calculation by 44% and 33%, respectively.

IV. Architecture Design

We design an accelerator architecture to implement DyREM, primarily consisting of two components: a nonzero state detector and a mitigation core array. The nonzero state



(a) Nonzero state detector with early termination unit.

(b) The mitigation module with dedicated mitigation core.

Fig. 6. The hardware design details of DyREM.

detector efficiently calculates the length of the longest identical segment for each window. The mitigation core array receives the values from downsampled matrices and noisy probabilities. Each mitigation core (MC) employs a three-layer multiplier to integrate the mitigation matrix calculation with the MVM step.

A. Nonzero State Detector

As shown in Figure 6(a), the nonzero state detector integrates an early termination unit with an adder. The early termination unit consists of a comparator and a NOT gate. The comparator receives the columns of each window and determines whether the bits in each column are identical. The comparator consists of a series of logic gates (AND, NOT, and XOR gates), forming a hierarchical tree. If the comparator outputs '0', it indicates that all bits in the column are identical; otherwise, they are not. Specifically, the output then passes through a NOT gate to generate an enable signal, indicating whether the subsequent comparison should proceed. In other words, if the bits in the previous column are not identical, there is no need to perform further comparisons. After each comparison, the adder receives the output from the NOT gate, accumulating them to obtain the identical table.

Figure 6(a) shows an example of the nonzero state detector operating on window w_1. The comparator first receives the bits in w_0^0 and determines they are identical, enabling the comparison of w_0^1. Meanwhile, the length increases by one. Similarly, the comparison result of w_0^1 enables the comparison of w_0^2, while incrementing the length to 2. Since each element in w_0^2 contains two bits, we divide w_0^2 into two parts and sequentially input them into the comparator. As shown in Figure 6(a), the first part is not identical. Thus, the output of the comparator is 1, disabling the comparison of the second part. Consequently, the length is incremented by 0. Finally, the length is 2, indicating w_1 has two identical columns.

TABLE I
THE HARDWARE PERFORMANCE COMPARISON.

Baseline	Technical feature	VQE [12]		QAOA [13]		DJ [14]	
		Latency (s)	Q-throughput (states/s)	Latency (s)	Q-throughput (states/s)	Latency (s)	Q-throughput (states/s)
Mthree [5]	Hamming pruning	2.52 (384×)	7.27×10^3 (583×)	4.22 (461×)	4.63×10^3 (758×)	0.66 (206×)	2.77×10^4 (192×)
SpREM [10]	HDSR format	0.48 (73.8×)	1.76×10^6 (2.4×)	0.56 (61.5×)	1.49×10^6 (2.3×)	0.031 (9.6×)	3.43×10^6 (1.5×)
IBU [6]	Bayesian unfolding	13.0 (2000×)	3.58×10^5 (11.8×)	17.1 (1879×)	2.72×10^5 (12.9×)	4.59 (1437×)	1.01×10^6 (5.2×)
QuFEM [7]	Finite element analysis	11.7 (1800×)	1.56×10^3 (2726×)	13.5 (1483×)	1.45×10^3 (2420×)	5.42 (1687×)	2.65×10^3 (2002×)
DyREM	Redundancy detection	$\mathbf{6.52 \times 10^{-3}}$	$\mathbf{4.24 \times 10^6}$	$\mathbf{9.12 \times 10^{-3}}$	$\mathbf{3.51 \times 10^6}$	$\mathbf{3.25 \times 10^{-3}}$	$\mathbf{5.31 \times 10^6}$

B. Mitigation Core

As depicted in Figure 6(b), the mitigation module comprises a data fetcher, a mitigation core array, an accumulator, and an output buffer. The data fetcher receives the similar table and the noisy distribution. Then, it extracts the values from the downsampled matrices and passes them to the mitigation core array. We allocate 4KB of on-chip SRAM for the downsampled matrices to support quick random accesses, with each element stored in FP16 format.

Each MC is responsible for a single window, including calculating the mitigation matrix elements and performing the mitigation. Within the MC, we set up a FIFO for each multiplier (1st layer) to stockpile the values from downsampled matrices. In particular, the FIFO in the first row is used to calculate the reused data. Each remaining FIFO corresponds to the calculation of a mitigation matrix element. After obtaining the reused data, we broadcast it to the multipliers in the 2nd layer. Then, we perform multiplication to calculate the mitigation matrix elements within the window. Finally, we execute mitigation by multiplying noisy probabilities and the mitigation matrix elements. The accumulator receives the results from each multiplier and sends the accumulated result to the partial result buffer. The results of the MCs within a row are accumulated to yield one mitigated probability.

Figure 6(b) illustrates an example of the MC processing w_0. The values 1.1 and 1.3 correspond to the sub-coordinates $(0, 0)$ and $(0, 0)$ of (s_0, s_0) in Figure 5, respectively. The multiplication result 1.4 is reused to compute mitigation matrix elements within w_0. After obtaining the values of $(s_0, s_0) \sim (s_0, s_3)$, we perform multiplication with noisy probabilities and accumulate the results to obtain the mitigated result of w_0.

V. EVALUATION

A. Evaluation Setup

Hardware implementation. We develop DyREM accelerator using Xilinx High-Level Synthesis (HLS) C++ and implement it with Vitis 2023.1. We evaluate its hardware performance on the Xilinx U50 FPGA, operating at 300 MHz.

Benchmarks. The benchmarks include Variational Quantum Eigensolver (VQE) [12], Quantum Approximate Optimization Algorithm (QAOA) [13], Feedback-Based Quantum Optimization (FALQON) [15], and Deutsch-Jozsa (DJ) [14] algorithms, with two layers set for both VQE and QAOA.

Baselines. We compare DyREM with state-of-the-art readout error mitigation methods, including Mthree [5], SpREM [10], IBU [6] and QuFEM [7]. For Mthree, SpREM, and IBU, we set the Hamming distance to 3. The convergence tolerance of IBU is set to 10^{-4}. The iteration number and pruning threshold of QuFEM are set to 2 and 10^{-4}. The window size of DyREM is set to 4 to achieve a high data reuse rate.

Evaluation platform. We perform error mitigation experiments on the 136-qubit Quafu [16] quantum platform, where the readout error rate is 4.5%. We utilize QuFEM [7] to perform qubit partition and mitigation matrix characterization, with the number of qubits in a group set to 2. We implement Mthree [5] and IBU [6] on NVIDIA A100 GPU (using `cuBLAS` v12.4 and `JAX-GPU` v0.4.13, respectively). QuFEM is a software-friendly approach, and we implement it on an AMD EPYC 9554 64-core CPU (using `NumPy` v1.24.3). We implement SpREM accelerator on the Xilinx U50 FPGA using HLS C++, operating at 300 MHz.

B. Evaluation of Hardware Performance

Table I shows the average hardware performance comparison of DyREM with baselines, evaluated on the VQE, QAOA, and DJ algorithms, using 16, 20, 24, and 28 qubits.

Latency: DyREM achieves a geometric average speedup of $9.6\times \sim 2000\times$ over baselines. Such a high speedup compared to conventional computing platforms (CPU and GPU) is attributed to the specialized computational architecture, which effectively detects and avoids redundant operations while generating the mitigation matrix. Although SpREM [10] exploits the Hamming sparsity to avoid operations involving zero values, it fails to eliminate redundant computations based on the noisy distribution. In contrast, DyREM compresses the mitigation matrix based on the output sparsity and detects redundant operations, achieving a maximum speedup of $73.8\times$ in the VQE algorithm compared to SpREM.

Q-throughput: We define q-throughput, measured in states per second (states/s), as a metric to evaluate mitigation performance. The geometric average q-throughput of DyREM for VQE, QAOA, and DJ algorithms reaches 4.2×10^6 states/s, 3.5×10^6 states/s, and 5.3×10^6 states/s, representing an improvement over baselines of $2.4\times \sim 2726\times$, $2.3\times \sim 2420\times$, and $1.5\times \sim 2002\times$, respectively. In particular, DyREM achieves $2.4\times$, $2.3\times$, and $1.5\times$ mitigation performance improvements over the custom accelerator SpREM [10]. Rather than relying on external memory to obtain the mitigation matrix (i.e., SpREM), DyREM utilizes the downsampled matrices to calculate the mitigation matrix on-chip. This strategy

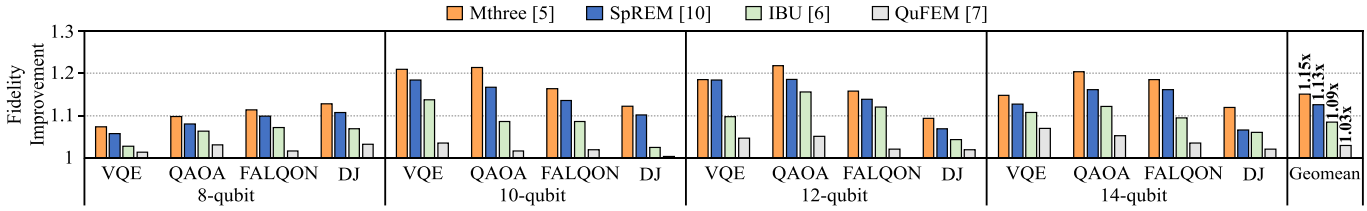
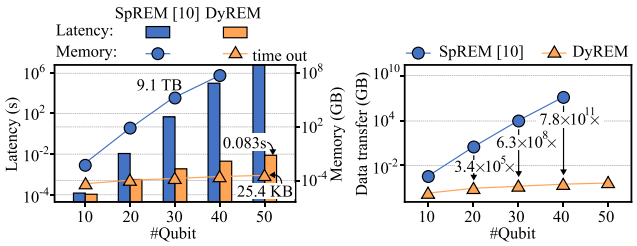


Fig. 7. Fidelity improvement over Mthree [5], SpREM [10], IBU [6], and QuFEM [7] on the benchmarks.



(a) Latency and memory usage of mitigating the DJ algorithm.

(b) Data transfer volume of mitigating the DJ algorithm.

Fig. 8. Comparison between SpREM [10] and DyREM on the DJ algorithm.

effectively reduces the bandwidth demand and improves the utilization of compute units, thereby enhancing throughput.

C. Evaluation of Fidelity

Although DyREM demonstrates outstanding hardware performance, it does not compromise on mitigation accuracy. Figure 7 illustrates the fidelity improvement of DyREM over the baselines. DyREM achieves an average fidelity improvement of 1.15×, 1.13×, 1.09×, and 1.03× over Mthree [5], SpREM [10], IBU [6], and QuFEM [7], respectively. The improvement comes from the fact that DyREM confines the mitigation space and utilizes the group matrices that consider the crosstalk. Conversely, Mthree and IBU use 2×2 mitigation matrices to construct the full-dimensional tensored mitigation matrix, failing to consider crosstalk between qubits. SpREM sets a large number of elements in the mitigation matrix to zero using Hamming distance, introducing system errors and reducing mitigation accuracy. Compared to QuFEM, DyREM confines the mitigation within the noise space, facilitating the convergence of noise probabilities towards ideal probabilities.

D. Comparison with SpREM

Figure 8 presents the mitigation performance comparison between SpREM [10] and DyREM on the DJ algorithm, with the number of qubits ranging from 10 to 50.

Latency and memory usage. Figure 8(a) shows the latency and memory usage comparison between SpREM [10] and DyREM. Based on nonzero state-oriented computation, DyREM achieves linear time and space complexity. For instance, in the 50-qubit DJ algorithm, DyREM requires only 0.083 seconds of latency and 25.4 KB of memory, whereas SpREM fails to output the mitigation results. The scalability of DyREM stems from the on-chip generation of the mitigation matrix, which circumvents the limitations imposed by finite bandwidth. In contrast, SpREM relies on reading the mitigation matrix from off-chip memory, where the matrix size grows exponentially with the number of qubits, making bandwidth a bottleneck that limits its fast mitigation.

Data transfer. Figure 8(b) compares the data transfer volume between SpREM [10] and DyREM. The data transfer volume of DyREM is linearly increased with the number of qubits, from 0.63 KB (10-qubit) to 37.8 KB (50-qubit), while SpREM exhibits exponential complexity. This is because DyREM only needs to retrieve the measured qubits and noisy distribution from off-chip memory and calculate the mitigation matrix on-chip. Conversely, the mitigation matrix for SpREM is too large to store on-chip (e.g., 9.1 TB for 30-qubit), leading to increased data transfer volume.

VI. Related Work

Matrix-based readout error mitigation. Mthree [5] and SpREM [10] exploit Hamming distance to compress the mitigation matrix. To implement scalable mitigation, IBU [6] utilizes a series of 2×2 qubit-independent sub-mitigation matrices to construct the mitigation matrix. To improve fidelity, CTMP [17] leverages Markovian noise models to mitigate correlated crosstalk errors. QuFEM [7] formulates the mitigation process as a tensor product using grouping matrices.

Other error mitigation techniques. Jigsaw [9] executes quantum programs in two models and employs a Bayesian post-processing step to mitigate measurement errors. HAMMER [18] combines the structure of quantum errors with a reclassification protocol to improve fidelity. Q-BEEP [19] introduces a Hamming spectrum model for characterizing localized and distant clustered Hamming errors. QuTracer [1] traces the states of qubit subsets to mitigate both gate and measurement errors.

VII. Conclusion

This paper proposes DyREM, a co-design approach that dynamically mitigates readout error with an embedded accelerator. The key innovation lies in exploiting the inherent sparsity in the nonzero probability distribution of quantum states and calculating the tensor product on an embedded accelerator. To adapt to varying measured qubits, our dataflow dynamically downsamples the original mitigation matrix. We then design DyREM architecture that flexibly gates the redundant computation of nonzero quantum states. DyREM achieves remarkable improvements compared to other readout mitigation methods.

Acknowledgment

This work was supported by National Key Research and Development Program of China (No. 2023YFF0905200) and Zhejiang Provincial Natural Science Foundation of China under Grant (No.LR25F020002).

REFERENCES

[1] P. Li *et al.*, "Qutracer: Mitigating quantum gate and measurement errors by tracing subsets of qubits," in *ISCA*, 2024, pp. 103–117.

[2] S. Maurya *et al.*, "Scaling qubit readout with hardware efficient machine learning architectures," in *ISCA*, 2023.

[3] IBMQ. (2024) Ibm quantum platform. [Online]. Available: https://quantum.ibm.com/

[4] Rigetti. (2024) Rigetti computing: Quantum computing. [Online]. Available: https://www.rigetti.com/

[5] P. D. Nation *et al.*, "Scalable mitigation of measurement errors on quantum computers," *PRX Quantum*, vol. 2, p. 040326, 2021.

[6] B. Pokharel *et al.*, "Scalable measurement error mitigation via iterative bayesian unfolding," *Phys. Rev. Res.*, vol. 6, p. 013187, 2024.

[7] S. Tan *et al.*, "Qufem: Fast and accurate quantum readout calibration using the finite element method," in *ASPLOS*, 2024, p. 948–963.

[8] B. Lienhard *et al.*, "Deep-neural-network discrimination of multiplexed superconducting-qubit states," *Physical Review Applied*, vol. 17, no. 1, p. 014024, 2022.

[9] P. Das *et al.*, "Jigsaw: Boosting fidelity of nisq programs via measurement subsetting," in *MICRO*, 2021, p. 937–949.

[10] H. Zhang *et al.*, "Sprem: Exploiting hamming sparsity for fast quantum readout error mitigation," in *DAC*, 2024.

[11] S. Atchley *et al.*, "Frontier: exploring exascale," in *SC*, 2023, pp. 1–16.

[12] A. Kandala *et al.*, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[13] E. Farhi *et al.*, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[14] D. Deutsch *et al.*, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[15] A. B. Magann *et al.*, "Feedback-based quantum optimization," *Physical Review Letters*, vol. 129, no. 25, p. 250502, 2022.

[16] B. A. of Quantum Information Sciences. (2022) Quafu quantum cloud computing platform. [Online]. Available: https://quafu.baqis.ac.cn/

[17] S. Bravyi *et al.*, "Mitigating measurement errors in multiqubit experiments," *Physical Review A*, vol. 103, no. 4, p. 042605, 2021.

[18] S. Tannu *et al.*, "Hammer: boosting fidelity of noisy quantum circuits by exploiting hamming behavior of erroneous outcomes," in *ASPLOS*, 2022, pp. 529–540.

[19] S. Stein *et al.*, "Q-beep: Quantum bayesian error mitigation employing poisson modeling over the hamming spectrum," in *ISCA*, 2023, pp. 1–13.

Cayman: Custom Accelerator Generation with Control Flow and Data Access Optimization

Youwei Xiao†, Fan Cui†, Zizhang Luo†, Weijie Peng†, and Yun Liang*†
†*School of Integrated Circuits, Peking University,* *Institute of Electronic Design Automation, Peking University*
shallwe@pku.edu.cn, pku_cf@stu.pku.edu.cn, {semiwaker, weijiepeng, ericlyun}@pku.edu.cn

***Abstract*—Custom accelerators enhance System-on-Chips' performance through hardware specialization. High-level synthesis (HLS) can automatically synthesize accelerators for given kernels but requires manual selection and extraction of kernels from applications. This paper proposes Cayman, the first end-to-end framework to synthesize high-performance custom accelerators with both control flow and data access optimization. Cayman automatically selects kernels for hardware acceleration based on a hierarchical program representation, which captures kernel candidates with general control flows. Besides, Cayman optimizes accelerators with specialized processor-accelerator interfaces for data access acceleration. Cayman further introduces a novel accelerator merging mechanism to synthesize reusable accelerators. Experiments on various benchmarks demonstrate that Cayman outperforms two state-of-the-art frameworks by 8.0× and 14.4×.**

I. Introduction

Integrating accelerators has emerged as a critical strategy for augmenting the performance of modern applications. Specialized tasks are offloaded from the CPU core to accelerators for faster execution. Customizing accelerators encompasses two principal processes: *candidate selection* decides on the offloaded programs regions for acceleration, and *hardware design* implements the hardware accelerator. Manual accelerator design [8] is inefficient owing to the inherent complexity of applications and the intricacy of hardware design. Candidate selection frameworks [29], [30] only explore candidate regions based on accelerator models without synthesizing hardware. High-level synthesis (HLS) tools [11], [15], [25], [26] can synthesize accelerator hardware design from software descriptions. Accelerators' performance, power, and area can be further optimized via design space exploration [20], [27], [28]. However, HLS frameworks rely on designers to manually select the candidate for acceleration, which is time-consuming and often leads to sub-optimal solutions for a complex application.

Prior works attempted to develop an end-to-end flow to synthesize custom accelerators. However, they have severe limitations: they either exclude control flow and data access from accelerated kernels or only synthesize sequential and unoptimized implementation, inhibiting performance improvement. Specifically, custom functional unit synthesis frameworks [2], [5], [6], [10], [16], [19], [21] only accelerate dataflow graphs (DFGs) from target application programs. In other words, they do not accelerate code regions including any control flow or data access to memory. Off-core accelerator synthesis frameworks [4], [22], [23], [31] synthesize accelerators with control flow and memory access support. However, they solely synthesize sequential control flow implementation and adopt slow processor-accelerator interfaces for data access. As a result, they fail to achieve satisfying performance speedup.

This paper proposes Cayman, the first end-to-end framework that synthesizes high-performance custom accelerators with full control flow and data access support. Cayman automates both candidate selection and hardware synthesis. Specifically, Cayman selects kernels to synthesize based on the whole-application program structure tree (wPST) representation, which captures program regions of rich control flows for hardware acceleration. Cayman introduces a dynamic programming-based algorithm with heuristic pruning and solution filtering for efficient and comprehensive selection exploration. To synthesize high-performance accelerators for selected kernels, Cayman proposes an accelerator model that considers specialized processor-accelerator data access interfaces: *coupled*, *decoupled*, and *scratchpad*. The model efficiently generates accelerator configurations comprising control flow optimization and data access interface application and conducts performance and area estimation. Moreover, Cayman proposes a novel accelerator merging mechanism that shares reconfigurable datapath units within a reusable accelerator to accelerate multiple program regions even with diverse control flows, effectively reducing area overhead.

In summary, this paper makes the following contributions:

- We introduce Cayman, an end-to-end framework to synthesize high-performance custom accelerator automatically.
- We propose a dynamic programming-based candidate selection algorithm with heuristic pruning, modeling processor-accelerator data access interfaces and control flows.
- We propose a novel accelerator merging mechanism to synthesize reusable accelerators.

For evaluation, we compare Cayman with two state-of-the-art custom accelerator synthesis frameworks on various applications. Experimental results show that Cayman outperforms NOVIA [21] and QsCores [23] by 14.4× and 8.0× with the same total area budget for the synthesized accelerators.

II. Background and Related Work

A. High-level Synthesis

High-level synthesis (HLS) [9] can synthesize hardware accelerators from software kernels. With a specified target clock period, HLS tools [15], [25], [26] schedule the operations to

* Corresponding author

TABLE I: Comparison between prior works and Cayman.

Methods	design entry	candidate selection	control flow	data access	hardware sharing
HLS[1]	kernel	manual	optimized	specified	/
CFU[2]	application	auto	/	scalar-only	restricted
OCA[3]	application	auto	sequential	slow	restricted
Cayman	**application**	**auto**	**optimized**	**specialized**	**flexible**

[1] [3], [11]–[13], [15], [17], [18], [24]–[26]; [2] [2], [5], [6], [10], [19], [21]; [3] [4], [22], [23], [31]

cycles within the kernel, subsequently crafting the accelerator design based on the scheduling outcome. HLS tools support various optimization techniques, such as loop pipelining and unrolling, which may transform the kernel's source code or direct the scheduler to refine the control logic of the accelerator, thereby enhancing performance. DSE frameworks [20], [27], [28] are proposed to explore the optimizations automatically.

Limitations: As shown in Table I, HLS tools' design entry is software kernels, requiring designers to extract and specify manually, rather than selecting them from applications automatically. As a result, HLS requires significant human interaction. The same problem applies to ASSIST [17], Longnail [18], and Bluespec Accelerate-HLS [3], all of which adopt HLS but require manual candidate selection. Besides, designers must choose data access interfaces for external memory operations in kernels, which impact scheduling and overall performance but are overlooked by existing DSE frameworks.

B. Prior Works on Custom Accelerator Synthesis

Based on the types of synthesized accelerators, we segregate prior custom acceleration synthesis works into two classes: custom functional unit (CFU) synthesis frameworks [2], [5], [6], [10], [19], [21] and off-core accelerator (OCA) synthesis frameworks [4], [22], [23], [31].

Limitations: As shown in Table I, CFU synthesis frameworks select candidates exclusively in the DFG form, precluding control flows. Their data access interfaces only take scalar operands and return results since they do not support other memory accesses. OCA synthesis frameworks support control flow and memory access. However, they only implement sequential control logic and show data access interfaces, such as the scan-chain interface characterized by high latency and low bandwidth in [22], [23], significantly circumscribing performance gains. For hardware sharing, some CFU synthesis frameworks [5], [21] merge candidates into a reusable CFU, while some OCA frameworks [4], [23] endorse sharing solely among almost identical loops or functions. In other words, they support restricted hardware sharing and cannot reuse one accelerator to accelerate multiple kernels with diverse control flows, missing area-saving opportunities.

III. Methodology

A. Overview

As overviewed in Fig. 1, Cayman ingests target application programs as input and automatically identifies the high-performance solutions, each comprising multiple accelerators. The framework compiles programs into LLVM IR and builds

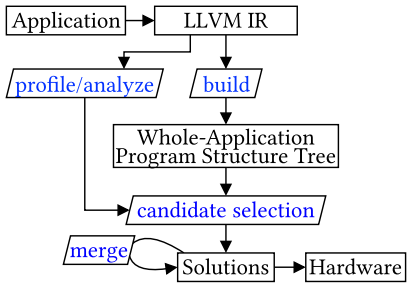
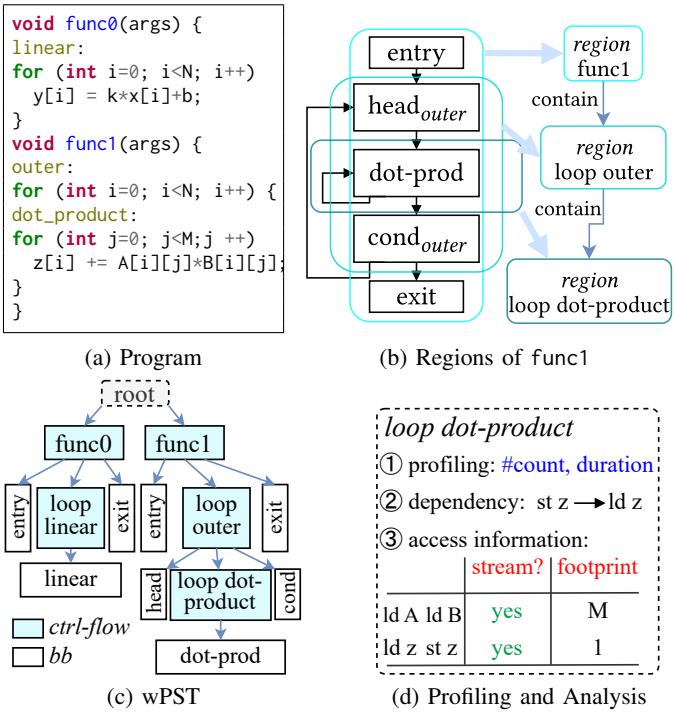


Fig. 1: Overview of the Cayman framework



(a) Program (b) Regions of func1

(c) wPST (d) Profiling and Analysis

Fig. 2: Example

the whole-application program structure tree (wPST) representation. The framework also runs profiling and program analysis through LLVM passes. The framework then leverages the profiling and analysis outcomes to conduct candidate selection on the wPST and produces solutions comprising selected kernels and accelerator configurations.

B. Representation, Profiling, and Analysis

Cayman introduces the whole-application program structure tree (wPST) representation for candidate selection. Traditional program structure tree (PST) [14] identifies single-entry-single-exit (SESE) *regions* inside a function as vertices and represents the containing relationships between regions as edges. The wPST extends PST with a *root* vertex representing the entire application and multiple function vertices representing every included function. We consider only SESE regions as legal acceleration candidates because the SESE property is necessary to guarantee that the candidate's execution can be completely isolated from the main processor through offloading and synchronization at their entry and exit points, respectively. The wPST captures all feasible acceleration candidates as its region vertices in a target application. Specifically, the

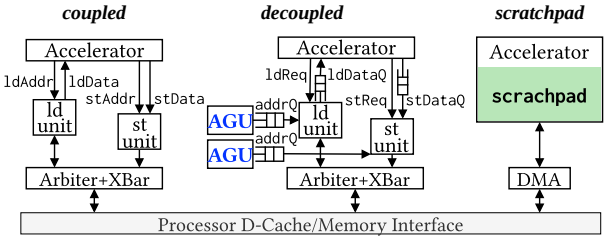


Fig. 3: CPU-accelerator data communication interfaces

wPST represents two region types: *bb* regions, corresponding to basic blocks, and *ctrl-flow* regions, corresponding to general control flow including loops and conditionals.

Cayman profiles application execution information including duration and execution count for every program region. The profiling results are essential for locating hotspot kernels and estimating performance gains. Besides, Cayman conducts program analysis to collect data access information, including memory dependencies, memory access patterns, and access footprints. Specifically, Cayman identifies loop-carried dependencies for every loop region, identifies *stream* access patterns whose access address sequence can be statically computed, and analyzes the access address range for every memory operation. The data access information is crucial for accelerator configuration decisions, including control flow optimizations and data access interface selections.

The program in Fig. 2a contains two functions, and Fig. 2c presents the corresponding wPST, showcasing *ctrl-flow* and *bb* regions as candidates for acceleration. Specifically, Fig. 2b shows the control-flow graph (CFG) of `func1` and illustrates three regions comprising control flows. Fig. 2d presents profiling ① and analysis ②③ results for the "loop dot-product" region. It includes one loop-carried dependency between the store and load operations to/from `z[i]`. Besides, all access operations have the *stream* access pattern, while `ld A` and `ld B` have the access footprint of `M` according to the induction variable `j` and `ld z` and `st z` have the access footprint of `1` due to the invariant `i` in the loop.

C. Accelerator Modeling

Data access interfaces between the processor and accelerators have nonnegligible impacts on the performance and area usage of the accelerators. However, it is overlooked by prior exploration works. Cayman systematically models three types of data access interfaces: *coupled*, *decoupled*, and *scratchpad*, as shown in Fig. 3:

- *Coupled* interface uses load/store units for access. A load-/store unit initiates data access to the memory system upon receiving requests with a target address from the accelerator, causing the accelerator to stall until the response arrives.
- *Decoupled* interface allocates dedicated address generation units (AGUs) for load/store units. The AGUs compute access addresses independently from the accelerator, which enables load units to start data access earlier than the accelerator's requests and enables store units to send out data later, both reducing the accelerator stall cycles. However, the *decoupled* interfaces pay extra area overhead for AGUs and data buffering FIFOs, and are only applicable to *stream*-pattern accesses for the requirements of independent address generation.
- *Scratchpad* interface reserves a dedicated buffer within the accelerator to cache a memory range. Data synchronization between the scratchpad and the memory system occurs before and after accelerator execution through a direct memory access (DMA) engine. It is beneficial when most positions in the cached memory range are accessed multiple times. However, the *scratchpad* interface incurs prominent area overhead for the scratchpad buffer and the DMA engine and requires statically analyzed footprints to determine the scratchpad size.

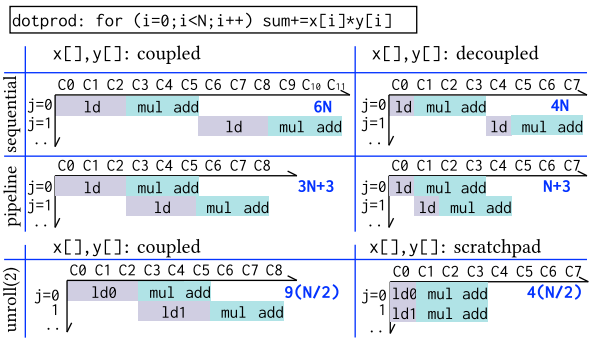


Fig. 4: Impacts of data communication interfaces.

Fig. 4 exemplifies data access interfaces' impacts on accelerator performance considering different control flow implementations. For sequential loop, the *coupled* interface leads to latency of 6N cycles, where N is the tripcount of the loop. The *decoupled* interface optimizes the latency to 4N cycles since it reduces the stall cycles associated with the loads by starting loads sooner before the accelerator requests. For loop pipelining, the *coupled* interface gives a pipeline initial interval (II) of 3 cycles, while the *decoupled* interface achieves an II of 1 cycle. For loop unrolling with the factor 2, the *coupled* interface leads to a latency of 9(N/2) cycles, and the *scratchpad* interface optimizes the latency to 4(N/2) cycles by enabling parallel access to the scratchpad without stalls. Cayman considers interface specialization for different access operations to achieve full data access acceleration.

Accelerator Configuration: Given the selected kernels for acceleration, Cayman can synthesize multiple accelerator configurations comprising control flow optimizations like loop unrolling and pipelining and data access interface strategies to achieve performance-area tradeoffs. Given the huge design space, Cayman adopts a fast exploration strategy. Specifically, it tries unrolling loops without loop-carried dependencies and pipelining the innermost loops after unrolling. Subsequently, Cayman determines data access interfaces heuristically for memory access operations: use *scratchpad* interface for operations with the total access count β times larger than the access footprint, indicating the necessity for caching; use *decoupled* interface for operations inside pipelined loops to achieve ideal II; and otherwise, use the *coupled* interface for area saving. Memory partitioning is configured for *scratchpad* interfaces inside unrolled loops to support parallel access. Consequently,

Algorithm 1: Candidate Selection

```
   Data: wPST T, profiling/analysis results R
   Result: Pareto-optimal selections
1  Function DP(vertex v):
2      if prune(v, R) then return ;
3      if v is bb then
4          F[v] ← filter(pareto(accel(v,R)));
5      else
6          F[v] ← ∅;
7          for u ∈ v.children do
8              DP(u);
9              F[v] ← filter(F[v] ⊗ F[u]);
10         if v is ctrl-flow then
11             F[v] ← filter(F[v] ∪ pareto(accel(v,R)));
12 DP(T.root);
13 return F[T.root];
```

Cayman efficiently generates candidate accelerator configurations with optimized control flow and data access.

Performance and Area Estimation: Cayman's accelerator model estimates performance gains and area usage for selected kernels with accelerator configurations without synthesizing complete hardware. The estimation comprises three steps: (1) apply loop unrolling to transform the kernels according to the configuration; (2) only synthesize pipelined loop regions P and sequential basic block regions B; and (3) estimate accelerators' latency and area through a bottom-up approach. Specifically, Cayman considers diverse interface-specific latencies and resource constraints when scheduling data access operations during synthesis. For every region in $P \cup B$, the cycle count is computed based on the scheduled latency and profiled execution count, and the area is retrieved from the synthesis report. For an outer region, the cycle count is calculated by summing the cycle counts of its child regions, and the area is estimated as the sum of its child regions' area plus extra control logic overhead. By doing so, the total cycle count of the accelerators is efficiently estimated as $Cycle_{cand}$. Given the profiled total duration of the accelerated kernels T_{cand} and the original execution duration of the entire program T_{all}, the overall performance gain is estimated as

$$Speedup = T_{all}/(T_{all} - T_{cand} + Cycle_{cand}/F) \quad (1)$$

where F is the target frequency of the synthesized accelerators.

D. Candidate Selection

Candidate selection determines a set of kernels corresponding to non-overlapping wPST subtrees for hardware acceleration to achieve the most performance gains under specific area budgets. Cayman models candidate selection as a knapsack problem: every wPST region, either *ctrl-flow* or *bb*, corresponds to a knapsack item, and the performance gain and area usage of the synthesized accelerator are the item's profit and weight, respectively. The knapsack problem has the constraint: if a vertex is selected, all its descendants cannot be selected, avoiding overlapping acceleration of kernels.

Cayman proposes a dynamic programming approach to solve the knapsack problem, as shown in Algorithm 1. The function DP(v) (line 1) solves the knapsack problem considering regions contained in the wPST subtree rooted at the vertex v. The function prune (line 2) terminates the search if the region v is identified as not a hotspot worth acceleration according to the profiling and analysis results R and a heuristic pruning strategy. We define $F[v]$ as the area-performance Pareto-optimal solution sequence accelerating kernels from v's subtree. Every solution $\phi \in F[v]$ comprises one or more non-overlapping kernels with accelerator configurations. Algorithm 1 builds $F[v]$ (line 3-11) according to the following recursion formula:

$$F[v] \leftarrow \begin{cases} \texttt{pareto(accel}(v,R)\texttt{)} & \text{if } v \text{ is } bb \\ \texttt{pareto(accel}(v,R)\texttt{)} \cup \bigotimes_{u \in v.children} F[u]) & \text{if } v \text{ is } ctrl\text{-}flow \\ \texttt{pareto}(\bigotimes_{u \in v.children} F[u]) & \text{otherwise} \end{cases}$$

where accel calls Cayman's accelerator model to generate configurations for selected kernels, producing accelerators with optimized control flow and data access and estimating performance and area as discussed in Section III-C, and pareto produces a Pareto-optimal sequence for the given solutions. Specifically, there are three cases for $F[v]$'s construction according to the type of vertex v. If v is a *bb* region, $F[v]$ gets accel(v, R) solutions accelerating v's dataflow subgraphs. If v is a *ctrl-flow* region, either the region v gets accelerated as an extracted kernel by a single accelerator, corresponding to accel(v, R) solutions, or its descendant regions get accelerated separately, applying the $\otimes$ operation to combine solutions from sibling subtrees rooted at v's children: $F[u_1] \otimes F[u_2]$=pareto$(\{\phi_1 \cup \phi_2 \mid \phi_1 \in F[u_1], \phi_2 \in F[u_2]\})$. Otherwise, v is the *root* vertex that cannot be selected directly, and $F[T.root]$ applies the $\otimes$ operation to combine solutions from different functions to form the overall selection solutions.

Algorithm 1 also features the filter function (line 4,9,11) that filters out unnecessary solutions that are too close to each other. Specifically, for a Pareto-optimal solution sequence $\{\ldots, \phi_i, \phi_{i+1}, \ldots, \phi_j, \ldots\}$ with increasing area $a_{i+1} > a_i$, suppose ϕ_j is the first solution whose area a_j is α times larger than that of ϕ_i. That is, $a_j > \alpha a_i$, filter removes solutions $\{\phi_{i+1}, \ldots, \phi_{j-1}\}$ from the sequence. After filtering, every neighboring solution pair (ϕ_i, ϕ_{i+1}) satisfies that $a_{i+1} > \alpha a_i$. For any knapsack with an area limit A, filter reduces the maximum possible number of solutions from A to $log_\alpha A$, greatly improving selection efficiency.

Algorithm Complexity: The complexity of Algorithm 1 is $O(N \log_\alpha^2 A + E)$, where N denotes the number of unpruned vertices in the wPST, A denotes the discretized area limit, and E denotes the total time complexity of the accelerator model. The term $N \log_\alpha^2 A$ arises from $F[v]$'s construction, where the $\otimes$ operation enumerates solution combinations from two F sequences, each of which contains at most $\log_\alpha A$ solutions.

E. Accelerator Merging

After the candidate selection, Cayman performs accelerator merging, enabling multiple program regions with distinct control flows to share a reusable accelerator. The core strategy is to merge basic blocks that share common operations by inserting multiplexers with reconfiguration bit registers, producing a reconfigurable datapath unit. Cayman then combines candidates that include the merged basic blocks into a reusable accelerator, each maintaining a standalone finite-state machine (FSM)

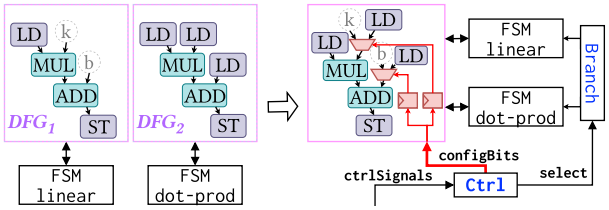


Fig. 5: Accelerator merging

for control. Upon triggering the reusable accelerator, a global control unit, named *Ctrl*, transmits configuration data to the datapath units and selects the appropriate control logic FSM to trigger. This merging strategy is based on the observation that basic blocks housing operations like floating-point arithmetic and data access contribute significantly to the accelerator area, while control logic FSMs demand much less area. Fig. 5 presents the example of merging the loop linear and loop dot-product candidates from Fig. 2. Cayman synthesizes a reusable accelerator containing a reconfigurable hardware unit merged from DFG_1 and DFG_2. The reusable accelerator accelerates both the loops with individual FSMs.

Cayman conducts accelerator merging in a heuristic manner for every Pareto-optimal selection solution that comprises multiple accelerators. It estimates the area savings for merging every pair of basic blocks contained in the solution, merges the basic block pair with the maximum estimated area savings into a reconfigurable datapath, and combines the two accelerators containing the merged basic blocks into a reusable accelerator. The merged datapaths and accelerators are also treated as normal basic blocks and accelerators, enabling further merging with others. The heuristic merging process is repeated until no area savings can be achieved.

F. Implementation

We implement Cayman based on LLVM 18 for candidate selection, accelerator merging, and hardware synthesis. We utilize the *RegionInfoAnalysis* pass to identify regions and build wPSTs. For profiling, we implement an instrumentation pass to insert execution count and timestamp recording instructions within regions; the modified bitcode is compiled and executed to gather profiling data. For program analysis, we employ the *MemoryDependenceAnalysis* pass to get memory dependencies, implement a custom pass to identify *stream* patterns, and run the *ScalarEvolution* pass to analyze footprints. Cayman's accelerator model retrieves delay and area of operations and data access interfaces by synthesizing them with OpenROAD [1] targeting Nangate45 PDK.

IV. Evaluation

A. Experiment Setup

All applications are compiled with the "-O3" flag. The target frequency of accelerators is set to 500 MHz. Speedup is calculated as per Equation 1. Area is retrieved from reports of synthesizing accelerators into complete hardware and is presented as ratios to that of a CVA6 RISC-V tile [32]. We compare Cayman to two state-of-the-art automated accelerator

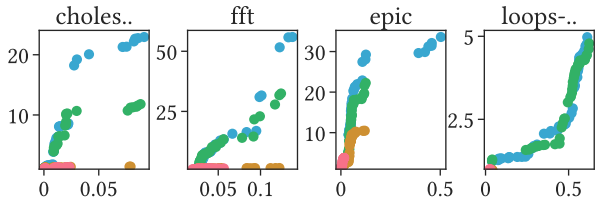


Fig. 6: Speedup (y-axis) and area (x-axis) of solutions: ● NOVIA, ● QsCores, ● *coupled*-only Cayman, and ● full Cayman.

synthesis frameworks, NOVIA [21] and QsCores [23]. We use diverse benchmarks from benchmark suites including PolyBench, MachSuite, MediaBench, and CoreMark-Pro for a comprehensive evaluation.

B. Result Analysis

We evaluate two different area budgets for the synthesized accelerators: small (25%) and large (65%). The area budgets are within the typical area range of custom accelerators [**?**], [7]. Table II presents experimental results. Cayman outperforms NOVIA and QsCores for all benchmarks. It achieves an average speedup of 14.4× and 8.0× over NOVIA and QsCores, respectively, under the budget 25%. The speedup increases to 27.2× and 15.0× when the budget is 65%, demonstrating the better acceleration potentials of Cayman. Cayman achieves superior performance because it optimizes control flow and adopts fast data access interfaces like *decoupled* and *scratchpad*. On the contrary, NOVIA fails to support control flow and memory accesses, and QsCores synthesizes sequential control flow and slow access interfaces, achieving much less performance improvement than Cayman.

Table II also presents kernel selection and accelerator configuration details of Cayman solutions. For applications with even-distributed hotspots, Cayman selects more kernels for acceleration when a larger area budget is available. One notable example is the loops-all-mid-10k-sp workload, where Cayman accelerates 13 sequential basic blocks and 1 pipelined region for the budget 25%, and accelerates 122 sequential basic blocks and 40 pipelined regions for the budget 65%. For applications with centralized hotspots, Cayman accelerates the performance-critical regions intently with more area. For example, Cayman synthesizes accelerators with the same #SB and #PR for two area budgets for 13 of 16 benchmarks from PolyBench in Table II, indicating consistent regions for acceleration. Cayman accelerates more access operations with a larger area budget: on average, it adopts 40 interfaces for the budget 25%, which is increased to 53 for the budget 65%. Besides, *decoupled* and *scratchpad* interfaces are widely adopted, occupying 83% and 81% on average for two budgets, respectively, demonstrating the significance of Cayman's interface specialization. Besides, more *scratchpad* interfaces are used when more area is available since the expensive *scratchpad* interface enables greater access parallelism.

Table II also shows that Cayman's accelerator merging mechanism effectively saves 36% and 35% area for two area budgets. Cayman widely synthesizes reusable accelerators, each of which accelerates 3 distinct program regions on average from the benchmarks. The area saving percentage goes up to

TABLE II: Results under two area budgets 25% and 65%. #SB and #PR denote the number of sequential basic blocks and pipelined regions from selected kernels. #C, #D, and #S denote the number of *coupled*, *decoupled*, and *scratchpad* interfaces in synthesized accelerators. The "Area saving" columns denote the area-saving percentage by accelerator merging.

Suite	Benchmark	Small Area Budget (25%)								Large Area Budget (65%)								Runtime (s)
		Speedup(×)		Configurations					Area saving(%)	Speedup(×)		Configurations					Area saving(%)	
		over [21]	over [23]	#SB	#PR	#C	#D	#S		over [21]	over [23]	#SB	#PR	#C	#D	#S		
PolyB	3mm	35.4	14.0	16	9	0	0	15	74	117.9	46.8	16	9	0	0	15	70	69.2
	atax	16.5	6.5	9	9	0	9	1	38	32.2	12.7	9	9	0	0	10	21	15.5
	bicg	22.2	8.7	8	6	0	9	2	38	42.5	16.8	8	6	0	0	11	42	17.5
	doitgen	72.6	29.0	9	6	0	0	7	5	72.6	29.0	9	6	0	0	7	5	37.3
	mvt	13.2	5.2	8	9	1	8	0	26	19.5	7.7	8	9	0	0	9	21	18.9
	symm	13.6	9.2	4	3	3	6	1	38	37.8	25.7	4	3	0	0	10	35	1.4
	syrk	19.4	7.8	9	6	0	6	0	24	79.9	31.9	4	3	0	0	4	0	3.2
	trmm	19.0	7.6	4	3	2	4	0	38	19.0	7.6	4	3	2	4	0	38	0.0
	cholesky	17.0	15.7	12	6	3	8	0	26	17.0	15.7	12	6	3	8	0	26	0.1
	gramschmid..	20.3	8.3	11	12	1	14	0	24	91.6	37.7	5	6	0	0	9	48	47.2
	lu	17.5	16.3	16	9	3	12	0	26	17.5	16.3	16	9	3	12	0	26	0.2
	trisolv	15.3	6.0	8	6	1	5	6	38	15.3	6.0	8	6	1	5	6	38	0.1
	covariance..	18.2	7.3	4	3	5	4	0	38	18.2	7.3	4	3	5	4	0	38	0.1
	jacobi-2d	15.0	13.1	13	6	0	12	0	24	102.9	90.3	13	6	0	0	12	50	47.0
	deriche	5.6	4.5	26	18	0	20	0	34	13.5	10.1	28	21	0	21	0	39	0.1
	floyd-wars..	7.5	6.8	8	6	0	7	0	24	7.5	6.8	8	6	0	7	0	24	0.0
MachS	fft	42.6	38.5	5	14	4	22	2	26	42.6	38.5	5	14	4	22	2	26	0.2
	md	7.6	4.7	18	23	12	11	2	44	7.6	4.6	18	23	12	11	2	44	0.1
	spmv	8.6	8.6	11	11	4	5	0	44	9.5	9.5	9	14	5	5	0	26	0.2
	nw	10.8	4.7	5	24	0	9	1	38	10.8	4.7	5	24	0	9	1	38	0.1
Media	cjpeg	2.8	1.9	47	39	30	25	47	47	3.9	2.4	73	51	5	38	119	39	0.6
	epic	7.2	2.6	106	62	2	10	25	41	8.5	3.1	100	60	0	0	33	41	1068.1
CoreM	cjpeg-rose..	5.0	4.2	99	38	75	150	0	47	5.8	4.8	106	31	107	13	140	42	172.9
	zip-test	6.4	6.4	85	38	14	279	0	48	7.5	7.5	87	39	14	295	0	35	21.4
	parser-125..	5.0	4.3	3	4	0	0	5	43	5.0	4.3	3	4	0	0	5	43	100.4
	nnet-test	4.5	4.6	5	19	6	89	0	21	8.2	8.4	36	22	6	120	76	46	272.6
	linear-alg..	12.5	12.1	63	12	24	23	34	49	22.6	21.8	72	12	24	28	34	44	78.2
	loops-all-..	1.4	1.4	13	1	0	0	8	48	5.0	4.9	122	40	86	110	0	48	10.5
	average	14.4	8.0	22	14	7	27	6	36	27.2	15.0	28	16	10	25	18	35	70.8

74% and 70% for the 3mm benchmark, which includes 3 loops with identical basic blocks for merging. However, Cayman only saves 5% area for the doitgen benchmark since the benchmark only includes one hotspot region for acceleration and does not need accelerator merging. In practice, Cayman runs very fast, which only takes 70.8s on average.

Fig. 6 presents the speedup and area usage of Pareto-optimal solutions produced by the baselines and Cayman for four benchmarks from different suites. It shows that Cayman solutions achieve superior performance with different area usage for all the benchmarks. Specifically, NOVIA solutions are always located in the lower-left corner, indicating their limited speedup and low area overhead. Cayman also achieves comparable or better performance than NOVIA for low area budgets since Cayman explores a broader accelerator synthesis design space that fully covers NOVIA's. QsCores's performance speedup scales worse than Cayman when the area budget increases. Especially, QsCores cannot accelerate the loops-all-mid-10k-sp workload from CoreMark-Pro because it contains intensive control flow and memory access which cannot be accelerated by QsCores' sequential control and slow data access interface. Compared to full Cayman solutions, *coupled*-only ones achieve lower speedup for most benchmarks, demonstrating the significance of modeling and specializing processor-accelerator interfaces. The only exception is the loops-all-mid-10k-sp workload, where the difference between *coupled*-only and full Cayman solutions are minor. The reason is that loops in the workload commonly have loop-carried dependencies between floating-point operations, restricting the achievable pipeline II and circumventing the performance gains of adopting *decoupled* and *scratchpad* interfaces.

V. Conclusion

This paper proposes Cayman, an end-to-end framework for high-performance custom accelerator generation. Cayman models processor-accelerator data access interfaces and conducts efficient dynamic programming-based candidate selection, synthesizing accelerators with optimized control flow and data access. Cayman also introduces a novel accelerator merging mechanism to synthesize reusable accelerators. Experimental results on various benchmarks demonstrate that Cayman significantly outperforms the state-of-the-art frameworks.

Acknowledgment

This work was supported in part by the National Science Foundation of China (Grant No. T2325001) and Shanghai Municipal Science and Technology Major Project.

REFERENCES

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[2] K. Atasu, L. Pozzi, and P. Ienne, "Automatic application-specific instruction-set extensions under microarchitectural constraints," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, 2003, pp. 256–261.

[3] Bluespec. (2024) Bluespec accelerate-hls. [Online]. Available: https://info.bluespec.com/acceleratehls

[4] I. Brumar, G. Zacharopoulos, Y. Yao, S. Rama, D. Brooks, and G.-Y. Wei, "Early dse and automatic generation of coarse-grained merged accelerators," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 2, Jan. 2023. [Online]. Available: https://doi.org/10.1145/3546070

[5] N. Clark, H. Zhong, and S. Mahlke, "Processor acceleration through automated instruction set customization," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. USA: IEEE Computer Society, 2003, p. 129.

[6] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific instruction generation for configurable processor architectures," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 183–189. [Online]. Available: https://doi.org/10.1145/968280.968307

[7] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "An analysis of accelerator coupling in heterogeneous architectures," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[8] M. Damian, J. Oppermann, C. Spang, and A. Koch, "Scaie-v: An open-source scalable interface for isa extensions for risc-v processors," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 169–174. [Online]. Available: https://doi.org/10.1145/3489517.3530432

[9] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin, *High—Level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media, 2012.

[10] V. Gnanasambandapillai, J. Peddersen, R. Ragel, and S. Parameswaran, "Finder: Find efficient parallel instructions for asips to improve performance of large applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[11] X. Hao, H. Rong, M. Zhang, C. Sun, H. Jiang, and Y. Liang, "Popa: Expressing high and portable performance across spatial and vector architectures for tensor computations," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 199–210. [Online]. Available: https://doi.org/10.1145/3626202.3637566

[12] L. Jia, Z. Luo, L. Lu, and Y. Liang, "Tensorlib: A spatial accelerator generation framework for tensor algebra," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE Press, 2021, p. 865–870. [Online]. Available: https://doi.org/10.1109/DAC18074.2021.9586329

[13] L. Jia, Y. Wang, J. Leng, and Y. Liang, "Ems: efficient memory subsystem synthesis for spatial accelerators," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 67–72. [Online]. Available: https://doi.org/10.1145/3489517.3530411

[14] R. Johnson, D. Pearson, and K. Pingali, "The program structure tree: Computing control regions in linear time," ser. PLDI '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 171–185. [Online]. Available: https://doi.org/10.1145/178243.178258

[15] L. Josipović, R. Ghosal, and P. Ienne, "Dynamically scheduled high-level synthesis," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 127–136. [Online]. Available: https://doi.org/10.1145/3174243.3174264

[16] C. Li, Y. Wang, H. Li, and Y. Han, "Append: Rethinking asip synthesis in the era of ai," ser. DAC '23, 07 2023, pp. 1–6.

[17] G. Liu, J. Primmer, and Z. Zhang, "Rapid generation of high-quality risc-v processors from functional instruction set specifications," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[18] J. Oppermann, B. M. Damian-Kosterhon, F. Meisel, T. Mürmann, E. Jentzsch, and A. Koch, "Longnail: High-level synthesis of portable custom instruction set extensions for risc-v processors from descriptions in the open-source coredsl language," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24, 2024. [Online]. Available: https://doi.org/10.1145/3620666.3651375

[19] L. Pozzi, K. Atasu, and P. Ienne, "Exact and approximate algorithms for the extension of embedded processor instruction sets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1209–1229, 2006.

[20] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "Autodse: Enabling software programmers design efficient fpga accelerators," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 147. [Online]. Available: https://doi.org/10.1145/3431920.3439464

[21] D. Trilla, J.-D. Wellman, A. Buyuktosunoglu, and P. Bose, "Novia: A framework for discovering non-conventional inline accelerators," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 507–521. [Online]. Available: https://doi.org/10.1145/3466752.3480094

[22] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: reducing the energy of mature computations," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1, pp. 205–218, 2010. [Online]. Available: https://dl.acm.org/doi/10.1145/1735970.1736044

[23] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, "Qscores: Trading dark silicon for scalable energy efficiency with quasi-specific cores," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011.

[24] Y. Xiao, Z. Luo, K. Zhou, and Y. Liang, "Cement: Streamlining fpga hardware design with cycle-deterministic ehdl and synthesis," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 211–222. [Online]. Available: https://doi.org/10.1145/3626202.3637561

[25] Xilinx Inc. (2024) Vitis high-level synthesis user guide (ug1399). [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls

[26] R. Xu, Y. Xiao, J. Luo, and Y. Liang, "Hector: A multi-level intermediate representation for hardware synthesis methodologies," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549370

[27] H. Ye, C. Hao, J. Cheng, H. Jeong, J. Huang, S. Neuendorffer, and D. Chen, "ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation," 2021. [Online]. Available: http://arxiv.org/abs/2107.11673

[28] C. H. Yu, P. Wei, M. Grossman, P. Zhang, V. Sarker, and J. Cong, "S2fa: an accelerator automation framework for heterogeneous computing in datacenters," ser. DAC '18, New York, NY, USA, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196109

[29] G. Zacharopoulos, A. Ejjeh, Y. Jing, E.-Y. Yang, T. Jia, I. Brumar, J. Intan, M. Huzaifa, S. Adve, V. Adve, G.-Y. Wei, and D. Brooks, "Trireme: Exploration of hierarchical multi-level parallelism for hardware acceleration," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 3, Apr. 2023. [Online]. Available: https://doi.org/10.1145/3580394

[30] G. Zacharopoulos, L. Ferretti, G. Ansaloni, G. Di Guglielmo, L. Carloni, and L. Pozzi, "Compiler-assisted selection of hardware acceleration candidates from application source code," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, 2019, pp. 129–137.

[31] G. Zacharopoulos, L. Ferretti, E. Giaquinta, G. Ansaloni, and L. Pozzi, "Regionseeker: Automatically identifying and selecting accelerators from application source code," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[32] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019.

Enhancing LLM-based Quantum Code Generation with Multi-Agent Optimization and Quantum Error Correction

Charlie Campbell
Department of Computing
Imperial College London
London, UK
charlie.campbell22@imperial.ac.uk

Hao (Mark) Chen
Department of Computing
Imperial College London
London, UK
hao.chen20@imperial.ac.uk

Wayne Luk
Department of Computing
Imperial College London
London, UK
w.luk@imperial.ac.uk

Hongxiang Fan
Department of Computing
Imperial College London
London, UK
hongxiang.fan@imperial.ac.uk

***Abstract*—Multi-agent frameworks with Large Language Models (LLMs) have become promising tools for generating general-purpose programming languages using test-driven development, allowing developers to create more accurate and robust code. However, their potential has not been fully unleashed for domain-specific programming languages, where specific domain exhibits unique optimization opportunities for customized improvement. In this paper, we take the first step in exploring multi-agent code generation for quantum programs. By identifying the unique optimizations in quantum designs such as quantum error correction, we introduce a novel multi-agent framework tailored to generating accurate, fault-tolerant quantum code. Each agent in the framework focuses on distinct optimizations, iteratively refining the code using a semantic analyzer with multi-pass inference, alongside an error correction code decoder. We also examine the effectiveness of traditional techniques, like Chain-of-Thought (CoT) and Retrieval-Augmented Generation (RAG) in the context of quantum programming, uncovering observations that are different from general-purpose code generation. To evaluate our approach, we develop a test suite to measure the impact each optimization has on the accuracy of the generated code. Our findings indicate that techniques such as structured CoT significantly improve the generation of quantum algorithms by up to 50%. In contrast, we have also found that certain techniques such as RAG show limited improvement, yielding an accuracy increase of only 4%. Moreover, we showcase examples of AI-assisted quantum error prediction and correction, demonstrating the effectiveness of our multi-agent framework in reducing the errors of generated quantum programs.**

***Index Terms*—Machine Learning, Quantum Code Generation, Quantum Computing, Multi-agent Large Language Models**

I. Introduction

Recent advances in Large Language Models (LLMs) have revolutionized a wide range of AI applications, including natural language understanding, machine translation, and automated content generation [1]. Building on the exceptional natural language processing capabilities of LLMs, recent research has extended into the exploration of multi-agent frameworks, where each agent is driven by an LLM to facilitate complex interactions in complicated environments. These multi-agent frameworks form the foundation for embodied AI systems [2], paving the way for a future where AI plays a critical role in assisting and supporting daily human activities.

A key application of LLMs is in automatic software development, where AI can assist developers in generating increasingly high-quality code [3]. Various LLMs tailored for code generation, such as StarCoder [4] or CodeLlama [5], have been introduced to produce high-quality code and test suites. Furthermore, recent research has explored multi-agent frameworks, such as AgentCoder [6], to enhance collaborative code generation. However, most of these efforts have focused on general-purpose programming languages like C++ and Python, with relatively less emphasis on domain-specific programming languages.

As a pioneering force in quantum computing, IBM has advanced the application of LLMs for quantum program generation, demonstrating the potential of LLMs in producing domain-specific programs [7]. To facilitate the development of LLM in quantum computing, they introduce training datasets, a customized LLM and testbench for quantum program generation. While this work lays a solid foundation for leveraging LLMs in quantum computing, several key challenges remain in optimizing LLM-assisted quantum code generation:

- The rapid pace of quantum computing advancements requires frequent updates to code libraries, resulting in a lack of high-quality, up-to-date training data. Models trained on data that are only a few months old can quickly become obsolete due to the release of new libraries or algorithms.
- Quantum programs generated by LLMs, with only standard supervised fine-tuning, tend to be error-prone. It is essential to explore the role of advanced prompt engineering in improving the reliability of LLM-assisted quantum code generation.
- Prior approaches have applied traditional code generation techniques used for general-purpose programming languages to quantum computing without accounting for domain-specific optimizations, such as Quantum Error Correction (QEC). This results in suboptimal performance. For instance, to meet the stringent standards of quantum code generation, LLMs must be capable of producing fault-tolerant quantum code using QEC techniques.

To address the aforementioned challenges, this paper proposes a novel multi-agent quantum code generation framework that integrates iterative multi-pass optimization and automatic quantum error correction. Inspired by OpenAI's *GPT-o1* [1] which emphasizes inference time optimization over training

[1]https://openai.com/index/introducing-openai-o1-preview/

time effort, we demonstrate that multiple inference-time optimization techniques can effectively mitigate the issue introduced by limited training data. To enhance the efficiency of quantum code generation, we explore the impact of advanced prompt engineering techniques such as Chain of Thought (CoT) [8] and Retrieval-Augmented Generation (RAG) [9], providing different insights for optimizing LLM-assisted quantum code generation. Furthermore, by leveraging the domain-specific optimization potential in quantum programming, our framework incorporates a quantum error predictor to facilitate automatic quantum error correction, ensuring robust and fault-tolerant quantum code generation. Overall, this work makes the following contributions:

- We propose a novel multi-agent framework comprising three key agents: a code generation agent, a semantic analyzer, and a quantum error predictor. The framework employs a multi-pass inference strategy for iterative optimization (Section III).
- We investigate the effectiveness of different advanced prompt engineering techniques, presenting detailed experimental results that provide insights into optimizing LLM-assisted quantum code generation (Section IV).
- Our framework integrates automatic quantum error correction by leveraging Quantum Error Correction (QEC) decoders, significantly reducing noise in quantum environments (Section V-B).

II. Background and Related Work

A. LLMs and Multi-Agent Frameworks

Large Language Models (LLMs) are built upon the transformer model [10]. The most powerful and recent models, such as GPT-4 [11], are built upon the decoder-only transformer architecture, in which the output sequence is based on the previous tokens, rather than having separate encoders and decoders. Such models are used for a variety of Natural Language Processing (NLP) tasks, proving to be versatile tools that have the potential to have wide-ranging effects.

One developing area of LLM research is multi-agent frameworks. In this, multiple LLM agents are combined together to allow interaction between different LLMs, which could result in a more accurate response or allow users to use multiple input formats. [6] [12]. This has many potential uses, such as safeguarding text generation or a hierarchical chat.

B. Chain-of-Thought and Retrieval-Augmented Generation

Inference-time optimisation techniques have been shown to improve the quality of output generated by LLMs [13]. These techniques often either augment the prompt with additional information or get the model to think in different ways to better their understanding. Two promising methods are Retrieval-Augmented Generation (RAG) [9] and Chain-of-Thought prompting [8].

RAG involves collecting data or documents relevant to the generation task and building a vectorstore database. We then retrieve the appropriate chunks of data based on a ranking algorithm and augment the prompt with this additional data [9]. This gives the model more specific context from which to infer the answer, improving generation accuracy. Chain-of-Thought (CoT) prompting [8] encourages models to use logical reasoning when generating answers. Zero-shot CoT [14], in which the model is explicitly told to think "step by step", encourages the usage of logical reasoning. Another technique is manual CoT [8] in which a sample question and answer pair is provided to demonstrate the type of reasoning the model should use. Overall, both RAG and CoT have been shown to improve the accuracy of models at the inference stage by augmenting the context or eliciting the logical reasoning ability of the LLM.

C. Quantum Computing

Quantum computing is a rapidly growing field that shows profound potential. It uses quantum phenomena such as entanglement and superposition to allow information to be stored in an exponentially more dense manner using qubits. Whilst this information cannot be accessed in a one-to-one manner, it does allow access to unique phenomena such as interference and true parallelism, creating new algorithmic possibilities. The ability to exploit these phenomena has allowed quantum algorithms to be made for many problems that are NP or NP-complete in the classical environment. One example is Shor's algorithm for finding prime factors [15], which is a problem that is NP for classical computers. This algorithm uses superposition and interference to manipulate qubits in such a way as to find the factors in polynomial time. There are many more examples of algorithms in cryptography [16] and beyond, making quantum computing a field that has exciting possibilities.

One unique challenge posed by developing quantum algorithms is the presence of quantum noise. This is caused by factors such as thermal fluctuations and can result in qubit measurements being incorrect, meaning that experimental results often differ from theoretical results. Much work has been done in the field of quantum error correction (QEC), with many QEC codes being developed, such as the Steane code [17]. One common subset of QEC codes are surface codes [18], which encode one logical qubit onto a lattice of physical qubits. The nature of these codes means that they are topology-dependent, making it difficult to apply QEC codes in a practical development environment.

D. Related Work

LLM development is undergoing rapid improvement in the field of code generation [3], [19]. Models have shown promising ability to capture both the syntax and semantics of a language, with popular models such as CodeLlama [5] being able to produce high quality code. These models achieve consistently high scores on coding benchmarks like HumanEval [20], which is developed to evaluate models' ability to produce code to the same standard as a human software engineer. Multi-agent frameworks have also been utilized to create a fully-fledged code development environment, with a framework that can produce code and generate an appropriate test suite [6].

Writing code for quantum computers poses significant challenges, as quantum algorithms must be reversible and error tolerant, which demands developers to approach problems in a

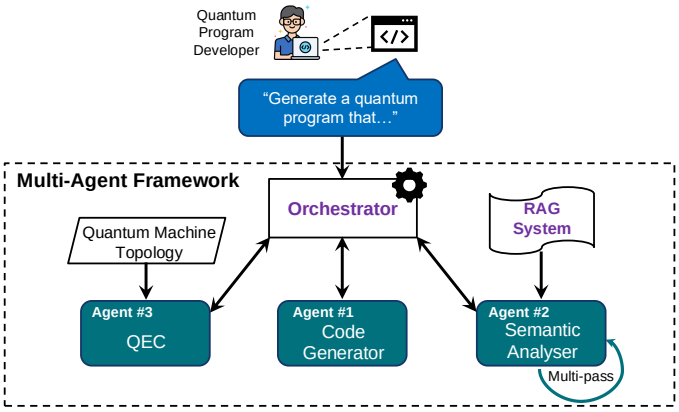


Fig. 1: Multi-Agent Framework for Quantum Code Generation.

different way. This creates a unique opportunity for LLMs to aid developers in code writing. Despite the accessibility of the Qiskit library [7], there is a steep learning curve for beginners to develop novel quantum algorithms. LLMs have the potential to enable developers to create accurate, fault-tolerant code without being experts in the field. IBM's Qiskit Code Assistant model [7] attempts to provide this by fine-tuning on a corpus of Qiskit programs. It achieves better accuracy on a HumanEval style benchmark [21] than traditional code generation models but still only achieves 46% accuracy, suggesting further development is needed. Furthermore, it does not take into account the presence of quantum noise in its code generation, so it will not produce fault-tolerant code.

III. Multi-Agent Framework and Training

A. Framework Overview

An overview of the proposed multi-agent framework is depicted in Figure 1. The framework mainly comprises three agents, each responsible for a specific task: code generation, semantic analysis, and quantum error correction.

- **Code Generation Agent**: We adopt Starcoder [4] as the base model for quantum code generation. To enhance the quality of the generated quantum program, we apply supervised fine-tuning on Starcoder using a custom dataset containing the latest open-source Qiskit [7] code. Additionally, we introduce several optimization techniques aimed to further improve the accuracy of the code generation agent, which will be detailed in Section IV.
- **Semantic Analysis Agent**: To enhance the quantum program generated by the initial model with better semantic accuracy, we design a dedicated agent for semantic analysis. This agent can help the code generation process to mitigate the lack of knowledge available in online sources about quantum algorithms, allowing users to incrementally improve the generated code to meet their requirements.
- **QEC Decoder Generation Agent**: Given the error-prone nature of quantum computing, primarily caused by quantum noise, it is essential to incorporate Quantum Error Correction (QEC) into our code generation process to enhance the reliability and robustness of the generated quantum programs. To achieve this, we adopt a QEC code decoder to predict and reduce quantum error, allowing our framework to generate fault-tolerant code.

By designing the multi-agent framework shown in Figure 1, we are able to address the three main challenges in optimizing quantum code generation. It allows us to not only generate syntactically and semantically valid quantum code, but also ensure that we can run the generated code in a stable environment, by the support of AI-generated Error Correction Codes (ECCs).

B. Training Dataset and Test Suite

We collected our training data by scraping Github repositories with an open-source license. Due to the fast-paced nature of quantum computing, we filtered the repositories to those updated after February 2024. We found that even filtering by a date this recent still resulted in out-of-date code, including code from official Qiskit Community repositories [22]. We further filtered the retrieved Python and Jupyter Notebook files based on whether they contained a Qiskit import statement. The notebooks were then split into code and markdown tiles based on sentinel tokens [7]. After filtering, our total number of tokens was 3M. Whilst this is a small dataset, it did not affect our approach to obtaining meaningful results, as we seek to compare the effectiveness of both fine-tuning techniques and a multi-agent structure. By demonstrating that these optimization techniques and our framework provide a meaningful improvement on the generated quantum code, we can justify working on creating a more accurate code generation agent by creating a larger dataset with higher data quality.

In order to perform our experiments, we created a set of prompt-answer pairs to be passed into our multi-agent system. These prompts cover a wide range of code-generation topics.

- **Basic**: These tests covers basic code generation for the Qiskit library, allowing us to test the basic syntactic requirements of our model. The tests include basic circuit generation and simulation on quantum devices, ensuring the model can generate and run code on real-world devices.
- **Intermediate**: These cover more complicated quantum circuits and algorithms, with a focus on well-known algorithms, such as Shor's [15] and Grover's [23] algorithms. This is intended to test advanced syntactic code generation, as well as allow for semantic testing on common applications.
- **Advanced**: This part covers more advanced topics specific to quantum computing. For example, this section contains prompts covering quantum teleportation [24], the quantum walk algorithm [25] and quantum annealing [26]. We expect the model to have little to no knowledge of these algorithms from the base training, so this section can be used to carry out advanced semantic testing.

Our test suite consists of 47% basic tests, 24% intermediate and 29% advanced. Compared to the Qiskit HumanEval benchmark [21], our test suite contains much more challenging prompts, asking the model to reason about complex quantum algorithms that the base model would have no knowledge of.

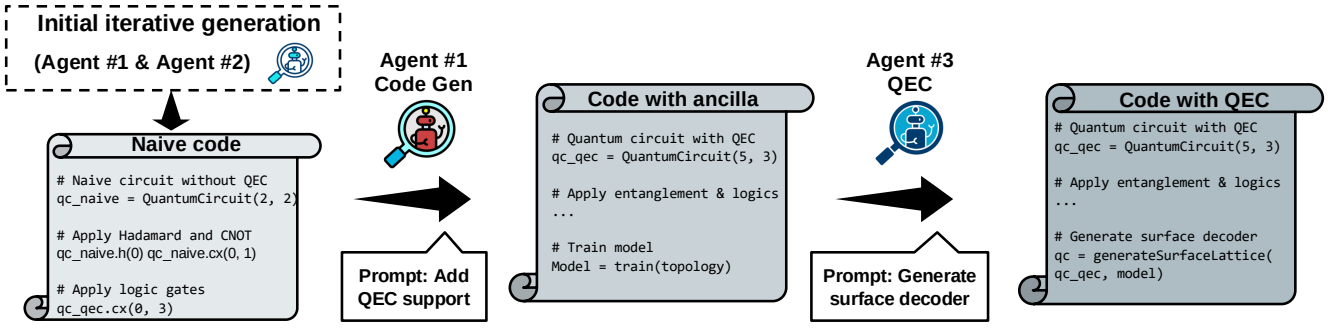


Fig. 2: Evolution of code during QEC generation.

IV. Optimization

A. Iterative Multi-Pass Optimization

Due to the limited availability of high-quality training data for quantum programming, the quantum programs generated by the fine-tuned LLM model remain error-prone. Inspired by recent advances in OpenAI's GPT-o1, we adopt an inference-time optimization approach to generate high-quality quantum programs. To this end, we implement an iterative multi-pass quantum code generation strategy. Specifically, we created a prompt template for multi-pass inference that includes the original prompt, as well as the generated code and error trace. This new prompt is then passed into the model in the hope of fixing the error generated on the previous pass.

By using a multi-pass structure and iterating upon the previous code, we are able to solve errors as they occur. Since we can pass the incorrect code back into the model multiple times, we can also account for multiple errors, allowing the model to focus on fixing a small, singular error, rather than regenerating the entire program.

B. Quantum Error Correction Enhancement

The quantum programs generated after passing through the code generation agents and semantic analyzer do not contain any form of quantum error correction. As a result, running these codes on a physical quantum computer would contain high levels of noise, which reduces the reliability of the generated quantum programs. Therefore, it is vital to incorporate the mechanism of error correction to mitigate the potential of quantum error.

To do this, we include a third agent in the framework, which generates a decoder for a Quantum Error Correction (QEC) surface code [18], [27]. As shown in Figure 2, this agent uses the topology of the quantum device and the number of logical qubits provided to generate a decoder that allows a surface error correction code to be used when running the algorithm. This is applied after the code has been generated and does not alter its semantics, but does add additional qubits in order to generate the appropriate surface code lattice. It maps the logical qubits required onto a subset of physical qubits, which allows us to take advantage of the topology of the device to implement a surface code. By applying this code, we extend the average qubit lifetime, which means we are less likely to see qubits affected by quantum noise. This, in turn, reduces the amount of error present in the results, improving their accuracy.

The main drawback of using a surface code is that they are topology-specific. This results in the model needing to be re-trained every time you want to run the code on a different device, which is extremely inefficient, so finding a way to make a QEC surface code that is topology-agnostic is vital to improving the framework's efficiency.

C. Prompt Engineering Optimisations

We adopted several techniques to mitigate the unique challenges posed by generating quantum code. The first was Retrieval-Augmented Generation (RAG) [9], to provide the model with knowledge of both the structure of quantum algorithms and the structure of the Qiskit library. We then implemented Chain of Thought (CoT) [8] and Structure Chain of Thought (SCoT) [28] prompting, as these have been shown to improve the accuracy of code generation. Our goal for these techniques was to improve the semantic accuracy of the code generated as we found that the lack of algorithmic knowledge available often led to syntactically correct but nonsensical code being generated. The final technique we implemented was multi-pass inference, which was intended to improve the syntactic accuracy of the code.

For our CoT and SCoT prompts, we manually created the first 5 prompts from our testing set using the same techniques demonstrated in previous code generation work [28]. For each subsequent prompt, we manually created a version in both CoT and SCoT and generated the same version using the GPT-4o model [11]. Figure 2 shows how generated CoT and SCoT prompts compare the original prompt that was provided to the model.

We collated two different RAG datasets. The first scraped the official Qiskit documentation repository on Github, taking the documentation for the latest Qiskit version, including Qiskit-adjacent libraries such as *qiskit-ibm-runtime*. The goal of this dataset is to improve the model's understanding of the library structure, reducing the amount of import errors due to deprecated features. The second consisted of a collection of guides and tutorials explaining the ideas behind and structures of a collection of quantum algorithms. The goal of this dataset is to improve the model's knowledge of algorithms, improving its

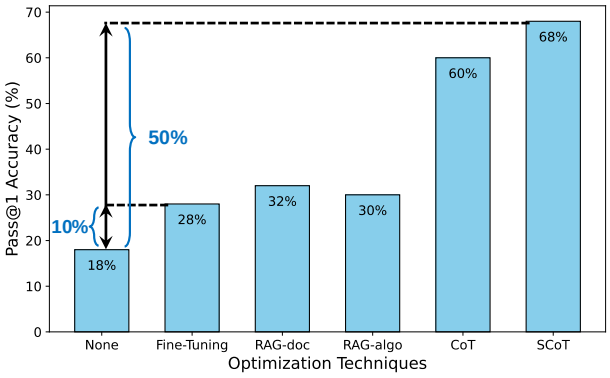


Fig. 3: The percentage of results that were semantically and syntactically valid for each technique.

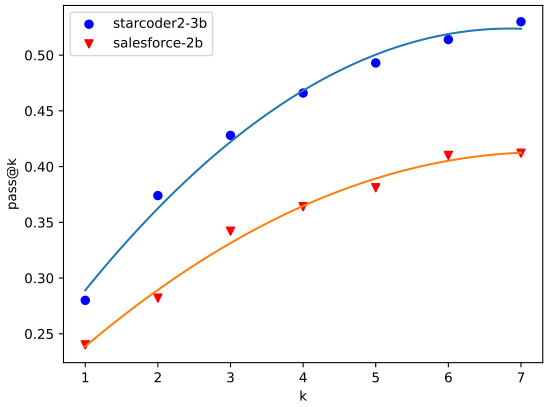


Fig. 4: pass@k values for fine-tuned models.

semantic accuracy. We used the langchain [29] and ragatouille [30] libraries to create the augmented prompts.

V. Experiments

A. Experimental Setup

We chose the StarCoder models [4] for fine-tuning since they were pre-trained on a wide corpus of languages and library files, making them suitable for adapting to new programming languages [31]. Furthermore, StarCoder models have demonstrated strong performance across a variety of coding tasks, showing their versatility and suitability for learning in new coding environments [3]. In our experiments, we selected StarCoder-3B as the main model for evaluation.

For fine-tuning, we used the *transformers* library [32] with LoRA adapter [33]. To facilitate the use of our custom dataset, we converted it into multiple chunked sets, with randomly applied Fill-in-the-Middle (FIM) transformations [34]. The chunk size was calculated upon training based on the amount of data provided. We found that the optimal FIM rate (the percentage of chunks to which a FIM transformation is applied) was 0.1. Training was conducted for 1400 steps with a batch size of 4. The learning rate was linearly increased from 0 to 3×10^{-4} over the first 100 warm-up steps and subsequently decayed using a cosine scheduler.

When testing our methods using our testing suite, each optimization method was evaluated independently. The CoT and SCoT prompts were passed directly into the model, upon which it generated the code. To further evaluate the accuracy of the fine-tuned model with no optimizations, we used the *transformers* library [32] to create a sample of results for each prompt and applied the pass@k metric [20] to obtain our results. Additionally, it was also our intention to compare our model with the recent IBM Qiskit Assistant [7] model. However, their model is paywall-locked, so it was not possible to get a comparison, despite multiple attempts of contacting the authors.

B. Overall Accuracy Results

Using our test suite, we were able to evaluate the accuracy of applying our multi-agent framework with the optimization techniques proposed. The overall results of the experiments are shown in Figure 3. The results provided are the percentage of prompts that are both syntactically and semantically correct. By training on the dataset of Qiskit repositories, we were able to increase the pass@1 metric by 10%, up to 28% overall. Figure 4 presents the accuracy of the model using the pass@k metric for k values ranging from 1 to 7, taken by generating a sample of prompts. The graph indicates that accuracy gains diminish as the sample size increases. Specifically, beyond 5 samples per prompt, we observe less improvement in the pass@k metric, indicating diminishing returns with additional samples. To investigate the effect of different model sizes on the model's pass@k accuracy, we also evaluate Salesforce-2B for comparison. As shown in Figure 4, the accuracy of Salesforce-2B model is consistently worse than that of StarCoder-3B across different k values. Therefore, the rest of the experiments adopt StarCoder-3B for evaluation.

C. Effects of RAG, CoT, and SCoT

As shown in Figure 4, applying RAG [9] optimization had negligible impact on the overall accuracy of the model. Whilst the algorithmic knowledge showed 4% improvement relative to the documentation knowledge, there is limited real-world benefit introduced by either method due to the negligible improvement.

The results for CoT [8] and SCoT [28] show remarkable increases, with CoT leading to an increase of 32% and SCoT an increase of 40% from the fine-tuned model. The majority of this increase came from a better semantic understanding of how the algorithms are structured, suggesting that the initial training dataset did not contain many good examples of such algorithms.

D. Effects of Multi-Pass Inference and QEC

We found that applying multi-pass inference into the semantic analyzer model can improve the accuracy to 34% using triple passes. However, additional inference passes, despite incurring higher computational costs, yielded limited benefit. The diminishing accuracy improvement is likely due to the nature of the error, which was mostly the misuse of imports or the use of deprecated code. If these issues were resolved,

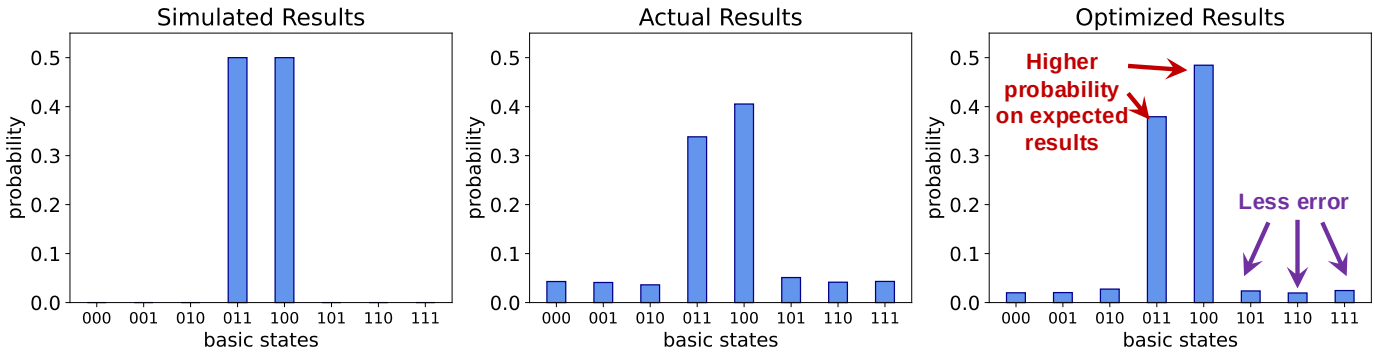


Fig. 5: Results for Grover's Algorithm Experiments.

the results did demonstrate evidence that multi-pass inference could be used to improve the model's ability to resolve other syntactic and semantic errors.

Leveraging our multi-agent framework with the support of quantum error prediction and correction, we were able to reduce the amount of quantum error in our experiment results. Figure 5 shows an example of Grover's Algorithm [23] under a quantum noise environment, with and without the use of our framework. Compared with the results without quantum error optimization, the optimized results demonstrate a general increase in the probability of the expected quantum states, as well as a decrease in observed errors across other quantum states throughout all experiments. Although this example with surface code applied is topology-dependent, it is sufficient to demonstrate the effectiveness of our framework in reducing quantum error.

E. Observation and Future Directions

Different from LLM-assisted code generation for general-purpose programming languages, our experimental results reveal that different optimization techniques have vastly distinct effects on quantum code generation. Applying RAG enhancement resulted in only a marginal increase in pass@1 accuracy. This technique aimed to enhance the model's understanding of the library structure and knowledge of algorithms. The limited accuracy increase is likely attributable to the fact that the documentation available for Qiskit is not up to date, preventing the effective mitigation of import errors. In contrast, other techniques such as Chain-of-Thought (CoT) reasoning introduced significant accuracy improvement. This improvement is attributed to the enhanced semantic knowledge provided to the model, allowing it to correctly structure the generated quantum program. This approach likely outperformed RAG, as it allowed us to more directly inform the model's decision-making process, rather than inferring how the algorithms work from the, rather limited, dataset we provided.

One limitation encountered during our experiments was the limited dataset sizes, which can be evidenced by both the poor increase in the fine-tuned models' pass@1 accuracy and the lack of improvement from applying the RAG technique. An important direction of our future work is to collect a larger and higher-quality dataset. This presents challenges due to the lack of data online and the rapidly changing nature of the field. One potential solution is to scrape other sites, such as the official Qiskit website and forums, while supplementing the dataset with hand-built examples. Other promising directions include the generation of synthetic data [35], federated learning [36], [37] and inference-time optimizations [38]–[40].

Moreover, a key area for future development is creating a topology-agnostic QEC decoder model. Currently, our work includes a model that is topology-specific, requiring retraining each time algorithms are adapted for different quantum computers. Whilst our work has shown that using a model like this can aid in producing fault-tolerant code, having a topology-specific model is not a scalable method. Developing a model capable of generating quantum error correction decoders for arbitrary topologies would significantly enhance the scalability of fault-tolerant quantum code generation.

VI. Conclusion

This paper proposes a novel multi-agent framework to provide an accurate, fault-tolerant quantum computing code generation LLM. We propose a three-agent system, with the first being a code generation based on LLM models. We investigate the effect of various conventional optimizations on our model, including Structured Chain of Thought. The second agent is a semantic analyser, which uses multi-pass inference and RAG to improve the accuracy of the generated code. The results observed during the code generation experiments suggest that a model with this purpose is necessary, as the code generation process lacks significant algorithmic knowledge. These techniques could be improved by collecting a larger dataset for RAG, which could also be combined with the multi-pass inference to improve the model's ability to understand why an error occurred. The third agent is a Quantum Error Correction model that predicts QEC surface code decoders. Our results showed a decrease in the average noise present in an experiment on IBM quantum computers. Overall, our framework makes a promising step forward in creating a tool to generate fault-tolerant quantum programs, which would greatly improve the accessibility of a complex field. In future work, we plan to extend our framework with advanced techniques such as test-time optimization and federated learning to further improve performance and scalability.

REFERENCES

[1] M. U. Hadi, Q. Al Tashi, A. Shah, R. Qureshi, A. Muneer, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu *et al.*, "Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects," *Authorea Preprints*, 2024.

[2] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, "A survey of embodied AI: From simulators to research tasks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 2, pp. 230–244, 2022.

[3] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[4] A. Lozhkov, R. Li, L. Ben Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[5] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[6] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui, "Agentcoder: Multiagent-code generation with iterative testing and optimisation," *arXiv preprint arXiv:2312.13010*, 2023.

[7] N. Dupuis, L. Buratti *et al.*, "Qiskit code assistant: Training LLMs for generating quantum computing code," *arXiv preprint arXiv:2405.19495*, 2024.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[10] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[11] OpenAI, "Gpt-4 technical report," 2023, accessed: 2023-08-15. [Online]. Available: https://openai.com/research/gpt-4

[12] Y. Talebirad and A. Nadiri, "Multi-agent collaboration: Harnessing the power of intelligent llm agents," *arXiv preprint arXiv:2306.03314*, 2023.

[13] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.

[14] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[15] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[16] E. Gerjuoy, "Shor's factoring algorithm and modern cryptography. an illustration of the capabilities inherent in quantum computers," *American journal of physics*, vol. 73, no. 6, pp. 521–540, 2005.

[17] A. M. Steane, "Simple quantum error-correcting codes," *Physical Review A*, vol. 54, no. 6, p. 4741, 1996.

[18] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 86, no. 3, p. 032324, 2012.

[19] J. Gai, H. Chen, Z. Wang, H. Zhou, W. Zhao, N. Lane, and H. Fan, "Exploring code language models for automated HLS-based hardware generation: Benchmark, infrastructure and analysis," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 988–994.

[20] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[21] S. Vishwakarma, F. Harkins, S. Golecha, V. S. Bajpe, N. Dupuis, L. Buratti, D. Kremer, I. Faro, R. Puri, and J. Cruz-Benito, "Qiskit humaneval: An evaluation benchmark for quantum code generative models," in *2024 IEEE International Conference on Quantum Computing and Engineering*, vol. 1. IEEE, 2024, pp. 1169–1176.

[22] Q. Community, "Qiskit machine learning," https://github.com/qiskit-community/qiskit-machine-learning, 2023, accessed: 2024-07-27.

[23] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing*, 1996, pp. 212–219.

[24] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, and A. Zeilinger, "Experimental quantum teleportation," *Nature*, vol. 390, no. 6660, pp. 575–579, 1997.

[25] M. Santha, "Quantum walk based search algorithms," in *International Conference on Theory and Applications of Models of Computation*. Springer, 2008, pp. 31–46.

[26] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll, "Quantum annealing: A new method for minimizing multidimensional functions," *Chemical physics letters*, vol. 219, no. 5-6, pp. 343–348, 1994.

[27] R. Sweke, M. S. Kesselring, E. P. van Nieuwenburg, and J. Eisert, "Reinforcement learning decoders for fault-tolerant quantum computation," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 025005, 2021.

[28] J. Li, G. Li, Y. Li, and Z. Jin, "Structured chain-of-thought prompting for code generation," *ACM Transactions on Software Engineering and Methodology*, 2023.

[29] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[30] B. Clavie, "Ragatouille: State-of-the-art document retrieval," https://github.com/bclavie/RAGatouille, 2024, accessed: 2024-08-21.

[31] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "Starcoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[32] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[34] M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen, "Efficient training of language models to fill in the middle," *arXiv preprint arXiv:2207.14255*, 2022.

[35] Z. Yang, N. Band, S. Li, E. Candès, and T. Hashimoto, "Synthetic continued pretraining," *arXiv preprint arXiv:2409.07431*, 2024.

[36] W. Zhao, H. Fan, S. X. Hu, W. Zhou, and N. Lane, "CLUES: Collaborative private-domain high-quality data selection for llms via training dynamics," *Advances in Neural Information Processing Systems*, vol. 37, pp. 141 183–141 207, 2024.

[37] W. Zhao, Y. Chen, R. Lee, X. Qiu, Y. Gao, H. Fan, and N. D. Lane, "Breaking physical and linguistic borders: Multilingual federated prompt tuning for low-resource languages," in *The Twelfth International Conference on Learning Representations*, 2024.

[38] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM test-time compute optimally can be more effective than scaling model parameters," *arXiv preprint arXiv:2408.03314*, 2024.

[39] H. M. Chen, W. Luk, K. F. C. Yiu, R. Li, K. Mishchenko, S. I. Venieris, and H. Fan, "Hardware-aware parallel prompt decoding for memory-efficient acceleration of LLM inference," *arXiv preprint arXiv:2405.18628*, 2024.

[40] H. M. Chen, F. Tan, A. Kouris, R. Lee, H. Fan, and S. I. Venieris, "Progressive mixed-precision decoding for efficient LLM inference," *arXiv preprint arXiv:2410.13461*, 2024.

MambaOPU: An FPGA Overlay Processor for State-space-duality-based Mamba Models

Shaoqiang Lu[1,2,⋆], Xuliang Yu[3,⋆], Tiandong Zhao[4], Siyuan Miao[4], Xinsong Sheng[3], Chen Wu[1,†], Liang Zhao[3,†], Ting-Jung Lin[1] and Lei He[1,4]
[1] Ningbo Institute of Digital Twin, Eastern Institute of Technology, Ningbo, China [2] Shanghai Jiao Tong University, Shanghai, China
[3] Zhejiang University, Hangzhou, China [4] University of California, Los Angeles, USA †E-mail: {cwu@idt.eitech.edu.cn, lzhao2020@zju.edu.cn}

***Abstract*—State-space models (SSMs), such as Mamba, have emerged as a promising alternative to Transformers. However, the recently developed Mamba2, based on state space duality (SSD), is highly memory-bound and suffers from limited computation efficiency. This inefficiency arises from its irregular broadcast element-wise multiplications and structured sparse computations. In this work, we propose MambaOPU, an FPGA overlay processor, to accelerate SSD. First, to reduce memory overhead, we introduce a software-hardware co-optimized operator fusion framework. Specifically, operator merging combines adjacent broadcast multiplication and summation operations into a single descriptor, while operator backward shifting embeds segment multiplication into subsequent operations. Both techniques shorten the computation path and improve computation efficiency. Second, to enhance sparse computation efficiency, we skip zero-region computations using a tensor-reorder-and-group algorithm combined with a sparse-predefined data fetcher. Additionally, since Mamba integrates linear operations with SSD, we develop a reconfigurable systolic array to improve data reuse across different computation modes. Extensive experiment results demonstrate that MambaOPU achieves up to $1812\times$ and $880.79\times$ higher normalized throughput and up to $12908\times$ and $24.27\times$ higher energy efficiency over Intel Xeon Gold 6348 CPU and NVIDIA A100 GPU, respectively.**

I. Introduction

Large language model (LLM) based technology has a profound impact on daily life via applications such as smart assistants, content creation [1] and personalized recommendations [2]. The impact comes from the Transformer architecture [3]–[5], renowned for its self-attention mechanism, which excels in capturing long-range dependencies within sequences. However, Transformers face challenges in the quadratic time complexity and substantial memory demands, which severely limit inference performance, especially on edge computing platforms. A new architecture called Mamba [6] offers an efficient alternative inspired by state-space models. Mamba effectively captures complex dependencies in sequential data with near-linear computational scaling through parallel scanning and recurrent computation, making it a promising solution for resource-constrained environments.

To accelerate the inference of Transformer-based LLMs, most FPGA and ASIC accelerators have leveraged sparse-computing optimizations and memory-centric strategies to boost throughput and energy efficiency. The former identifies weak-relevant tokens and reduces redundant computations through pruning [7]–[16] and quantization [17]–[21], while the latter adopts techniques like model splitting, computation graph fusion and dataflow optimization [22]–[28] to balance memory load and computation efficiency. Despite their exceptional performances, these techniques cannot be directly applied to Mamba. The selective SSM mechanism necessitates recurrent computation [29] and element-wise operations, which are not well-suited for tensor-core performance. To resolve these issues, MARCA [30] proposes the first reconfigurable processing element (PE) array to optimize linear, element-wise operations and nonlinear computations, significantly improving energy efficiency and inference speed. On the other hand, Mamba2 [31] translates the SSM-recurrence computations into block semi-separable matrix multiplications through introducing SSD mechanism, thereby effectively leveraging GPU parallelism. However, accelerating Mamba2 still requires tailored hardware designs as most MARCA techniques do not apply.

Fig. 1 summarizes the challenges of developing Mamba2 accelerators: **(1) Large Memory Overhead Resulted from Tensor-Expansion**. Broadcast multiplications in SSD phase expand smaller tensors to larger ones for element-wise multiplications, significantly increasing memory footprint and traffic during inference. The element-wise operations between high-dimensional intermediate representations will also impose significant memory dependencies. Specifically, the peak memory usage reaches up to 41.55GB and 10.5GB for dual attention and recurrence modes, respectively. **(2) Limited Performance for Element-wise Multiplication on Existing Hardware**. Broadcast multiplications, *i.e.,* tensor expansion followed by element-wise multiplication, tend to have limited computation efficiency on existing hardware equipped with optimized units for tensor contraction, *e.g.,* tensor core [32] or systolic array [33], due to the mismatch of computation patterns. Profiling reveals that broadcast multiplications consume 46.68% of total inference time while element-wise operations dominate the SSD computation with 83.88% latency. **(3) Redundant Computation for SpMM and SDMM**. The inter-block sparse-sparse matrix multiplications (SpMM) resulting from zero-padded input sequences, along with sparse-dense matrix multiplications (SDMM) tied to the lower triangular causal matrix, introduce four classes of structural sparsity patterns. These sparsity patterns scale quadratically or linearly with the sequence length, leading to dynamic ineffective calculations. For instance, sparsity rates decrease from nearly 100% to zero as sequence length increases in Mamba2 computation phases S_{a1}, S_{a5}, and S_{a6}, while causal matrix multiplications maintain a sparsity close to 50% in phases S_{r4} and S_{r6} for long sequences (see Table. I for the phase definition).

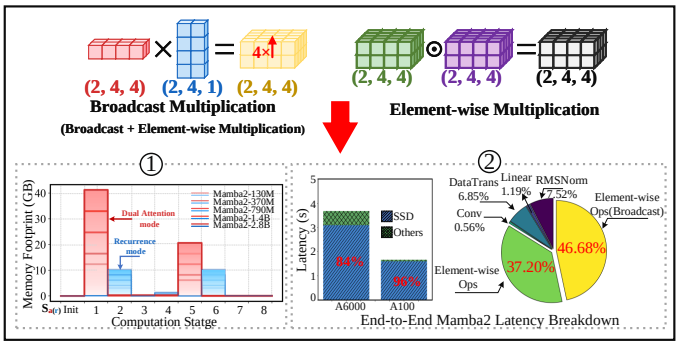
To this end, we propose an FPGA overlay processor for Mamba2, named MambaOPU. It combines the merits of operator fusion and reconfigurable sparse computation architecture, to support efficient always-on-chip SSD computation. The four key contributions of this work are summarized as follows:

- A software-hardware co-optimized operator fusion framework through operator merging and backward shifting to alleviate the memory pressure induced by tensor expansion.
- A sparsity-predefined prediction engine combining tensor-reorder-and-group algorithm and data fetcher for sparse computation acceleration.
- A reconfigurable systolic array that can efficiently support four computation modes (i.e., GEMM, SpMM, element-wise operations and segment-mul).
- Extensive experiments on Mamba2 models consistently demonstrating the significant advantages of MambaOPU over CPUs and GPUs, in terms of end-to-end latency, normalized throughput and energy efficiency.

II. Background

State Space Duality. Mamba1 applies the selective thinking to modify LTI-SSM [34], allowing parameter matrices to vary with

⋆ Equal contribution, † Corresponding authors. This work was supported by Ningbo key R&D program (2023Z214) and Key technologies of "Science and Technology Innovation in Yongjiang 2035"(2024Z283) and BTD Inc.



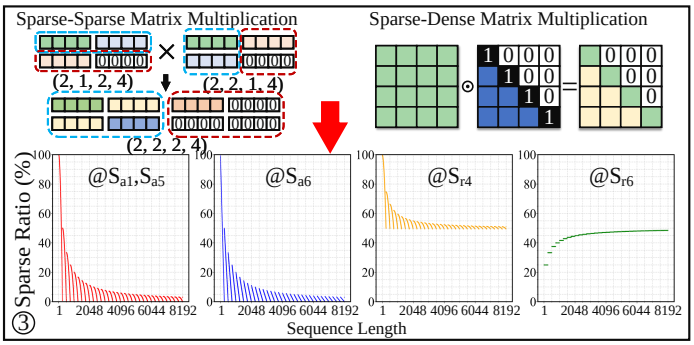


Fig. 1: Challenges in Mamba2 computation: ① Large memory overhead resulted from broadcast multiplication, ② Limited performance for element-wise multiplication on existing hardware, and ③ Redundant computation for SpMM and SDMM.

Note: The detailed definitions of the different computation stages (i.e., S_{ai} or S_{ri}) within SSD computation are provided in the Table I.

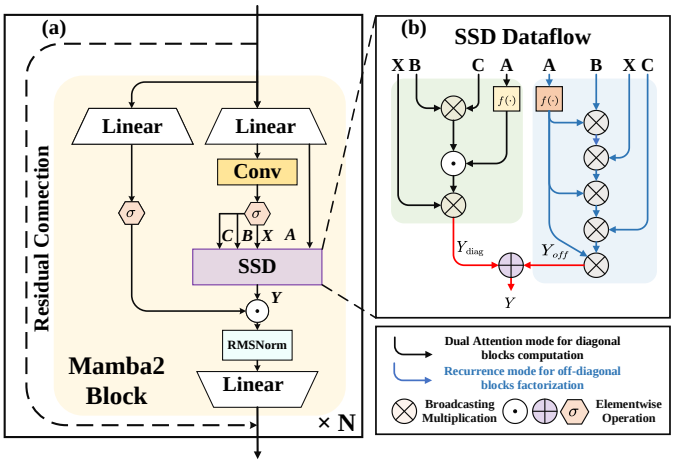


Fig. 2: The key components and SSD dataflow within Mamba2.

inputs and enhancing focus on more informative tokens. However, this recursive process is difficult to accelerate on GPUs, as it does not align with the parallel convolution paradigm of LTI systems. SSD is introduced in Mamba2 to unify selective SSMs with the attention mechanism, enhancing computational efficiency. Specifically, the selective SSMs are transformed into matrix multiplication as follows:

$$h_t = \sum_{s=0}^{t} A^{\times}_{t:s} B_s x_s = A_t \cdots A_1 B_0 x_0 + A_t \cdots A_2 B_1 x_1 + \cdots + B_t x_t$$
$$y_t = \sum_{s=0}^{t} C_t^T A^{\times}_{t:s} B_s x_s = Mx, M_{ji} := C_j^T A_j \cdots A_{i+1} B_i \quad (1)$$

where M denotes the causal matrix which is N-sequentially semi-separable [35] (NSS, N denotes the state dimension). For an arbitrary element M_{ji}, the vectors B_i, C_j and matrix A_j represent the input/output transformation and state transition matrix [36], respectively. Moreover, M can be reformulated in linear attention notation when A_j has reduced scalar-time structure (i.e., $A = aI$):

$$M = L \circ (CB^T) = \text{1SS}(a) \circ (CB^T), M_{ji} = A_{j:i} \cdot (C_j^T B_i) \quad (2)$$

where L denotes causal matrix with 1SS structure and the computation can be implemented using a *segment-sum* operator. Furthermore, M degenerates to a form dependent only on a, (i.e., $M = \text{1SS}(a)$) when C_j and B_i are scalars. Consequently, $y = Mx$ in Eq. 1 can be computed using the linear recurrence:

$$y_t = a_t(a_{t-1:0}x_0 + \cdots + a_{t-1:t-1}x_{t-1}) + a_{t:t}x_t = a_t y_{t-1} + x_t. \quad (3)$$

Mamba2 Workflow Mamba2 translates the SSD computation into a hybrid framework supporting linear recurrence and dual attention

TABLE I: Detailed phase description for Mamba2 algorithm

Phase	Description	Phase	Description	Phase	Description
S_{a1}	$G_{\text{temp}} = B \odot C$	S_{a2}	$G = \text{Sum}(G_{\text{temp}}, \text{dim} = -1)$	S_{a3}	$M_{\text{temp}} = G \odot \text{1SS}(a)$
S_{a4}	$M = \text{Sum}(M_{\text{temp}}, \text{dim} = -1)$	S_{a5}	$Y_{\text{temp}} = M \odot H_{\text{state}}{}'$	S_{a6}	$Y_{\text{diag}} = \text{Sum}(Y_{\text{temp}}, \text{dim} = 3)$
S_{r1}	$B_{\text{decay}} = B \odot D_{\text{state}}$	S_{r2}	$H_{\text{decay}} = B_{\text{decay}} \odot H_{\text{state}}{}'$	S_{r3}	$\text{State} = \text{Sum}(B_{\text{decay}}, \text{dim} = 3)$
S_{r4}	$\text{Result}_{\text{temp}} = \text{state} \odot \text{1SS}(A_{\text{cum}})$	S_{r5}	$\text{Result} = \text{Sum}(\text{Result}_{\text{temp}}, \text{dim} = -1)$	S_{r6}	$C_{\text{state,temp}} = C \odot \text{Result}$
S_{r7}	$C_{\text{State}} = \text{Sum}(C_{\text{state,temp}}, \text{dim} = -1)$	S_{r8}	$Y_{\text{off}} = C_{\text{state}} \odot \exp(A_{\text{cum}})$	S_{o1}	$H_{\text{state}}{}' = H_{\text{state}} \odot \text{Softplus}(X)$
S_{o2}	$D_{\text{residual}} = H_{\text{state}} \odot D$	S_{o3}	$Y = Y_{\text{diag}} + Y_{\text{off}} + D_{\text{residual}}$	S_{o4}	$Y_{\text{scan}} = \text{RMSNorm}(Y \odot \text{Gate})$

computation by leveraging the 1SS and NSS properties within M. The key components and flow of Mamba2 are shown in Fig. 2, where each block consists of layer normalization, linear projection, convolution, SSD block, and residual connection. Rather than interleaving Mamba2 blocks with feed-forward blocks, Mamba2 is built by stacking blocks with standard normalization and residual connections.

During SSD processing, the input feature is used to create a map from $[X, A, B, C]$ to Y through two computational branches. In Fig. 2, the path indicated by black solid lines denotes dual attention mode for diagonal block computation (Y_{diag}), while the path in blue solid lines indicates inter-chunk state transitions (Y_{off}) in linear recurrence mode. The full computation phases of Mamba2 are summarized in Table I, where S_a and S_r denote the core phases of dual attention and recurrence computation branches respectively, and S_o covers other operations within the SSD computation.

III. Software Stack

The software stack of MambaOPU shown in Fig. 3 (a) includes six key components: data-flow graph unrolling, operator fusion, data allocation, array mode selection, memory scheduling and event generation. Among them, the operator fusion and data allocation are crucial for reducing the memory load and enabling on-chip SSD computation. The software stack runs on a Host-CPU to compile the Mamba model, generating instructions and data, which are then transmitted to the MambaOPU via PCIe.

A. Operator Fusion

To reduce feature-induced memory access, an operator fusion framework is designed to merge operators and shift them backward.

Operator Merge: We merge the matrix broadcast multiplication and summation into a descriptor, thus unifying them with linear-based operations. As illustrated in Fig. 4, an arbitrary row-vector y_i within Y can be yielded via vector-matrix multiplication between x_i and the weight matrix W. Alternatively, y_i can be yielded by broadcast multiplication of x_i and the transpose of W, followed by column-direction summation. Each element of y_i can be expressed as follows:

$$y_{ij} = \underbrace{x_i W[:, j] = \sum_{k=1}^{K} x_{i,k} w_{k,j}}_{\text{(a) Linear-based Operations}} = \underbrace{x_{i,1}w_{1,j} + x_{i,2}w_{2,j} + \cdots + w_{i,K}w_{K,j}}_{\text{(b) Fused Operations}} \quad (4)$$

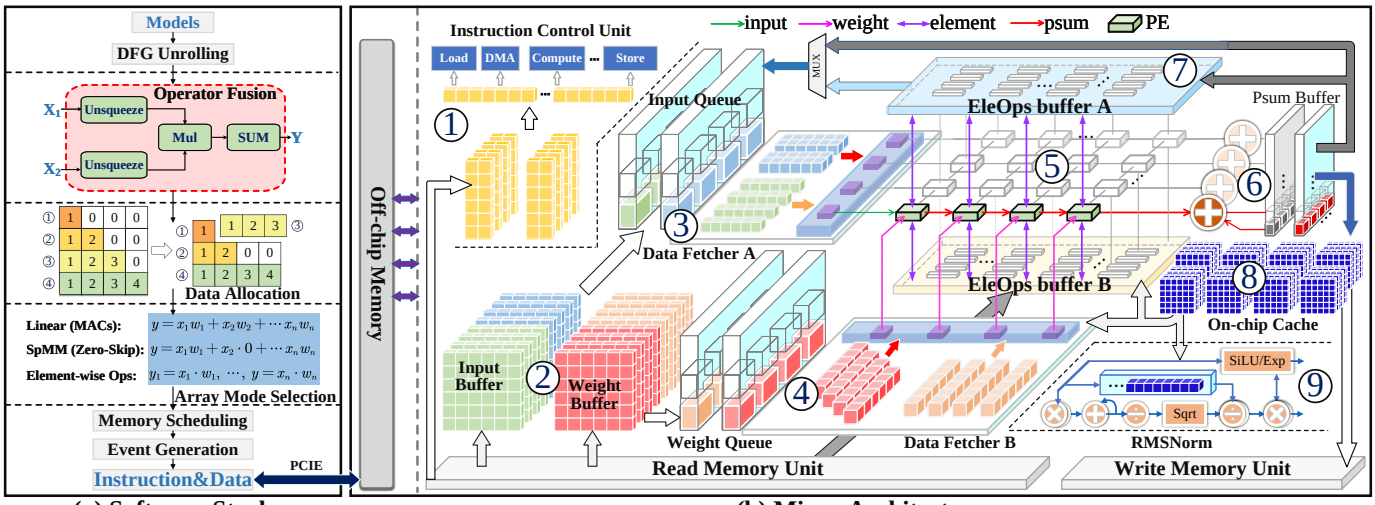


Fig. 3: Software stack and micro-architecture ① instruction control unit (ICU), ② input and weight buffer, ③④ data fetcher A and B, ⑤ reconfigurable systolic array, ⑥ psum buffer,⑦ element-wise buffer, ⑧on-chip cache and ⑨ special function unit (SFU).

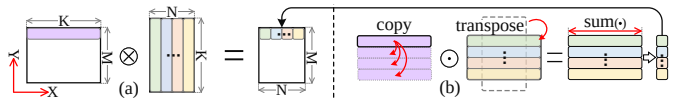


Fig. 4: Unifying (a) Linear-based operations with (b) broadcast multiply-and-accumulate.

Based on the **Operator Merge**, we can merge the operator pairs $\{S_{a1}, S_{a2}\}$, $\{S_{a3}, S_{a4}\}$, $\{S_{a5}, S_{a6}\}$, $\{S_{r2}, S_{r3}\}$, $\{S_{r4}, S_{r5}\}$ and $\{S_{r6}, S_{r7}\}$ into a one-phase operation.

Operator Backward Shift: The causal matrix M is first generated via *segment-mul* (containing *segment-sum* and *element-wise* exp), and then multiplied element-wise with the intermediate states H'_{state}. To shorten the computation path, we mask *segment-sum* by embedding *segment-mul* into element-wise multiplications between M and H'_{state}, based on the 1SS property $a^{\times}_{j:i} = a_{j-1} \cdot a^{\times}_{j-1:i}$, as expressed:

$$M_{ji} = m^{\times}_{ji} = \prod_{k=i}^{j-1} m_k = \exp(\sum_{k=i}^{j-1} a_k), m_k = \exp(a_k),$$
$$y_{ji} = M_{ji}h_{ji} = m^{\times}_{ji}h_{ji} = m^{\times}_{ji}h_j i = m_j \cdot m^{\times}_{j,i-1}h_{ji} = m_j y_{j,i-1}, \tag{5}$$

where a_k, h_{ji} are the corresponding elements within state transition matrix A and hidden state H'_{state}, respectively.

B. Data Allocation

We deploy two computation modes in each core of MambaOPU to support tile-granularity multiply-and-accumulate (MAC) operations, which are GEMM (yellow) and SpMM (green) shown in Algorithm 1. Each core can independently acquire the initial state, compute the transition state based on the input and return the output. For very long sequences with size of *Seqlen*, we partition the sentences into *ratio* chunks and distribute them across *Ncore* cores to achieve sequence parallelism, where *ratio* is defined:

$$\text{ratio} = (\text{Csize - Seqlen \% Csize} + \text{Seqlen})/\text{Csize}. \tag{6}$$

GEMM Mode can support linear layer computation and fused tensor broadcast MAC operations. To alleviate memory footprint, we can fetch the corresponding tiles for matrix multiplication and accumulation per cycle, based on the pre- and post-broadcast address mapping list. Once the tile data are prepared in a systolic-skew arrangement, vector-wise MAC is executed in the PE array.

Algorithm 1 Data Allocation and Mapping Algorithm

Input: NCore, Ratio, Head, Csize, Ssize, Tsize, Sflag, Mask, $\mathcal{X} \in \mathcal{R}^{\text{Ratio}\times 1\times \text{Csize}\times \text{Head}\times \text{Ssize}}$, $\mathcal{Y} \in \mathcal{R}^{\text{Ratio}\times \text{Csize}\times 1\times \text{Head}\times \text{Ssize}}$
Output: $\mathcal{Z} \in \mathcal{R}^{\text{Ratio}\times \text{Head}\times \text{Csize}\times \text{Csize}}$

for $r = 1$ **to** Ratio//NCore **do**
  for $h = 1$ **to** Head **do**
    for $c_1 = 1$ **to** Csize//Tsize; $c_2 = 1$ **to** Csize//Tsize; $s = 1$ **to** Ssize//Tsize **do**
      Xtiles, Ytiles ← **FetchTiles**($\mathcal{X}$, $\mathcal{Y}$)
      if Sflag is False **then**
        for $t_1 = 1$ **to** Tsize; $t_2 = 1$ **to** Tsize **do**
          RESULT[t_1, t_2]← **MAC**(Xtiles[t_1], Ytiles[t_2])
        end for
        $\mathcal{Z}[r, h, c_1 \times \text{Tsize} : (c_1+1) \times \text{Tsize}, c_2 \times \text{Tsize} : (c_2+1) \times \text{Tsize}] +=$ RESULT
      else
        MXtiles, MYtiles ← **BroadcastMask**(Xtiles,Ytiles, Mask)
        RXtiles **or** RYtiles ← **Reorder**(MXtiles) **or Reorder**(MYtiles)
        for $t_1 = 1$ **to** Tsize// $2+1$; $t_2 = 1$ **to** Tsize **do**
          RESULT[t_1, t_2]← **MAC**(RXtiles[t_1, t_2], RYtiles[t_1, t_2])
        end for
        $\mathcal{Z}[r, h, c_1 \times \text{Tsize} : (c_1+1) \times \text{Tsize}, c_2 \times \text{Tsize} : (c_2+1) \times \text{Tsize}] +=$ **Recover**(RESULT)
      end if
    end for
  end for
end for

SpMM Mode can eliminate ineffective computations and enhance the hardware efficiency by leveraging the sparsity within the output matrix. Specifically, zero-masking is applied to certain operand positions based on the triangular sparse prior within M. Then, more compact tensors can be obtained for matrix multiplication-and-accumulation through tensor-reorder-and-group transformation. For example, the optimized computation process for B and C can be unfolded as follows:

(1) B, C are broadcasted to achieve dimension matching.

(2) The broadcasted tensors B' and C' require the application of a lower triangular mask. The zero elements in either tile $B'[i]$ or $C'[i]$ will be filled with the non-zero elements from tile $B'[N-2-i]$ or $C'[N-2-i]$, where $i = 0, 1, 2, \cdots, \lfloor M/2 \rfloor$.

(3) The grouped tensor performs vector-wise MAC along a specified dimension to generate results, which are then reorganized into the final sparse output via *Recover*(·).

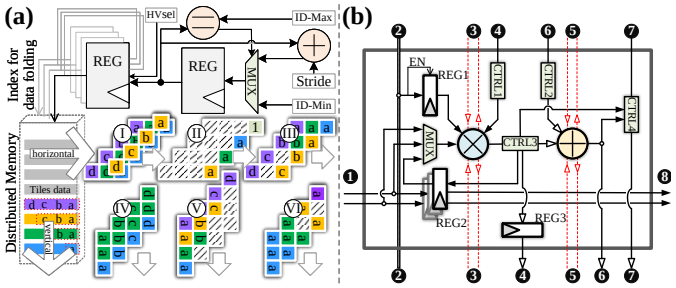


Fig. 5: (a) Sparse-predefined data fetcher supporting various computation modes (I/IV: S_{a1}, II/V: S_{a3}, III/VI: S_{a5}) and (b) PE with its in-out signals: ❶ CUR_IN, ❷ W_IN, ❸ ELE_MUL, ❹ CAS_MUL, ❺ ELE_ADD, ❻ CAS_MAC, ❼ BYPASS, ❽ NEXT_IN.

TABLE II: Computation modes of MambaOPU

Mode	Descriptions
Load/Store	Load data from DDR to on-chip buffer,Store data from on-chip buffer to DDR.
Fetch	Schedule data in a computation-friendly manner and correctly feed into the computing array.
GEMM	Support linear-based operations and broadcast multiplications: $C_{i,j} = \sum_{l=1}^{k} A_{i,l} \times B_{l,j}$
SpMM	sparse-sparse and sparse-dense: $C_{i,j} = \sum_{(i,l)\in \mathrm{NZ}(A)} A_{i,l} \times B_{l,j},\ \forall (i,j) \in \mathrm{NZ}(C)$
SegMul	Support causal matrix calculation $L(i,j) = \prod_{k=i}^{j} l_k$
EleOps	Support element-wise operations: $C_{i,j} = A_{i,j} \cdot B_{i,j}$ or $C_{i,j} = A_{i,j} + B_{i,j}$
SFU	RMSNorm(·), exp(·), SiLU(·), and other non-linear functions

C. Computation Mode Selection

We integrate high-level packaged instructions within the compiler to enhance support for various computation modes. This includes eight instructions listed in Table II. *Load* and *Store* denote data transfer between off-chip and on-chip memories, while *Fetch* configures the dataflows to align with the specific feeding requirements. *GEMM*, *SpMM*, and *EleOps* represent dense GEMM, optimized sparse SpMM, and element-wise operations, respectively. *SegMul* generates causal mask in the SSD phase, and *SFU* manages nonlinear computations such as RMSNorm(·), EXP(·), and SiLU(·).

IV. Hardware Micro-architecture

The micro-architecture of MambaOPU is shown in Fig 3 (b), which consists of eight key components: an instruction control unit (ICU), input and weight buffers, sparse-predefined data fetchers A/B, a reconfigurable systolic array, partial summation (psum) buffers, element-wise operation (EleOps) buffers A/B and a special function unit (SFU). Importantly, the reconfigurable systolic array and the data fetcher working together enable the implementation of different computational modes and achieve efficient end-to-end inference.

A. Sparse-predefined Data Fetcher

To support efficient reading of data streams and ensure compatibility across different computation modes, we design a sparse-aware data fetcher composed of an address sequence generator (ASG), a register group, and a distributed queue, as shown in Fig. 5 (a). Once the data fetcher receives the start signal from ICU, the ASG increments from *ID-min* by a step size of *Stride* until *ID-max*, where *Stride* can be configured by the ICU to generate various address sequences. During each cycle, the address index output from the ASG will pass through the register group to generate folded indexes, which will be sent to the input queue for reading data horizontally or vertically. Meanwhile, the corresponding PE control signals are generated and fed into the reconfigurable systolic array. In SSD computation, the sparse steps S_{a1}, S_{a3}, and S_{a5} correspond to I/IV, II/V, and III/VI indexing formats, respectively. Address IDs are generated according to the reordering rules to achieve compact tensor feeding. For sparse triangles, this compact stitching can reduce 49.80% of the redundant computations in a tensor of CSize × CSize.

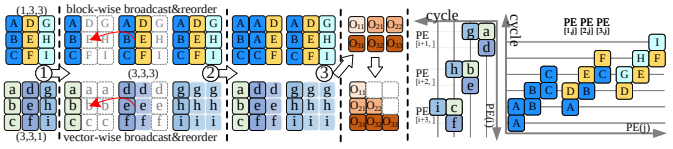


Fig. 6: Workflow of the systolic array for sparse computations.

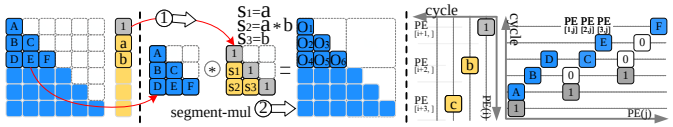


Fig. 7: Workflow of the systolic array for segment-mul primitive.

B. Reconfigurable Systolic Array

Following the computational requirements set by ICU, we design a reconfigurable systolic array that coordinates with the data fetcher to achieve four computational primitives: *GEMM*, *element-wise operations*, *SpMM*, and *segment-mul*. The systolic array consists of 32 × 32 cascaded processing elements (PEs) and incorporates with two EleOps buffers, with the details of PE shown in Fig. 5. Each PE is configured with a multiplexing switch, three registers (REG1∼REG3), four control signals (CTRL1∼ CTRL4), and eight data signals (❶ ∼ ❽).

• **GEMM**: We employ a classical systolic array to support GEMM within the software stack. For linear-based and merged broadcast MACs, feature embeddings and weights are fed into REG2 and REG1 via CUR_IN and W_IN, respectively. In the next cycle, the operands in REG1 and REG2 perform MAC, and outputs are passed through CAS_MAC to the next PE for post-accumulation. For one-dimensional convolutions, input data and weights are loaded onto REG1 and REG2, respectively. Once it completes, CTRL2 disconnects the cascading of adjacent PE rows, transferring the result to external buffers.

• **Element-wise Operations**: To enable computations attributed to EleOps to share the systolic array, we incorporate two dedicated EleOps buffers into the systolic array to support the high bandwidth demands of point-to-point computations. Specifically, the intermediate activations stored in dual buffers can be directly transferred to each cascaded PE for parallel point-granularity operations via the ELE_ADD or ELE_MUL. The results can be written back to buffer along the same routes, overwriting the values at the original locations.

• **SpMM**: To alleviate ineffective computations resulted from zero-padded input sequences and lower triangular causal mask, we employ CTRL2 and CTRL4 to disconnect the cascade between row-wise PEs for zero-skipping. Besides, we introduce additional input channels to prefill more input element c_{ij}, allowing each PE to coordinate with the sparse-predefined data fetcher for SpMM and SDMM. Fig. 6 illustrates the detailed SpMM for tensors $B \in \mathbb{R}^{1\times3\times3}$ and $C \in \mathbb{R}^{3\times3\times1}$. Each PE loads two elements into REG2 via CUR_IN and one element into REG1 via W_IN per cycle. Under MUX control, the current computation unit PE$[i,j]$ fetches a specific element from REG2 to perform a MAC with the loaded element b_{ij}. The resulting partial product is passed vertically to the next PE$[i+1,j]$ in the following cycles, while the input elements pre-stored in REG2 move horizontally to the right-side PE$[i,j+1]$. Each PE alternately performs calculations until all partial accumulations are complete, with the final results output by the last row of PEs.

• **Segment-mul**: To optimize computation resources for $\exp(\cdot)$ and segment-sum$(\cdot)$ operations during causal mask generation, we implement segment-mul (shown in Eq. 5) through the cooperation

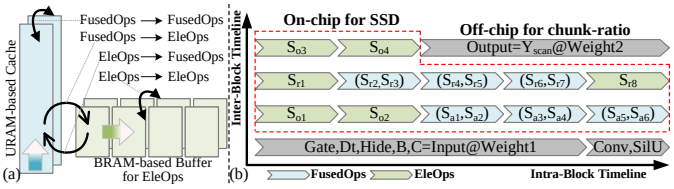


Fig. 8: Memory management and on-chip SSD

of datalines (CAS_MUL and BYPASS) and control signals (CTRL3 and CTRL4). The detailed process of segment-mul is shown in Fig. 7. Specifically, each PE loads an element of the exp-processed causal vector $\{1, \mathbf{a}_i\}$ onto REG2 via CUR_IN and the masked intermediate activations $\{\mathbf{b}\}$ into REG1 via W_IN for element-wise multiplications. The generated partial-product within PE$[i, j]$ is stored back either in REG2 or REG3. The result in REG2 is multiplied by element b_{ij} loaded on W_IN in the next cycle, while the result in REG3 is passed down through CAS_MUL to PE$[i+1, j]$ for cumulative multiplication with element $a_{i,j+1}$. Once each PE completes the computation for a specific position, the result is output to an external register via BYPASS by gating CTRL3 and CTRL4.

C. Always-on-chip SSD Computation

Mamba2 mainly consists of linear transformation and EleOps-centric SSD computations. The linear transformation has high weight reuse and low bandwidth, while the EleOps-centric SSD relies on high bandwidth due to the low data reuse potential. To leverage the on-chip BRAM's access rate of up to 5832 GB/s, we distribute data sequences across multi-cores and place CSize × CSize data blocks on chip for SSD. For linear and convolution operations, we instead use DDR memory with an access rate of 76.8GB/s for data communication. The complete inference process can be shown in Fig. 8. To support coherent on-chip inference, we employ a width-first buffer for EleOps and a depth-first cache for FusedOps operations. Additionally, we achieve a seamless interface between FusedOps and EleOps through memory management optimizations, which includes:

(1) FusedOps → FusedOps: Both input reading and result writing are implemented on the cache.

(2) FusedOps → EleOps: The result of FusedOps is written back to the EleOps buffer for the EleOps stage.

(3) EleOps → EleOps: The EleOps requires elements to be read from the buffer row by row, enabling the systolic array to perform parallel point-to-point multiplication or addition. The result is then written back to the EleOps buffers, overwriting the input operand for subsequent operations.

(4) EleOps → FusedOps: The results generated by EleOps are stored in EleOps buffer and read row by row for FusedOps.

D. Special Function Unit

We integrate other operations of Mamba2 into SFU, including SiLU(·) [37], EXP(·) and RMSNorm(·) [38], etc. For RMSNorm(·), the SFU reads vector-wise data $\{\mathbf{x}\}$ from the cache and processes it through a pipeline of square computation, accumulation, and reciprocal operations. Once the denominator $Var=\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$ is obtained, each element in FIFO [39] will be popped to calculate final normalized results x_i/Var.

V. Evaluation

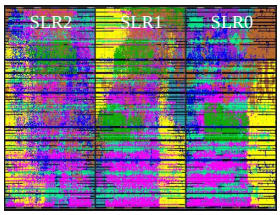
A. Evaluation Methodology

We take five Mamba2 models (shown in Table III) as benchmarks, and generate MambaOPU instructions and data with FP16 [40], [41] using the proposed software optimization stack. The hardware of

TABLE III: Parameter configuration of Mamba2 models

Parameter	130M	370M	780M	1.3B	2.8B
Hidden_size	768	1024	1536	2048	2560
Num_hidden_layers	24	48	48	48	64
Num_heads	24	32	48	64	80
Num_groups	1	2	4	8	8

Note: $head_dim, state_size, chunk_size, conv_kernel_size$ = {64, 128, 256, 4}



Modules	LUT	FF	DSP	BRAM	URAM
PE Array	131K	265K	4096	0	0
EleOps Buffer	2.86K	18.1K	0	1824	0
Cache	0.56K	6.72K	0	0	512
Fetcher	211K	461K	0	0	0
Psum Buffer	72.5K	202K	0	0	120
SFU	325K	418K	256	0	62
Others	130K	145K	12	178.5	84
Total	875K	1519K	4364	2002.5	778

Fig. 9: FPGA layout and resource utilization of MambaOPU-IV

MambaOPU is developed with Verilog, synthesized and implemented on Xilinx U200 FPGA with Vivado. To evaluate effectiveness, we compare the latency, normalized throughput (i.e., throughput per DSP is per GPU/CPU core for processing FP16 operations per cycle), and energy efficiency of MambaOPU with Intel Xeon Gold 6348 CPU, NVIDIA RTX A100 and A800. Power for GPU and CPU is measured with Nvidia-smi and PyRAPL, respectively, while MambaOPU is measured with the Xilinx Board Utility tool [42]. Latency for GPU and CPU is measured by CUDA Events and Python timers, respectively.

B. Experimental Results

We implement the single-core (MambaOPU-I) and four-core (MambaOPU-IV) versions, and the layout and resource utilization of MambaOPU-IV are shown in Fig. 9. On the U200 FPGA, the resource utilization is 74.03% of LUTs, 64.26% of FFs, 63.80% of DSPs, 92.70% of BRAMs, and 81.04% of URAMs. The speedup, normalized throughput and energy efficiency improvements for different sequence lengths are shown in Fig. 10.

V-B1 Speedup. By leveraging sequence parallelism, MambaOPU-IV achieves a 3.67× speedup over MambaOPU-I. This is because the long input sequence can be divided into proportional chunks that are processed in parallel across multiple cores. On the Mamba2-130M model, MambaOPU-IV delivers 1160× and 3.26× over the CPU and A100, respectively. This advantage comes from the operator fusion strategy and reconfigurable systolic array. For the Mamba2-2.8B model, MambaOPU-IV demonstrates a comparable latency to that of the A100. This is because the significant computational demands of linear layer transformations and multi-head SSD for large models or long texts exceed the resource limitations of U200.

V-B2 Throughput. In terms of normalized throughput, MambaOPU-I achieves 263.01×/880.79× improvement over A100 with a sequence length of 64, while MambaOPU-IV gains by 65.75×/220.20× with a sequence length of 8192, respectively. This is because MambaOPU optimizes the SSD computation, achieving comparable inference latency with fewer multiplication resources, leading to higher throughput. Compared to CPU, MambaOPU-I improves throughput by 1104×/1813× when running Mamba2-130M/2.8M with a sequence length of 64, while MambaOPU-IV reaches 794×/1014× with a sequence length of 8192. The results demonstrate that MambaOPU offers superior performance compared to the CPU, through careful optimization of both computation and storage.

V-B3 Energy Efficiency. For end-to-end energy efficiency, MambaOPU-I outperforms MambaOPU-IV in shorter sequence

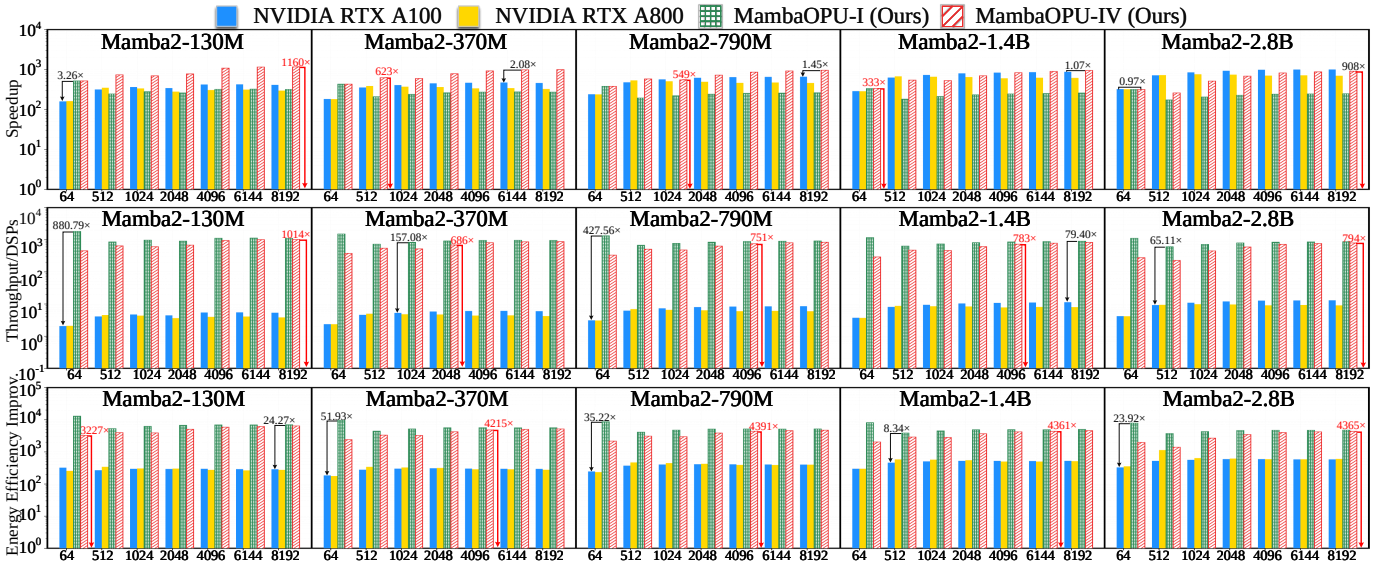


Fig. 10: Latency, throughput per DSP and energy efficiency comparison of MambaOPU and A100/A800 GPUs against Intel CPU (DSP normalized to 276,480 FP16 ops/cycle for A100/A800 and 3,583 FP16 ops/cycle for CPU) [43]. The horizontal axis represents sequence length.

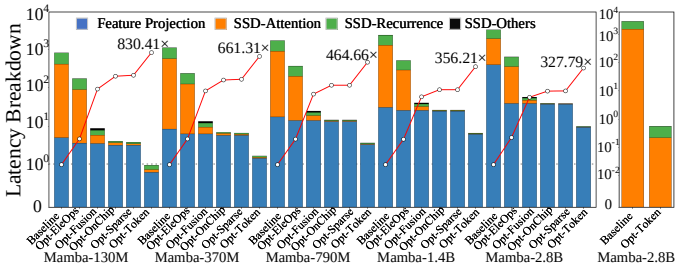


Fig. 11: Latency breakdown of MambaOPU with sequential optimization.

length, primarily due to reduced multi-core resource idling. On specific models such as Mamba2-2.8B and Mamba2-130M, MambaOPU-IV achieves energy efficiency gains of 4365×/6418× over the CPU, while MambaOPU-I achieves 7874×/12908×, respectively. For long text, MambaOPU-I improves energy efficiency by 8.14×/24.27× over A100 and A800, while MambaOPU-IV reaches 7.46×/22.25×. This demonstrates that the FPGA-based MambaOPU achieves excellent energy efficiency in power-constrained applications.

The superior throughput and energy efficiency improvements of MambaOPU come from optimized memory management and on-chip SSD computation that minimize off-chip memory communication. Also, the systolic array supports multiple computational modes, allowing for more efficient resource allocation.

C. Ablation Study

Fig. 11 presents the latency breakdown for five Mmaba2 models on MambaOPU-IV. Each model is accelerated through a sequence of optimization strategies, where each strategy builds on the previous one: element-wise operations(*Opt-EleOps*), operation fusion (*Opt-Fusion*), on-chip SSD computations (*Opt-OnChip*), causal mask sparse computations (*Opt-Sparse*) and sequence sparse computations(*Opt-Token*). Fig. 12 shows the access times and percentages for on-chip and off-chip memory using *Opt-Fusion* and *Opt-OnChip*.

Speedup of Optimization Strategies. The progressive optimization strategies introduce substantial enhancements in Mamba2's computational efficiency. Specifically, the *Opt-Token* accelerates by 327.79×/830.41× across different models compared to the baseline, though the linear layer for feature projection remains a primary

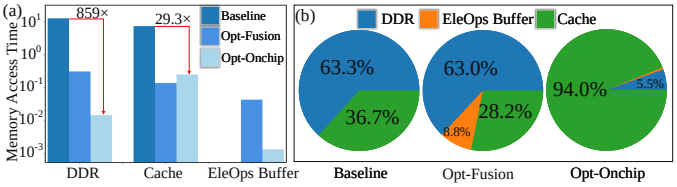


Fig. 12: Memory access time and its percentage for MambaOPU.

bottleneck. Examining individual strategies, *Opt-EleOps* effectively reduces the SSD computations latency by 4.74×, while *Opt-Fusion* further reduces latency for SSD-Attention and SSD-Recurrence by a remarkable 39.28×. These improvements stem from two key factors: first, the reconfigurable systolic array boosts point-to-point multiplication efficiency by introducing additional routes that work in tandem with EleOps buffers; second, operator fusion strategy reduces computation paths, thus enhancing the computation efficiency.

Building on the *Opt-Fusion* advancements, the *Opt-OnChip* strategy enables fully on-chip SSD computations, reducing the latency for attention and recurrence modes by 8× and 5.77×, respectively. Furthermore, *Opt-Sparse* and *Opt-Token* eliminate redundant computations and significantly improve SSD efficiency by a sparsity-predefined data fetcher.

Memory Access. Compared to the baseline, *Opt-OnChip* reduces off-chip memory access by 859×, allowing 94% of data access to be handled within the on-chip cache. This demonstrates that operator fusion and on-chip SSD effectively reduce communication overhead.

VI. Conclusion

We present MambaOPU, an overlay processor to accelerate Mamba2 models. Its compiler applies operator fusion and tensor-reorder-and-group to support efficient data allocation and reduce memory footprint of intermediate states. Its micro-architecture introduces a sparse-predefined data fetcher and a reconfigurable systolic array to optimize computational resource utilization and improve data reuse across various computation modes. Additionally, we design always-on-chip SSD computation. Our MambaOPU demonstrates superior throughput and energy efficiency, achieving 880.79× higher normalized throughput and 12908× greater energy efficiency than A100 GPU, respectively.

REFERENCES

[1] Chen-Hsuan Lin, Jun Gao, Luming Tang, and *et al.* Magic3D: High-resolution Text-to-3D content creation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[2] Suman Bhoi, Mong-Li Lee, Wynne Hsu, and Ngiap Chuan Tan. REFINE: A Fine-Grained medication recommendation system using deep learning and personalized drug interaction modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[3] Ashish Vaswani, Noam M. Shazeer, and *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[4] Alexey Dosovitskiy, Lucas Beyer, and *et al.* An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[5] Alec Radford, Jong Wook Kim, Chris Hallacy, and *et al.* Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021.

[6] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2024.

[7] Tae Jun Ham, Sungjun Jung, and *et al.* A^3: Accelerating attention mechanisms in neural networks with approximation. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2020.

[8] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[9] Thierry Tambe, Coleman Hooper, and *et al.* EdgeBERT: Sentence-level energy optimizations for latency-aware multi-task NLP inference. In *International Symposium on Microarchitecture (MICRO)*, 2021.

[10] Peiyan Dong, Mengshu Sun, Alec Lu, Yanyue Xie, Kenneth Liu, and *et al.* HeatViT: Hardware-efficient adaptive token pruning for vision transformers. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[11] Xuliang Yu, Tianwei Ni, Xinsong Sheng, Yun Pan, Liang Zhao, and Lei He. AESHA: Accelerating eigen-decomposition-based sparse transformer with hybrid RRAM-SRAM architecture. In *International Conference on Computer Aided Design (ICCAD)*, 2024.

[12] Tiandong Zhao, Shaoqiang Lu, Chen Wu, and Lei He. ChatOPU: An FPGA-based overlay processor for large language models with unstructured sparsity. In *International Conference on Computer-Aided Design (ICCAD)*, 2024.

[13] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, and *et al.* FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2024.

[14] Yunxuan Yu, Tiandong Zhao, Mingyu Wang, Kun Wang, and Lei He. Uni-OPU: An FPGA-based uniform accelerator for convolutional and transposed convolutional networks. In *IEEE Tansactions on very large scale integration systems (TVLSI)*, 2020.

[15] Chen Wu, Zhuofu Tao, Kun Wang, and Lei He. SkeletonGCN: A simple yet effective accelerator for gcn training. In *International Conference on Field-Programmable Logic and Applications (FPL)*, 2022.

[16] Yunxuan Yu, Tiandong Zhao, Kun Wang, and Lei He. Light-OPU: An FPGA-based overlay processor for lightweight convolutional neural networks. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2020.

[17] Zheng Qu, Liu Liu, Fengbin Tu, Zhaodong Chen, Yufei Ding, and Yuan Xie. DOTA: detect and omit weak attentions for scalable transformer acceleration. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[18] Jyotikrishna Dass, Shang Wu, Huihong Shi, and *et al.* ViTALiTy: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[19] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, and *et al.* ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2021.

[20] Yubin Qin, Yang Wang, and *et al.* FACT: FFN-Attention co-optimized transformer architecture with eager correlation prediction. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2023.

[21] Tiandong Zhao, Siyuan Miao, Shaoqiang Lu, Jialin Cao, Jun Qiu, Xiao Shi, Kun Wang, and Lei He. Token packing for transformers with variable-length inputs. In *International Conference on Field-Programmable Logic and Applications (FPL)*, 2023.

[22] Teng Wang, Lei Gong, Chao Wang, Yang Yang, Yingxue Gao, Xuehai Zhou, and Huaping Chen. ViA: A novel vision-transformer accelerator based on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2022.

[23] Hanhwi Jang, Joonsung Kim, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. MnnFast: A fast and scalable system architecture for memory-augmented neural networks. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2019.

[24] Yang Bai, Xufeng Yao, Qi Sun, and Bei Yu. AutoGTCO: Graph and tensor co-optimize for image recognition with transformers on GPU. In *International Conference On Computer Aided Design (ICCAD)*, 2021.

[25] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. DFX: A low-latency Multi-FPGA appliance for accelerating transformer-based text generation. In *International Symposium on Microarchitecture (MICRO)*, 2022.

[26] Yunxuan Yu, Chen Wu, Tiandong Zhao, Kun Wang, and Lei He. OPU: An FPGA-based overlay processor for convolutional neural networks. In *IEEE Tansactions on very large scale integration systems (TVLSI)*, 2020.

[27] Chen Wu, Mingyu Wang, Xinyuan Chu, Kun Wang, and Lei He. Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration. In *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2022.

[28] Chen Wu, Jinming Zhuang, Kun Wang, and Lei He. MP-OPU: A mixed precision FPGA-based overlay processor for convolutional neural networks. In *International Conference on Field-Programmable Logic and Applications (FPL)*, 2021.

[29] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. In *Advances in neural information processing systems (NeurIPS)*, 2021.

[30] Jinhao Li, Shan Huang, Jiaming Xu, and *et al.* MARCA: Mamba accelerator with reconfigurable architecture. In *International Conference On Computer Aided Design (ICCAD)*, 2024.

[31] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024.

[32] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. NVIDIA A100 tensor core GPU: performance and innovation. *IEEE Micro, 41, 2 (2021), 29–35.*

[33] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, et al. In-datacenter performance analysis of a tensor processing unit. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2017.

[34] Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob N. Foerster, Satinder Singh, and Feryal M. P. Behbahani. Structured state space models for in-context reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[35] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

[36] Zhang Michael and K Saab Khaled. Effectively modeling time series with simple discrete state spaces. In *International Conference on Learning Representations (ICLR)*, 2023.

[37] Hamza Khan, Asma Khan, Zainab Khan, Lun Bin Huang, Kun Wang, and Lei He. NPE: an FPGA-based overlay processor for natural language processing. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2021.

[38] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[39] Shaoqiang Lu, Tiandong Zhao, Rumin Zhang, Ting-Jung Lin, Chen Wu, and Lei He. An FPGA-based multi-core overlay processor for transformer-based models. In *International Symposium of Electronics Design Automation (ISEDA)*, 2024.

[40] Hung-Yueh Chiang, Chi-Chih Chang, Natalia Frumkin, Kai-Chiang Wu, and Diana Marculescu. Quamba: A post-training quantization recipe for selective state space models. In *arXiv preprint arXiv:2410.13229*, 2024.

[41] John T. Halloran, Manbir Gulati, and Paul F. Roysdon. Mamba state-space models can be strong downstream learners. In *arXiv preprint arXiv:2406.00209*, 2024.

[42] Xilinx. Vitis unified software platform documentation: Application acceleration development (ug1393). 2022.

[43] NVIDIA. Nvidia ampere architecture whitepaper. 2020.

Leveraging Critical Proof Obligations for Efficient IC3 Verification

Lingfeng Zhu[1, 2], Xindi Zhang[1, 2], Yongjian Li[1, 2, *], Shaowei Cai[1, 2]
[1]*Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences*
[2]*School of Computer Science and Technology, University of Chinese Academy of Sciences*
{zhulf, zhangxd, lyj238, caisw}@ios.ac.cn

***Abstract*—IC3 and its variants are SAT-based model-checking methods that play a critical role in hardware verification. Efficient management of proof obligations, which track states that need to be proven unreachable, is essential for improving verification performance. This paper presents a novel approach that utilizes Critical Proof Obligations (CPOs) to improve proof obligation management. We propose two techniques, CPO-Driven UNSAT Core Generation and CPO-Driven Proof Obligation Propagation, to promote lemma propagation and frame refinement. Experimental results on HWMCC benchmarks demonstrate significant improvements in CPO discovery and lemma propagation, resulting in notable performance gains.**

***Index Terms*—model checking, IC3, critical proof obligation, assumption core consistency**

I. Introduction

Safety model checking (MC) automatically determines whether a transition system $S = \langle V, I, T \rangle$ satisfies a specified safety property P. MC is a fundamental software and hardware verification challenge with extensive theoretical and practical applications [9]. IC3 [8] (also known as PDR [10]) is a prominent symbolic MC algorithm that leverages high-performance incremental SAT solvers and sophisticated induction-based interpolation sequence refinement techniques. In recent years, IC3 and its variants [11, 15, 18] have gained significant attention due to their completeness and scalability compared to Bounded Model Checking (BMC) and Interpolation-based Model Checking (IMC).

IC3 maintains an over-approximation sequence known as *frames*, incrementally constructing and refining them to either generate a proof for the specified safety property or provide a counterexample. This refinement is achieved by recursively blocking bad cubes, which are counterexamples to induction (CTIs) capable of reaching the negation of the safety property. During this process, proof obligations (POs) are generated to track states that need to be proven unreachable. These POs are then generalized to produce lemmas, which are added to the frames and propagated to subsequent frames, reducing redundancy and accelerating convergence.

Lemma propagation plays a crucial role in the efficiency of IC3, as highlighted by several studies. [13] explore blocking states farther from the bad states to discover stronger inductive lemmas, enabling deeper generalization and enhancing lemma propagation in IC3. [22] directs induction generalization towards propagating i-good lemmas to accelerate convergence. [21] predicts a possible minimal lemma before dropping variables using the counterexample to propagation. These studies underscore the importance of effective lemma propagation in optimizing IC3 performance.

Proof obligation management, in turn, facilitates improved lemma propagation and contributes to the overall efficiency of IC3. [12] treats the negation of a discovered lemma as a may-proof-obligation to attempt possible lemma propagation and uses the reachable states that witness a failure of a may-proof-obligation to improve future lemma generalization. [20] presented a comprehensive study on proof obligation generalization techniques in IC3, including lifting, 01X-simulation, and justification-based generalization, highlighting the significant impact of different PO generalization methods on experimental performance.

Motivated by these insights, we introduce Critical Proof Obligations (CPOs), a novel concept aimed at improving the effectiveness of IC3 by prioritizing the handling of proof obligations that significantly influence the lemma propagation process. By identifying and focusing on these critical proof obligations, we streamline the verification process, boost lemma propagation, and improve the algorithm's convergence speed.

Contributions: This paper introduces the concept of CPO, which prioritizes highly impactful proof obligations to improve lemma propagation and frame refinement and presents two CPO-driven methods designed to enhance the IC3/PDR algorithm. The first method, CPO-Driven UNSAT Core Generation, optimizes UNSAT core generation by leveraging CPOs to prioritize relevant literals in SAT queries, effectively identifying CPOs that drive lemma propagation. The second method, CPO-Driven Proof Obligation Propagation, selectively propagates proof obligations based on their criticality, minimizing unnecessary lemma accumulation and promoting efficient verification.

We implemented our two novel CPO-driven methods in two well-performing IC3 safety model checkers: IC3ref [8] and modelchecker [3]. Additionally, we conducted experiments using HWMCC benchmarks to compare our solvers against three other state-of-the-art IC3/PDR algorithms. A comprehensive evaluation of the proposed methods demonstrates that identi-

*Corresponding author

fying CPOs boosts lemma propagation and proof efficiency, leading to substantial performance gains and validating the effectiveness of our approach.

Structure: Section II provides the necessary background information. Section III introduces the concept of CPO and two CPO-driven methods for optimizing proof obligation management. We evaluate the proposed methods in Section IV, discuss related work in Section V, and conclude the paper in Section VI.

II. Preliminaries

This paper focuses on bit-level symbolic model-checking methods. The following subsections introduce fundamental concepts relevant to propositional logic, finite-state transition systems, safety checking, SAT solving, and the IC3 algorithm.

A. Propositional Logic

Given a set of Boolean variables $V = \{v_1, \ldots, v_n\}$, a *literal* of a variable v is either the variable itself (v) or its negation ($\neg v$). We use $var(l)$ to denote the variable corresponding to a literal l.

A *clause* (resp. *cube*) is defined as the disjunction (resp. conjunction) of a set of literals. The negation of a cube forms a clause (also referred to as a *lemma*), and vice versa. If all literals of clause (resp. cube) d are included in clause (resp. cube) c, then d is called a sub-clause (resp. sub-cube) of c, denoted by $d \subseteq c$. The *conjunctive normal form* (CNF) represents a formula as a conjunction of clauses.

A full (resp. partial) *assignment* $\alpha : V \to \{false, true\}$ maps all (resp. some) variables to Boolean values and can be represented as a cube. We use $\alpha \models F$ (resp. $\alpha \not\models F$) to indicate that an assignment α satisfies (resp. falsifies) a formula F, making α a *model* of F. Additionally, $F \Rightarrow G$ implies that if F holds, G must also hold, meaning all models of F are also models of G.

B. Finite-state Transition System and Safety Checking

A *finite-state transition system* (FTS) is typically represented as a triple $S = \langle V, I, T \rangle$, where V is a set of Boolean variables, $I(V)$ is the initial state, and $T(V, V')$ is the transition relation describing the state transitions within the system. A state s of an FTS S is a cube composed of literals over V. We use the primed version s' to denote the next state of s according to $T(V, V')$. A state s is called an F-state if $s \models F$.

A *trace* of a finite-state transition system (FTS) S is a sequence of states $\gamma = s_0, s_1, s_2, \ldots$, which can be either finite or infinite. The initial state s_0 must satisfy the initial condition I, and for each state pair s_i, s_{i+1} in the trace γ, the transition relation $s_i, s'_{i+1} \models T$ must hold. A state s is said to be *reachable* if it appears in any trace of the system S.

In formal verification, safety properties of transition systems are commonly verified through *safety model checking*. A safety property $P(V)$ asserts that only those states satisfying P (referred to as P-states) are reachable. The property P is considered an *invariant* for the system S if all reachable states are P-states. If P is not invariant, there exists a finite counterexample trace $s_0, s_1, s_2, \ldots, s_k$ with $s_k \models \neg P$, where s_k is referred to as a *bad state*.

A lemma c is called *inductive* if it satisfies the following two conditions:

- *Initiation*: c holds for all initial states, i.e., $I \Rightarrow c$.
- *Consecution*: c is closed under the transition relation, i.e., $c \wedge T \Rightarrow c'$.

A lemma c is said to be *inductive relative* to F_i if $I \Rightarrow c$ and also ensures that $F_i \wedge c \wedge T \Rightarrow c'$. This can be verified by checking whether the formula $F_i \wedge c \wedge T \wedge \neg c'$ is unsatisfiable. An *inductive strengthening* of a safety property P is a formula F such that the conjunction $F \wedge P$ is inductive.

C. SAT Solving and Unsatisfiable Core

The propositional satisfiability (SAT) problem is a decision problem that determines whether a given propositional formula is satisfiable. SAT solvers are algorithms designed to solve this problem, typically taking inputs in *conjunctive normal form* (CNF).

Incremental SAT solvers are a specialized class of SAT solvers that can efficiently solve a sequence of related CNF formulas. One key feature of incremental SAT solvers is their ability to solve a formula while considering a set of *assumption* literals, which are frozen to true during the search process.

Given a CNF formula F and a set of assumptions A, if F is satisfiable, the solver returns "SAT" along with a satisfying assignment. If F is unsatisfiable, the solver returns "UNSAT" and a subset A_{core} of A, known as the UNSAT core, representing the reason for the formula's unsatisfiability.

For clarity in describing the following algorithms, we assume that the SAT solver used in this paper provides the following APIs for incremental SAT solving [16]:

- **is_SAT**(F, A): Checks the satisfiability of a propositional formula F under the assumption A. This can be expressed as is_SAT($\overline{F}$), where $\overline{F} = F \wedge A$.
- **get_model**(): If is_SAT(F, A) returns "SAT", this API returns a model for F under the assumption A. We use val(v) to denote the value of the variable v.
- **get_core**(). If is_SAT(F, A) returns "UNSAT", this function returns the UNSAT core A_{core} for F under the assumption A.

D. IC3 Algorithm Overview

The IC3 algorithm, also known as PDR, is an efficient model-checking technique designed to verify safety properties in finite-state transition systems. A high-level overview of IC3/PDR is presented in Alg. 1.

IC3 incrementally constructs an inductive proof of correctness using a sequence of frames, denoted as $F_0, F_1, F_2, \ldots$, where each frame F_i over-approximates the states reachable within i steps from the initial state. The algorithm starts with $F_0 = I$, representing the initial state set. It builds subsequent frames such that $F_i \Rightarrow F_{i+1}$ and $F_i \wedge T \Rightarrow F_{i+1}$, progressively strengthening them to ensure no bad states are reachable. The goal is to eventually obtain a frame F_k that is safe and

inductive, i.e., it contains no bad states and is closed under the transition relation, thereby proving the system's safety.

IC3 iteratively selects a cube s in F_k such that it transitions to a bad state in one step. Such an s is identified as a counterexample to induction (CTI), which starts from F_k and transitions to a bad state. IC3 attempts to block CTIs by maintaining a proof obligation (PO) queue and adding new lemmas to the frames. Each PO is a pair (s, i), where s is a CTI and i is a frame number, indicating that s must be proven inductively relative to F_i; otherwise, the property fails. If s is inductively relative to F_i, meaning it is unreachable within $i+1$ steps from the initial state, we say (s, i) is blocked.

The blocking process may involve the recursive generation of proof obligations, which are propagated backward through frames until reaching a safe resolution or the initial state. Once blocking ensures that F_k cannot reach a bad state, the propagation phase begins, pushing a lemma c in $F_i \setminus F_{i+1}$ to F_{i+1} if it is inductive relative to F_i. When two consecutive frames become identical ($F_i = F_{i+1}$), an inductive invariant is established, and the model's safety is proved.

III. Leveraging Critical Proof Obligations

In the IC3 algorithm, managing and effectively processing proof obligations (POs) is crucial for ensuring the system's safety, as it enables iterative refinement of frames to exclude bad states. In this section, we present our methodology for enhancing the IC3/PDR algorithm by evaluating and prioritizing proof obligations to promote effective lemma propagation during frame refinement. This will thereby accelerate convergence towards proving system safety.

A. Intuition behind Critical Proof Obligations

As discussed in the preliminaries, a proof obligation (PO) is defined as a pair (s, i), where s represents a counterexample to induction (CTI) leading to a bad state and should be proved inductive relative to F_i. In the IC3/PDR framework, blocking all possible states that may lead to a bad state within a frame is impractical due to the prohibitive computational cost. Instead, the primary objective is to ensure that each blocked counterexample contributes a useful lemma that facilitates lemma propagation and the construction of system invariants. Specifically, for a lemma $\neg c$ in $F_i \setminus F_{i+1}$, and a CTI s such that $c \subseteq s$, the successful processing of the PO (s, i)—i.e., showing that s is inductive relative to frame F_i and adding the lemma $\neg s$ to frame F_{i+1}—is crucial for propagating the lemma $\neg c$ from frame F_i to frame F_{i+1}. This motivates the formal definition of a critical proof obligation (CPO).

Definition 1 (Critical Proof Obligation)**.** *Let (s, i) be a proof obligation, where s represents a counterexample to induction (CTI) leading to a bad state, and i is a frame index. We define (s, i) as a Critical Proof Obligation if at least one lemma $\neg c$ exists in $F_i \setminus F_{i+1}$ such that $c \subseteq s$.*

Theorem 1. *Let $\neg c$ be a lemma in $F_i \setminus F_{i+1}$. The lemma $\neg c$ can be propagated to F_{i+1} if only all critical proof obligations (s, i) such that $c \subseteq s$ are blocked.*

Algorithm 1: Overview of IC3 and our methods

```
/* Codes related to CPO-Driven UNSAT Core
   Generation in Sect. III-B (resp. CPO-Driven
   Proof Obligation Propagation in Sect. III-C)
   is colored red (resp. blue).                  */
1  function inductive_relative (cube c, level l) :
2      return not is_SAT(F_l ∧ ¬c ∧ T, c');
3
4  function generalize (cube c, level l) :
5      for each p ∈ c do
6          if I ⇒ ¬c\{p} and inductive_relative(c\{p}, l) then
7              c := c\{p};
8      return c;
9
10 function get_predecessor (cube s, level l) :
11     input, t ← {i ∈ get model() | var(i) ∈ V};
12     assert(not is_SAT(input ∧ T ∧ ¬s', reverse_sort(t)));
13     pre := get_core();
14     for each ¬c ∈ F_{l-1} \ F_l do
15         if c ⊆ pre then
16             reward_core(pre);
17             Break;
18     return pre;
19
20 function block_obligation () :
21     while not obls.empty() do
22         ⟨c, l⟩ ← obls.pop() ;
23         if inductive_relative(c, l) then
24             if I∧ get_core() then s := generalize(c, l);
25             else s := generalize(get_core(), l);
26             while l ≤ k and inductive_relative(s, l) do
27                 l := l + 1;
28             for each ¬t ∈ F_l \ F_{l+1} do
29                 if t ⊆ c and l ≤ k then
30                     obls.insert(⟨c, l⟩);
31                     Break;
32             for i ∈ [1, l] do F_i := F_i ∪ {¬s} ;
33         else if l = 0 then return false ;
34         else obls.insert(⟨get_predecessor(c, l), l − 1⟩) ;
35     return true;
36
37 function propagate (level l) :
38     for i ∈ [1, l) do
39         for ¬c ∈ F_i \ F_{i+1} do
40             if inductive_relative(c, i) then
41                 F_{i+1} := F_{i+1} ∪ {¬c};
42         if F_i = F_{i+1} then return true ;
43     return false;
44
45 procedure ic3(I, T, P):
46     if is_SAT(I ∧ ¬P) then return unsafe ;
47     F_0 ← I, k ← 1, F_k ← ⊤;
48     while true do
49         while is_SAT(F_k ∧ ¬P) do
50             c := get_predecessor(get_model(), k);
51             obls.insert(⟨c, k − 1⟩);
52             if not block_obligation() then return "unsafe" ;
53         k + 1, F_k ← ⊤;
54         if propagate(k) then return "safe" ;
```

Proof. Assume a proof obligation (s, i) exists such that $c \subseteq s$ is not blocked. Then, there exists a state t in F_i such that $t \wedge T \Rightarrow s'$. Since $\neg c$ is present in F_i, we have $t \wedge \neg c \wedge T \Rightarrow t \wedge T \Rightarrow s' \Rightarrow c'$. Therefore, there exists a state t in F_i such that $t \wedge \neg c \wedge T \Rightarrow c'$, implying that the lemma $\neg c$ is not inductive relative to F_i, and thus cannot be propagated to F_{i+1}.

This contradicts the original claim that $\neg c$ can be propagated to F_{i+1}. Therefore, our assumption is false, and all critical proof obligations (s, i) such that $c \subseteq s$ must be blocked for $\neg c$ to propagate. □

Critical Proof Obligations are crucial because they directly impact the propagation of lemmas across frames, which is fundamental to the success of the IC3 algorithm. Our approach aims to identify and leverage CPOs and prioritize their impact to effectively propagate lemmas, refine frames, and contribute to verification. The following subsections introduce two methods for leveraging and managing Critical Proof Obligations for optimal impact.

B. CPO-Driven UNSAT Core Generation

The branching strategy is a crucial feature of modern Conflict-Driven Clause Learning (CDCL) [16] SAT solvers. Traditional branching heuristics, like Variable State Independent Decaying Sum (VSIDS) [17], play an important role in prioritizing decision variables based on their involvement in conflicts, ensuring that frequently involved variables are selected to drive the solver towards resolution.

In our approach, we adopt a similar idea to influence the generation of UNSAT cores by ordering the assumption literals in SAT queries according to their scores (sort from high to small, see line 12 of Alg. 1). This score reflects how often each literal appears in recent Critical Proof Obligations. The key observation is that modern SAT solvers apply the assumption literals in the order provided by the user. Consequently, the UNSAT core produced when the formula is unsatisfiable depends heavily on this order, with literals that appear earlier being more likely to be included in the core.

Our goal is to ensure that each returned UNSAT core leads to the generation of a Critical Proof Obligation. To achieve this, we keep track of the UNSAT cores that successfully produce CPOs and heuristically adjust the input assumption order, aligning the variables in the assumption list with those frequently observed in successful CPO-generating cores. To facilitate this process, we introduce a metric called *Assumption Core Consistency* (ACC) to evaluate the consistency of variable occurrence in CPOs.

Definition 2 (Assumption Core Consistency)**.** *Let A be the set of assumptions, and let $O_A(a)$ denote the index of $var(a)$ in the default assumption ordering. Let $C_A = \{a_1, \ldots, a_n\} \subset A$ represent the UNSAT core obtained during the CTI search, and let $i \in \{1, \ldots, n\}$ denote the index of variable $var(a_i)$ in C_A. The rules governing UNSAT core generation hold that $i \leq O_A(var(a_i))$ for all variables in the core.*

The ACC score for a variable $v \in \{var(l) \mid l \in C_A\}$ is defined as the difference between its index in the UNSAT core and its original position in the assumption list:

$$acc(var(a_i)) = O_A(var(a_i)) - i$$

A higher ACC score indicates greater inconsistency in the ordering, suggesting a need for adjustment.

ACC serves as a basis for refining the assumption ordering to improve UNSAT core generation. The refinement process is outlined below (function *get_predecessor* of Alg. 1):

ACC-based Generalization Refinement: After identifying a predecessor state t of the CTI s in frame F_l, the SAT solver produces an UNSAT core *pre*, which serves as a generalized version of t and forms a new proof obligation $(pre, l-1)$. For each variable $v \in \{var(l) \mid l \in pre\}$, we calculate its ACC score $acc(v)$ and adjust its sorting score $sc(v)$ only if $(pre, l-1)$ is identified as a Critical Proof Obligation. This ensures that variable scores are updated exclusively when the associated proof obligation is crucial for lemma propagation across frames.

The score is updated as follows (line 16 of Alg. 1):

$$sc(v) := sc(v) + acc(v)$$

This adjustment improves UNSAT core generation by compacting the trail of assumptions to effectively refute the given assumptions and leveraging CPO historical data to identify better UNSAT cores and enhance lemma propagation.

C. CPO-Driven Proof Obligation Propagation

In the IC3/PDR algorithm, proof obligation propagation is critical in frame refinement by generating lemmas that block bad states. Traditionally, all proof obligations are propagated to the latest frame, which increases the chance of finding counterexample paths further from the current frame. However, several studies ([5, 12]) have indicated that unguided lemma learning can accumulate redundant lemmas. Similarly, indiscriminate proof obligation propagation can result in excessive lemmas generated from blocking distant counterexamples, which are unreachable in the current frame, making it more challenging to propagate all lemmas within the same frame to prove system safety.

Therefore, our approach leverages Critical Proof Obligations for a more efficient trade-off. Instead of propagating all proof obligations, we selectively push forward CPOs that are directly linked to learned lemmas in the corresponding frames. This targeted approach ensures that only the most impactful proof obligations contribute to lemma propagation, reducing the production of unnecessary lemmas while still facilitating the discovery of critical counterexample paths.

The implementation of CPO-Driven Proof Obligation Propagation (function *block_obligation* in Alg. 1) is straightforward, as it builds upon the existing IC3 framework with minimal changes to the core algorithm. When a proof obligation (s, i) is blocked, we first check for any lemma $\neg c$ in frame F_i such that $c \subseteq s$. If such a lemma exists, we identify (s, i) as

TABLE I: Tested tools and configurations in the experiments.

Tools	IC3ref	MCer	ABC	CAV23	DAC24
Available Flags	-uc -pr	-uc -pr	-pdr	-	-

a CPO, generalize s to F_l, and push the CPO to frame F_{l+1} (see line 23–31 of Alg. 1). If no such lemma is found, the PO is generalized and discarded. By selectively propagating CPOs, our method enhances lemma propagation efficiency and accelerates convergence, as shown in Section IV-C.

IV. Evaluation

A. Experimental Setup

We implemented our two proposed methods—CPO-Driven UNSAT Core Generation and CPO-Driven Proof Obligation Propagation—on two top-performing baseline solvers, using the flags "-uc" and "-pr" to enable each method, respectively:

- **IC3ref** [8]. Developed by the original inventor of IC3, this solver is competitive and commonly used as a baseline in recent studies [19, 21, 22].
- **modelchecker** [3]. This tool implements IC3 using the advanced incremental SAT solver CaDiCaL [7] and is referred to as MCer for brevity.

Additionally, we selected three state-of-the-art open-source IC3/PDR algorithms for comparison:

- **ABC** [1]. This tool includes the well-known implementation of the PDR algorithm.
- **IC3ref-CAV23** [22]. This version enhances IC3Ref by controlling lemma generalization to discover i-good lemmas and is referred to as "CAV23" for brevity.
- **IC3ref-DAC24** [21]. This version applies a generalization technique based on lemma prediction within IC3ref and is referred to as "DAC24" for brevity.

Table I lists the available configurations for the tools mentioned above.

We conducted our experiments using the AIGER format benchmarks from the Hardware Model Checking Competitions (HWMCC) of 2019, 2020, and 2024, totaling 786 instances. All benchmarks were converted to AIGER version 1.8 [6] using the "aigmove" command from the AIGER tool suite [2] to ensure compatibility with open-source model-checking tools, such as ABC, which have limitations in handling invariant constraints.

Since all solvers discussed in this paper are deterministic, each instance was run once on a dedicated CPU core, with a wall time limit of 5000 seconds and an 8 GB memory limit. Experiments were conducted on a cluster with two AMD EPYC 7763 CPUs (128 cores, 1024 GB RAM) running CentOS 7.7.1908. All solvers were implemented in C/C++ and compiled with GNU gcc/g++ 9.4.0. To ensure reproducibility, the source code for our optimized solvers is available at https://github.com/IC3Contributor/DAC25.

TABLE II: Performance Summary among different configurations. Versions with our proposed optimizations are marked with "-uc", "-pr", or both.

name	#solved	#safe	#unsafe	#gain	#lost
IC3ref	463	388	75	-	-
IC3ref -uc	478	397	81	19(11/8)	4(2/2)
IC3ref -pr	471	392	79	11(6/5)	3(2/1)
IC3ref -uc -pr	**481**	397	**84**	20(11/9)	2(2/0)
MCer	455	380	75	-	-
MCer -uc	461	388	73	18(14/4)	12(6/6)
MCer -pr	461	383	78	11(5/6)	5(2/3)
MCer -uc -pr	470	394	76	25(19/6)	10(5/5)
ABC -pdr	465	**398**	67	-	-
CAV23	476	392	**84**	-	-
DAC24	466	390	76	-	-

B. Experimental Results

The experimental results are summarized in Table II, Fig. 1, and Fig. 2, which present the overall performance, configuration comparisons, and runtime scatter plots of implementations with and without our proposed optimizations, respectively. By incorporating our optimizations, different implementations solved more cases than the original versions across various time limits, achieving higher efficiency in solving a more significant number of instances. Notably, IC3ref-uc-pr and MCer-uc-pr solved 18 and 15 additional instances, respectively, demonstrating significant gains.

Compared to three state-of-the-art IC3/PDR algorithms, our best configuration, IC3ref-uc-pr, solved 16, 5, and 15 more instances than ABC, CAV23, and DAC24, respectively. Specifically, for unsafe instances, IC3ref-uc performed on par with CAV23, solving most cases, while for safe instances, IC3ref-uc-pr achieved similar performance to the best competitors (ABC), with only one instance difference. IC3ref-uc-pr showed notable improvements in verifying both safe and unsafe instances, achieving better results with fewer losses compared to the baseline and competing solvers.

In conclusion, the performance summary table, Cumulative Distribution Function (CDF), and scatter plots consistently demonstrated runtime improvements with our methods. Both IC3ref-uc-pr and MCer-uc-pr achieved significant speedups across multiple instances, underscoring the effectiveness of our optimizations in enhancing IC3's performance and efficiency.

C. Further Analysis

To demonstrate the effectiveness of our methods, we computed two metrics for each instance: the CPO Identification Rate ($SR_{cpo} = N_{cpo}/N_{cti}$) and the Lemma Propagation Success Rate ($SR_{pr} = N_{sp}/N_{pr}$). Here, N_{cpo} is the number of CTIs identified as CPOs, N_{cti} is the total number of generated CTIs, N_{sp} is the number of successful lemma propagation attempts, and N_{pr} is the total number of lemma propagation attempts. The average SR_{cpo} for MCer, MCer-

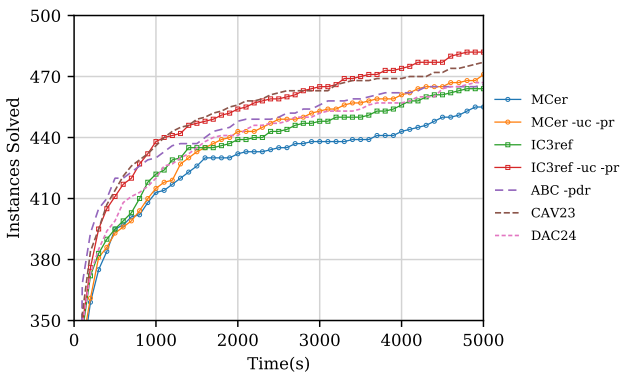


Fig. 1: Cumulative distribution function (CDF) comparing the performance of our improved solvers with state-of-the-art solvers (for clarity, results for single heuristic configurations are omitted).

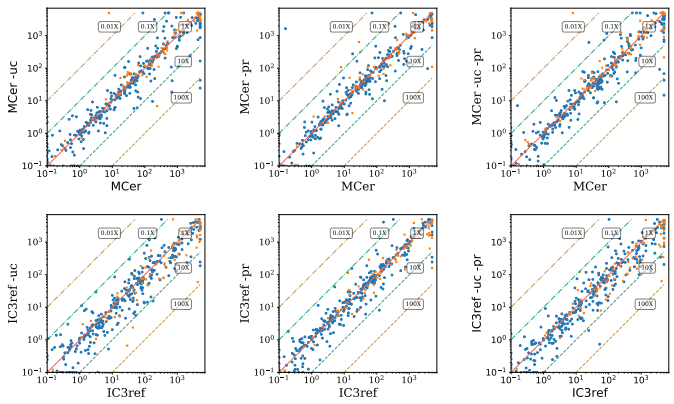


Fig. 2: Scatter plots comparing runtimes of MCer/IC3ref and their optimized versions. Points below the diagonal indicate better performance, with orange for unsafe instances and blue for safe instances.

uc-pr, IC3ref, and IC3ref-uc-pr were 0.4962, 0.5378, 0.2657, and 0.3152, respectively, indicating that our optimizations consistently improved CPO identification. Likewise, the average SR_{pr} for these solvers were 0.7052, 0.7259, 0.6585, and 0.7056, showing enhanced lemma propagation efficiency and accelerated convergence.

Fig. 3 visualizes the impact of our methods through a scatter plot, comparing the deviation in SR_{cpo} (y-axis) with the deviation in verification time (x-axis) for MCer/IC3ref with and without our optimizations. For each instance, let t and t' represent the verification times for MCer/IC3ref and MCer-uc-pr/IC3ref-uc-pr, respectively, and sr and sr', denote their corresponding CPO Identification Rates. Each point on the plot is defined by $t - t'$ on the x-axis and $sr' - sr$ on the y-axis. More points in the first quadrant provide more substantial evidence for the effectiveness of our method, as they indicate higher CPO identification rates and shorter verification times. The results in Fig. 3 for both MCer and IC3ref clearly demonstrate that identifying CPOs contributes to improved model-checking performance by reducing verification time.

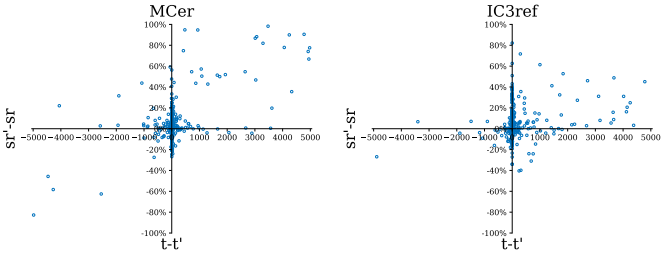


Fig. 3: Scatter plot illustrating the impact of our optimizations on verification time and CPO Identification Rate.

V. Related Works

SAT-based model-checking algorithms are fundamental in hardware verification, driving the development of various IC3 [8] and PDR [10] variants to improve efficiency and applicability. For example, [18] proposed a forward/backward hybrid version of PDR, and [4] introduced IC3 integrated with a specialized SAT solver called "SAT Modulo SAT". Additionally, kAvy [14] combined IC3 with k-induction to enhance its capabilities.

Lemma propagation involves propagating learned lemmas through different frames to block invalid states and accelerate convergence towards proving safety. [13] introduced counterexamples to generalization to enhance blocking bad states and lemma propagation; [21] refined failed pushed parent lemmas as predictions to avoid dropping variables. Our approach differs from these works by introducing Critical Proof Obligations (CPOs) to optimize proof obligation management, thereby enhancing lemma propagation and frame refinement.

Another related work is QUIP [12], which treats the negation of a discovered lemma c as a may-proof-obligation, attempting to push it to the next frame to propagate more lemmas. While both QUIP and our method focus on proof obligation management and selective lemma propagation, our approach evaluates and prioritizes existing proof obligations that directly impact lemma propagation across frames rather than introducing new ones.

VI. Conclusions and Future Works

This work enhances the IC3 algorithm's efficiency through Critical Proof Obligations and two CPO-driven methods that optimize proof obligation handling and lemma propagation. Extensive experiments demonstrated that our methods significantly improved the identification of impactful proof obligations and increased the success rate of lemma propagation, highlighting their effectiveness in enhancing overall performance. In the future, we plan to design IC3 algorithms that tightly cooperate with SAT solvers, extend our techniques to word-level IC3 tools, and adapt CPO-driven methods for broader application in other model-checking algorithms.

Acknowledgment

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300.

References

[1] ABC. https://github.com/berkeley-abc/abc

[2] AIGER Tools. https://fmv.jku.at/aiger/aiger-1.9.9.tar.gz

[3] Modelchecker. https://github.com/ISCAS-modelchecker/modelchecker

[4] Bayless, S., Val, C.G., Ball, T., Hoos, H.H., Hu, A.J.: Efficient modular SAT solving for IC3. In: FMCAD. pp. 149–156. IEEE (2013)

[5] Berryhill, R., Veira, N., Veneris, A., Poulos, Z.: Learning lemma support graphs in quip and ic3. In: 2017 IEEE 2nd International Verification and Security Workshop (IVSW). pp. 105–110 (2017). https://doi.org/10.1109/IVSW.2017.8031554

[6] Biere, A.: The AIGER And-Inverter Graph (AIG) format version 20071012. Tech. Rep. 07/1, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria (2007)

[7] Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)

[8] Bradley, A.R.: Sat-based model checking without unrolling. In: VMCAI. Lecture Notes in Computer Science, vol. 6538, pp. 70–87. Springer (2011)

[9] Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer (2018)

[10] Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: FMCAD. pp. 125–134. FMCAD Inc. (2011)

[11] Griggio, A., Roveri, M.: Comparing different variants of the ic3 algorithm for hardware model checking. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**(6), 1026–1039 (2016)

[12] Gurfinkel, A., Ivrii, A.: Pushing to the top. In: FMCAD. pp. 65–72. IEEE (2015)

[13] Hassan, Z., Bradley, A.R., Somenzi, F.: Better generalization in IC3. In: FMCAD. pp. 157–164. IEEE (2013)

[14] Krishnan, H.G.V., Vizel, Y., Ganesh, V., Gurfinkel, A.: Interpolating strong induction. In: CAV (2). Lecture Notes in Computer Science, vol. 11562, pp. 367–385. Springer (2019)

[15] Li, J., Dureja, R., Pu, G., Rozier, K.Y., Vardi, M.Y.: Simplecar: An efficient bug-finding tool based on approximate reachability. pp. 37–44. Springer (2018)

[16] Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 133–182. IOS Press (2021)

[17] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th annual Design Automation Conference. pp. 530–535 (2001)

[18] Seufert, T., Scholl, C.: Combining PDR and reverse PDR for hardware model checking. In: DATE. pp. 49–54. IEEE (2018)

[19] Seufert, T., Scholl, C., Chandrasekharan, A., Reimer, S., Welp, T.: Making PROGRESS in property directed reachability. In: MBMV. pp. 1–2. VDE/IEEE (2022)

[20] Seufert, T., Winterer, F., Scholl, C., Scheibler, K., Paxian, T., Becker, B.: Everything you always wanted to know about generalization of proof obligations in pdr. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **42**(4), 1351–1364 (2023). https://doi.org/10.1109/TCAD.2022.3198260

[21] Su, Y., Yang, Q., Ci, Y.: Predicting lemmas in generalization of ic3. In: Proceedings of the 61st ACM/IEEE Design Automation Conference. pp. 1–6 (2024)

[22] Xia, Y., Becchi, A., Cimatti, A., Griggio, A., Li, J., Pu, G.: Searching for i-good lemmas to accelerate safety model checking. In: CAV (2). Lecture Notes in Computer Science, vol. 13965, pp. 288–308. Springer (2023)

DDRoute: a Novel Depth-Driven Approach to the Qubit Routing Problem

Alessandro Annechini
Politecnico di Milano
Milan, ITA
alessandro.annechini@mail.polimi.it

Marco Venere
Politecnico di Milano
Milan, ITA
marco.venere@polimi.it

Donatella Sciuto
Politecnico di Milano
Milan, ITA
donatella.sciuto@polimi.it

Marco D. Santambrogio
Politecnico di Milano
Milan, ITA
marco.santambrogio@polimi.it

***Abstract*—In the Noisy Intermediate-Scale Quantum (NISQ) era, the topological constraints present in many of the currently available quantum devices pose a physical limit on the feasible interactions between qubits. To comply with such limitations, the compilation of quantum circuits requires solving the Qubit Routing Problem (QRP), by inserting `SWAP` operations among qubits. The State of the Art provides heuristic algorithms addressing this task, yet the depth of the output circuits is often incompatible with the current limits of quantum hardware. Therefore, we propose DDRoute, a novel heuristic algorithm to solve QRP, designed to reduce the depth overhead introduced by the routing process in the compiled circuits. Our experimental evaluation proves the efficiency of our approach, with a depth reduction of up to 70% with respect to the state-of-the-art routing procedures.**

***Index Terms*—Quantum Computing, Qubit Mapping, Qubit Routing, Quantum Circuit, Depth Minimization.**

I. Introduction

Quantum computing has raised particular interest in the last decades due to its potential in different relevant applications. Quantum algorithms are able to solve important problems with remarkable speedups over their classical counterparts, such as integer factorization [1] and unstructured database search [2]. The availability of quantum computing devices has by now become a reality, with IBM [3], Rigetti [4], and Google [5], among others, releasing powerful quantum processors. Current devices offer hundreds of quantum bits (qubits), but their computational capabilities are limited due to the connectivity constraints between qubits in the Quantum Processing Unit (QPU) [6]. Moreover, further technological issues lie between the development of quantum software and its implementation on quantum hardware, such as the limited fidelity of the operations and the restricted coherence time of each qubit, which introduce errors in the computation and limit the maximum allowed execution time [7].

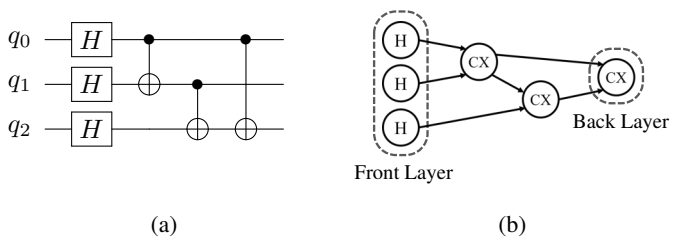
To ease development, current software toolchains allow engineers to represent quantum algorithms as abstract circuits, based on a high-level instruction set (referred to as *gates* of the circuit) and *logical* qubits with unlimited connectivity. This abstraction layer does not consider hardware limitations, and relies on compilation to produce hardware-compliant quantum assembly. Such a process resembles classical Electronic Design Automation, since it requires Place & Route algorithms to *map* high-level logical qubits to the *physical* qubits belonging to the machine, and to *route* such qubits, to overcome their connectivity issues and implement multi-qubit operations.

More specifically, the Qubit Routing Problem (QRP) considers the connectivity constraints among qubits. To execute a two-qubit operation between two unconnected physical qubits, it is necessary to modify their mapping, by inserting `SWAP` (or equivalent) gates along the circuit. Such a process may increment both the number of two-qubit operations and the depth of the final circuit, thus limiting reliability and support from current NISQ architectures. In particular, the insertion of `SWAP` operations decreases the fidelity of the circuit, due to the high error rates of physical implementations of gates. Furthermore, the increase in depth due to the addition of `SWAP`s may also produce circuits that are incompatible with the coherence time of the physical qubits. Algorithms addressing QRP should, therefore, minimize both the number of `SWAP` operations and the overall circuit depth. Unfortunately, these two objectives represent NP-complete tasks [8], [9], and the State of the Art provides heuristics [10]–[14], which often trade-off optimality for compilation runtime. As a consequence, output circuits are often excessively deep, thus exceeding the allowed limits to maintain coherence.

In this work, we propose DDRoute[1], a depth-driven *routing* algorithm with a heuristic approach that aims to reduce the overhead in circuit depth introduced by the routing process. We employ a custom min-path procedure, which exploits a *generalized distance function* between qubits to schedule `SWAP` gates efficiently, allowing several quantum operations to be executed in parallel, in order to reduce the final depth of the output quantum circuits. Finally, we introduce DDPlace, a qubit *mapping* technique which provides convenient initial conditions for DDRoute.

Our work aims to provide *flexibility* with respect to the specific topology of the QPU; *scalability*, to maintain reduced compile time even in presence of hundreds of logical qubits; and *gate length awareness*, meaning that the instruction scheduling is sensitive to the duration of the supported gates on the QPU. We theoretically evaluate the computational complexity of DDRoute, and we provide experimental validation, demonstrating that we achieve up to 70% reduction in depth with respect to state-of-the-art routing algorithms.

[1]Our work is available at https://github.com/necst/ddroute



(a) (b)

Fig. 1: (a) Quantum circuit describing 3 logical qubits (i.e., q_0, q_1, and q_2), 3 Hadamard gates and 3 CNOT/CX gates. (b) Corresponding DAG representation, highlighting front and back layer of the circuit.

II. Preliminaries

In this Section, we provide an introduction to quantum computing and QRP. We specifically refer to the gate-based model of quantum information.

A. Gate-based Model of Quantum Computing

Quantum computing is a paradigm of computation based on the laws of quantum mechanics. Its unit of information is the *qubit*. While classical bits can lie in one of two states, 0 and 1, qubits are described as a *superposition* of two basis states, $|0\rangle$ and $|1\rangle$. The state of a qubit can be represented as a linear combination of the two basis states: $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex coefficients, and $|\alpha|^2 + |\beta|^2 = 1$ [15]. We can modify a quantum state by applying reversible operations, called *gates*. Current physical implementations of gates support either one operand (single-qubit gate) or two operands (two-qubit gate). Among the most employed single-qubit gates are the Pauli X gate (equivalent to a classical NOT operation) and the Hadamard H gate, often used to create uniform superpositions of basis states; among the most common two-qubit operations are the Controlled-X (CX or CNOT), which takes as input a control qubit and a target qubit and bit-flips the latter if the former is set to $|1\rangle$, and the SWAP operation, employed for swapping the state of two qubits. One SWAP gate can be implemented using three CNOTs, and more generally any n-qubit quantum gate can be decomposed in a set of single-qubit gates and CNOTs [16].

A common representation of quantum algorithms is the *quantum circuit*, where qubits are represented as wires and gates as blocks acting along the wires of their operands. An example of quantum circuit is given in Figure 1a. For circuit compilation, a more convenient representation is the Directed Acyclic Graph (DAG), where nodes represent gates, and edges represent dependencies between gates. We define the front layer F of a DAG as the set of nodes without incoming arcs, corresponding to the operations that can be executed without waiting for previous dependencies. The back layer B, instead, is the set of nodes without outgoing arcs. An example of DAG for the circuit in Figure 1a is shown in Figure 1b.

We now introduce three distinct concepts of *depth*, acting at different levels of the quantum algorithm. The *gate depth* of a node in a DAG is the length of the longest path between any of the nodes in the front layer F and the given node. As an example, the first CX from the left in Figure 1b has a gate depth equal to 1, since both its operands belong to F. The *qubit depth* of a given qubit, instead, is $1 +$ the length of the longest path between any of the nodes in F and the node corresponding to the last operation applied to that qubit. This can be interpreted as the number of *time steps* required to execute all the operations on a specific qubit, assuming that independent operations are performed in parallel, when possible. In Figure 1a, q_0 and q_2 have qubit depth equal to 4, while q_1 has qubit depth equal to 3. Finally, the *circuit depth* of a given circuit is defined as the maximum among the qubit depths of all the qubits belonging to that circuit. This can be interpreted as the number of time steps required to completely execute the given circuit. For the circuit in Figure 1b, the overall circuit depth is equal to $\max\{4, 3, 4\} = 4$.

B. The Qubit Routing Problem

Current NISQ devices mainly support only single-qubit and two-qubit operations, with limited gate set and connectivity. Therefore, in order to compile a generic n-qubit gate to a specific device backend, this operation is decomposed in single-qubit and two-qubit gates which are directly supported by the platform. After gate decomposition, the n logical qubits are associated with physical qubits on the device (*mapping* or *placement*), and operations are scheduled accordingly [17]. This mapping can be mathematically described by a function $\pi : q \mapsto \widetilde{q}$, mapping a logical qubit q to a physical qubit $\widetilde{q}$.

Whilst single-qubit gates can be directly applied onto their operands $\widetilde{q}$, two-qubit gates require that their operands are directly connected in the quantum processor. For example, if the two-qubit gate G is applied to logical qubits q_1 and q_2, then the corresponding physical qubits $\widetilde{q_1}$ and $\widetilde{q_2}$ must be directly connected. Such topological constraints can be visualized as a 2D *coupling map*, where nodes are physical qubits and edges represent supported interconnections for two-qubit operations [6]. In case such constraints are not respected by the initial mapping π, qubit routing must be performed to modify the physical mapping of q_1 and q_2, such that their newer mapping $\widetilde{q_1}$ and $\widetilde{q_2}$ are interconnected.

Assuming that we need to change the physical location of a logical qubit q from $\widetilde{q_a}$ to $\widetilde{q_b}$, two possible cases arise: if $\widetilde{q_a}$ and $\widetilde{q_b}$ are directly connected in the coupling map, a SWAP operation between $\widetilde{q_a}$ and $\widetilde{q_b}$ will be sufficient; alternatively, if $\widetilde{q_a}$ and $\widetilde{q_b}$ are not directly connected, the routing algorithm must find a path between $\widetilde{q_a}$ and $\widetilde{q_b}$ in the coupling graph, and schedule SWAP operations accordingly. Identifying a convenient selection of qubits to swap in order to minimize the overall circuit depth corresponds to solving the QRP.

C. Related Works

The literature provides a theoretical analysis of the QRP, showing that both the minimization of the number of SWAP gates [8], useful to improve fidelity, and the minimization of

the circuit depth [9] are NP-complete problems. Therefore, based on the complexity of the instances and the requirements of the users, current approaches to address QRP may prefer reduced execution time or higher accuracy. Thus, two families of algorithms were developed, having different objectives. The first family tackles the problem by formulating all the topological constraints and the quantum operations by means of mathematical constraints or clauses, to be resolved by a SAT solver or other exact procedures [17]–[22]. These algorithms provide accurate solutions, but their exponential time complexity makes their use unfeasible for large circuits. For instance, the exact solution introduced in [17] requires $\mathcal{O}((n!)^2 \cdot n \cdot g)$ steps to solve the QRP on a circuit with n qubits and g two-qubit gates, and other similar approaches only tackle quantum circuits with few qubits and hundreds of operations. The second family, currently in use in modern quantum computing frameworks, follows a heuristic approach, effectively reducing execution time and providing good solutions, but without guaranteeing optimality [10]–[14]. As an example, the approach employed in t|ket⟩ [12] and Cirq [13] is to discard single-qubit gates and divide the circuit into independent sets of two-qubit gates called *timesteps* or *layers*. Then, the QRP is solved on each timestep, employing a heuristic cost function to select the `SWAP`s to be inserted. A similar approach is followed by QMAP [14], which employs an A*-search-based procedure to find the best `SWAP`s, and StochasticSwap [11], which uses a randomized approach to select an efficient permutation to be applied to logical qubits. Following a different direction, the algorithm SABRE [10] does not act on independent timesteps, but performs routing incrementally using a similar heuristic cost function, with a worst-case time complexity of $\mathcal{O}(gn^{2.5})$.

The primary objective of these algorithms is to reduce the number of additional gates, yet *not* to minimize the final depth of the given circuit. Their final solutions often present excessive depth, which is inconvenient for the satisfaction of the coherence time constraints. Moreover, their heuristics often exclude solutions where the addition of further `SWAP`s may result in a circuit with reduced depth, due to parallelization of operations. We deem that this feature is pivotal for NISQ devices, whose supported circuit depth is still limited [23]. Therefore, DDRoute follows a *depth-driven* approach.

III. Our Proposed Algorithm

This Section introduces DDRoute, DDPlace, and an analysis of the computational complexity of both components.

A. DDRoute: Algorithm Overview

Given a logical circuit C_l, the goal of DDRoute is to compute an equivalent physical circuit C_p where all the two-qubit gates comply with the coupling map of the device. To do so, logical operations from C_l are sequentially evaluated and added to C_p, along with the necessary `SWAP` gates.

The setting conditions of DDRoute, i.e., the initial mapping $\pi(\cdot)$, are provided by DDPlace (Section III-B), and are tailored to reduce the number of `SWAP`s needed to route the first instructions of C_l. After this initialization, DDRoute begins

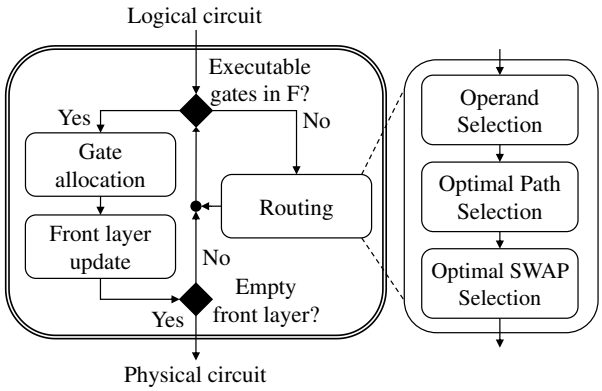
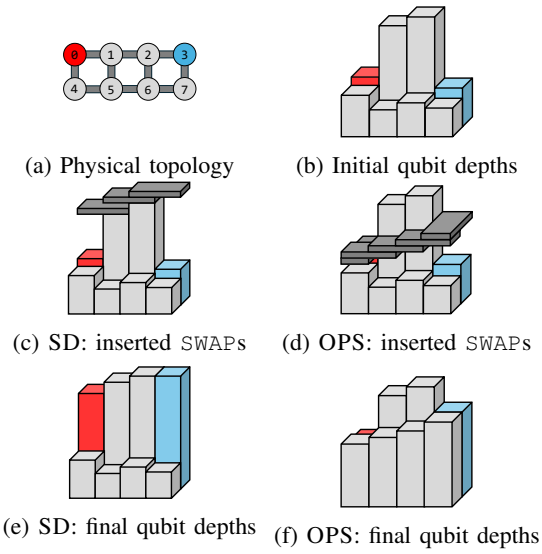


Fig. 2: Overview of DDRoute. The procedure explores F and performs routing when necessary. At every iteration, two-qubit gates are allocated in the physical circuit and F is updated.

by inspecting F, as shown in Figure 2. If one of the gates in F is directly executable, i.e., its operands are mapped to adjacent physical qubits, it can be added to C_p. Afterwards, such a gate is removed from the DAG, and F is updated accordingly. If none of the gates in F can be executed due to topological constraints, it means that the circuit contains two-qubit gates whose operands are not interconnected on the coupling graph. Therefore, `SWAP` operations have to be scheduled to allow those two-qubit gates to be executed. The criterion for choosing which `SWAP` operations to perform is pivotal, since it defines the depth of the final circuit. Our `SWAP` allocation process is based on three main heuristic components: OperandSelection, OptimalPathSelection, and OptimalSWAPSelection. DDRoute iteratively exploits such components until all the gates in C_l have been correctly routed.

1) Operand Selection: The OperandSelection procedure is responsible for choosing the *operands* of the `SWAP` gates, i.e., two qubits that need to be routed. The first one is the input of a gate in F whose mapping needs to be modified. The choice of a specific qubit aims to minimize the schedule time of the inserted `SWAP`s, to reduce the overall circuit depth. To do so, we select the logical operand whose corresponding physical qubit (according to the current mapping) has a low *qubit depth*. Since the depth of a possible `SWAP` applied to the selected qubit is determined by the qubit depth of both its operands, we improve this metric by also considering the qubit depth of the neighbors of the physical qubit. The one which minimizes these factors is selected as the current operand to be routed. We refer to the selected logical qubit as q_{curr}, to the corresponding gate in F as g_{curr}, and to the other operand as q_{oth}.

2) Optimal Path Selection: The OptimalPathSelection routine is employed to compute a path in the topology graph that connects the physical qubits corresponding to the operands selected in the previous step. Inserting `SWAP`s along this path would route the selected logical qubit in a position adjacent to its corresponding operand, making the



(a) Physical topology (b) Initial qubit depths

(c) SD: inserted SWAPs (d) OPS: inserted SWAPs

(e) SD: final qubit depths (f) OPS: final qubit depths

Fig. 3: Shortest Distance (SD) and OPTIMALPATHSELECTION (OPS) on a routing problem between qubit 0 and 3.

operation feasible on hardware. Most algorithms employ a simple shortest-path approach, while the goal of OPTIMALPATHSELECTION is to exploit the same efficient shortest-path procedures, yet with a tailored, depth-driven distance metric. To do so, we define a *generalized distance function* $\delta(\widetilde{p})$, able to reduce the depth of each operation along the path and, most importantly, the depth of the final circuit.

More specifically, given a physical source qubit $\widetilde{q}$, another physical qubit $\widetilde{p}$ in the coupling map, and a path ρ from $\widetilde{q}$ to $\widetilde{p}$, we define $\delta_{depth}(\widetilde{p})$ as the depth of the last SWAP gate along the path ρ; $\delta_{swap}(\widetilde{p})$ as the number of SWAP required to reach $\widetilde{p}$; and $\delta_{next}(\widetilde{p})$, that is a score depending on the quality of the mapping resulting after applying the SWAP gates required by ρ. In particular, for each of the gates immediately after g_{curr} in the logical circuit, a positive penalty is assigned to SWAP gates that bring its operands further apart and a negative discount is assigned to SWAP gates that bring its operands closer.

Thus, we combine such scores and define the generalized distance $\delta(\widetilde{p})$ from $\widetilde{q}$ to $\widetilde{p}$ as:

$$\delta(\widetilde{p}) = \delta_{depth}(\widetilde{p}) + \delta_{swap}(\widetilde{p}) + \delta_{next}(\widetilde{p})$$

This definition allows the distance function to be efficiently computed using Dijkstra's algorithm, with the same computational complexity as a simple shortest-path computation. The algorithm visits one physical qubit at a time, selecting at each step the nearest one not yet visited and updating the shortest path to its neighbors. The procedure terminates when the visited node is the routing destination.

Figure 3 provides an example with a physical topology of eight qubits (3a). We assume that a certain number of gates has already been allocated in C_p, with the qubit depths represented by the height of the corresponding columns (3b). Notice that the *circuit depth* equals to the highest column. The goal is to modify the mapping of an operand from qubit 0 to qubit 3 by means of SWAPs (represented as darker blocks). The simple shortest distance (SD) approach solves the problem by inserting three SWAPs (3c), with an unnecessary increase in the circuit depth (3e). The OPTIMALPATHSELECTION (OPS), instead, follows a seemingly longer path (3d), yet reaching the destination qubit at a lower depth without increasing the overall circuit depth thanks to $\delta(\widetilde{p})$ (3f).

3) Optimal SWAP Selection: During the OPTIMALPATHSELECTION, the routing algorithm obtains a path ρ between the physical operands $\widetilde{q} = \pi(q_{curr})$ and $\widetilde{p} = \pi(q_{oth})$. Placing *all* the SWAPs along the path ρ is not, however, the best strategy, since it does not take into account the possible SWAPs that can be performed by other operands in parallel. A simple yet effective strategy to exploit parallelism, employed by the OPTIMALSWAPSELECTION, is to allocate just the first operation along the path, updating the mapping of q_{curr} accordingly. If $\pi(q_{curr})$ and $\pi(q_{oth})$ are now adjacent, meaning that the end of ρ has been reached, then the corresponding gate g_{curr} can be inserted in C_p and removed from F. If $\pi(q_{curr})$ and $\pi(q_{oth})$ are not adjacent, then, instead of inserting the remaining SWAPs, the routing process is restarted from the OPERANDSELECTION step. Then, if the selected operand is still q_{curr}, the optimal path computation is not required, as the best operation to be inserted is the next SWAP in the previously computed path ρ. If the selected operand is different, instead, the previous optimal path is discarded and OPTIMALPATHSELECTION is performed again. This assures that all the SWAPs added to C_p are associated with the current best operand, thus reducing the schedule time of all the inserted SWAPs. Once all the gates in C_l have been inserted in C_p, causing F to be empty, the routing process can terminate and C_p can be executed on the QPU.

Finally, a key aspect is the allocation of single-qubit gates. Most routing algorithms ignore the presence of single-qubit operations as they are not subject to topological constraints, inserting them in the circuit after the routing procedure. To reduce their depth contribution, DDRoute postpones their allocation until a two-qubit gate, i.e., an operation from C_l or a routing SWAP, has to be inserted. In the first case, it has no choice but to insert all the postponed single-qubit gates right before the new operation. In the case of a SWAP, instead, it can exploit the fact that the SWAP gate depth depends on the highest qubit depth among its operands. If the physical operand with the lowest qubit depth corresponds to the physical mapping of a logical qubit, then the routing procedure inserts single-qubit gates acting on that logical qubit before the new SWAP, until it fills the difference in qubit depth between the two qubits or it meets a two-qubit gate or the end of the circuit. Using this strategy, the algorithm can minimize the contribution of single-qubit gates to the final circuit depth.

B. DDPlace

DDPlace is our depth-driven procedure to compute the initial mapping $\pi(\cdot)$ for DDRoute. It begins by sorting all the two-qubit gates in C_l by depth. Then, it selects two random physical qubits and maps the operands of the first two-qubit

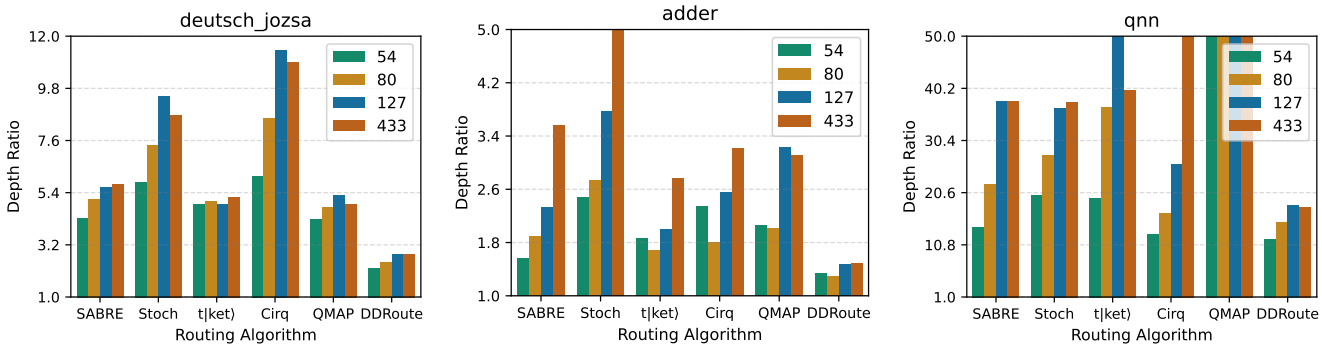


Fig. 4: Depth ratio comparison on DEUTSCH_JOSZA, ADDER, and QNN circuits, targeting the 54-qubit Google Sycamore, 80-qubit Rigetti Aspen, 127-qubit IBM Eagle, and 433-qubit IBM Osprey topologies.

gate onto them. The procedure then iterates over the remaining two-qubit gates, ordered by depth, mapping their remaining operands. If both the operands have not been mapped yet, they are assigned to two physical qubits which are the closest to the last assigned physical qubit. If one of the two operands has not been mapped yet, it is associated with the *free* physical qubit closest to the one assigned to the other operand. Finally, if some logical qubits have not been mapped, they are assigned to the physical qubits closest to the last assigned physical qubit.

C. Computational Complexity

The computational complexity of DDPlace is $\mathcal{O}(g \log n)$, where g and n are the number of two-qubit gates and the number of logical qubits in the circuit, respectively. Indeed, computing the depth of all the two-qubit gates in the graph requires $\mathcal{O}(g)$ operations. Then, F is continuously updated using a binary min-heap. Since the number of two-qubit gates in F at each step is at most $\lfloor \frac{n}{2} \rfloor$, removing the gate with the lowest depth and updating F requires $\mathcal{O}(\log n)$ operations. In the worst case, when the whole circuit traversal is necessary, DDPlace requires $\mathcal{O}(g \log n)$ operations.

The computational complexity of DDRoute is $\mathcal{O}(gn^{1.5} \log n)$. Indeed, on a 2D coupling map, the average number of `SWAP`s required to map two operands of a gate onto adjacent physical qubits is $\mathcal{O}(\sqrt{n})$, resulting in a total of $\mathcal{O}(g\sqrt{n})$ `SWAP` insertions. In the worst case, the selected operand q_{curr} is changed for each `SWAP`, causing the recomputation of the optimal path ρ, where $\mathcal{O}(n \log n)$ steps are required by the OPTIMALPATHSELECTION procedure. Therefore, the complexity of the whole algorithm is $\mathcal{O}(g\sqrt{n}(n \log n)) = \mathcal{O}(gn^{1.5} \log n)$.

IV. EXPERIMENTAL RESULTS

In this Section, we experimentally evaluate DDRoute, and its DDPlace initialization, comparing its performance against other state-of-the-art algorithms. We consider multiple benchmark suites and target superconducting architectures.

A. Experimental Setup

Our evaluations aim to compare the performance of DDRoute, initialized by DDPlace, against other state-of-the-art procedures. The algorithms we use for comparison are SABRE [10] with Qiskit v1.2.0, t|ket⟩ [12] v1.31.0, Google Cirq [13] v1.4.1, and MQT QMAP [14] v2.6.0, initialized by their default mapping technique; and StochasticSwap [11], with Qiskit DenseLayout [24] as a mapping procedure. We consider state-of-the-art benchmark suites: the Munich Quantum Toolkit Benchmark [25], QASMBench [26] v1.4, VeriQBench [27], and Qiskit Circuit Library [28], and target the following QPUs: Google Sycamore [29] (54 qubits), Rigetti Aspen [30] (80 qubits), IBM Eagle [31] (127 qubits), IBM Osprey [32] (433 qubits), and IBM Condor [33] (1121 qubits). All our experiments were executed on a 32-core Intel Xeon E5-2680 CPU with 32GB of memory and the Ubuntu 24.04 LTS operating system, setting a compile time limit of 3 hours. We perform three classes of testing: a depth analysis, a scaling analysis, and an analysis on gate length awareness.

B. Depth Analysis on Benchmark Circuits

This class of experiments compares DDRoute against the State of the Art, targeting medium-sized circuits and QPUs, with a focus on the depth overhead introduced by the routing. To provide more accurate results, we run all stochastic algorithms 30 times, and we average depth and compile time values. We define the *depth ratio* of circuit c as:

$$\text{DepthRatio[c]} = \frac{\text{FinalDepth[c]}}{\text{InitialDepth[c]}}$$

which quantifies the overhead of the routing process on the final depth. Indeed, the addition of `SWAP` gates produces outputs with increased depth with respect to the input circuit. The more the routing procedure is efficient, the closer the depth ratio to 1. Additionally, we define the improvement in depth of DDRoute with respect to a generic algorithm ALG for circuit c as:

$$\text{Improvement[c]} = \frac{\text{FinalDepth}_{\text{ALG}}[c] - \text{FinalDepth}_{\text{DDRoute}}[c]}{\text{FinalDepth}_{\text{ALG}}[c]}$$

The results of our experiments, such as the ones presented in Figure 4, show a consistent reduction in circuit depth. DDRoute is able to outperform all the algorithms employed in the current quantum computing frameworks, both in depth

TABLE I: Improvement of DDRoute with respect to the State of the Art, targeting the 80-qubit Rigetti Aspen, the 127-qubit IBM Eagle, and the 433-qubit IBM Osprey topology.

#Qubits	Improvement (%) of DDRoute w.r.t.				
	SABRE	Stochastic	t|ket⟩	Cirq	QMAP
54	25.52	46.98	39.13	26.14	51.62
80	34.93	47.17	41.51	29.05	55.10
127	36.06	53.87	47.41	45.27	70.86
433	47.97	58.28	47.53	60.68	71.15
Average	**36.88**	**52.02**	**43.02**	**40.92**	**62.56**

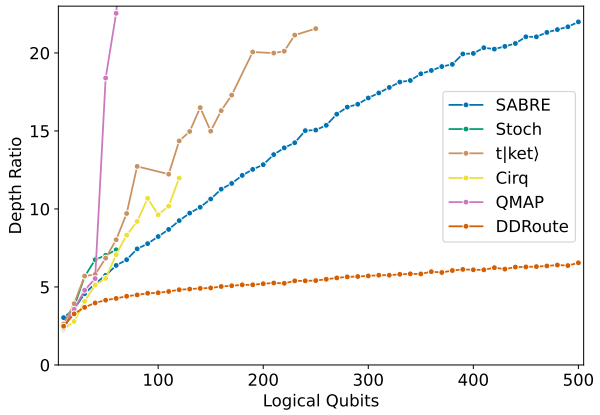


Fig. 5: Depth ratio comparison on circuits generated with Qiskit Circuit Library , targeting the 1121-qubit IBM Condor.

and compile time, with an average depth improvement of over 36.8% and a geomean average speedup of 92.8× over all experiments. We point out that the achieved speedup in compilation time also depends on the execution runtime of the algorithms, which is based on Python and C++ for DDRoute, and mainly on Python for the State of the Art. Regarding the SWAP count of the output circuits, we note that DDRoute is outperformed by algorithms like SABRE and QMAP, where the minimization of the number of added operations is the main target objective, rather than circuit depth. Table I shows a detailed comparison of DDRoute with the six algorithms on circuits and topologies with different sizes. We highlight that the increase in circuit size and physical qubits produces up to 71.15% improvement.

C. Scalability Analysis & Awareness of Gate Length

To further quantify the scalability trend, we performed another set of experiments targeting the 1121-qubit IBM Condor topology. We generated input circuits combining several components of the Qiskit Circuit Library [28], from 10 to 500 qubits. The results of these experiments are shown in Figure 5. Out of the six compared algorithms, only DDRoute and SABRE were able to complete routing for all circuits without exceeding the time limit of 3 hours. As it can be seen, DDRoute outperforms SABRE, with an average depth ratio of 6.51 against 21.99 for the largest test case, reducing the circuit depth by 70.4%. Regarding the count of SWAPs, which is *not* the main target of DDRoute, SABRE only achieves 38.7% improvement over DDRoute in the largest test case. Therefore, DDRoute proves to provide an important reduction in circuit depth with only a limited increase in SWAPs.

TABLE II: Depth comparison with different metrics between SABRE and DDRoute, on circuits generated with Qiskit Circuit Library, targeting the 1121-qubit IBM Condor.

#Qubits	**Metric**	**SABRE**	**DDRoute**	**Improvement** (%)
100	1-1-3	43065	24206	43.79
	1-2-6	82082	46171	43.75
	1/0-11-33	430457	241768	43.83
300	1-1-3	270919	90509	66.59
	1-2-6	530601	172695	67.45
	1/0-11-33	2853356	919159	67.79
500	1-1-3	581238	172020	70.40
	1-2-6	1143512	329845	71.16
	1/0-11-33	6183399	1761279	71.52

As a final class of experiments, we aim to evaluate the awareness of gate length, i.e., the ability of the routing procedure to take into account $\tau(g)$, corresponding to the duration of each gate g, and minimize the circuit depth accordingly. To perform such evaluation, we repeated the experiments on the largest available topology, IBM Condor, varying the metric over which the circuit depth is computed. We employed three different metrics for $\tau(g)$: **1-1-3**, where the depth of single-qubit gates, CXs, and SWAPs are 1, 1, and 3, respectively; **1-2-6**, where the depth of single-qubit gates, CXs, and SWAPs are 1, 2, and 6 [34]; and **1/0-11-33**, where X and SX gates have a depth of 1, RZs have a depth of 0, CXs of 11, and SWAP operations of 33. The latter is proportional to the average gate execution time on IBM devices [35], [36]. Table II shows a comparison for some of the test cases between SABRE and DDRoute, highlighting that DDRoute achieves significant improvements in the final depth for all three metrics, with the ones reflecting different gate durations having an even more important reduction, up to 71.52%. For further flexibility, we also provide the possibility of specifying different $\tau(g, \widetilde{q_1}, \widetilde{q_2})$ for different operands of the same gate.

V. Conclusion

In this paper, we proposed DDRoute, a depth-driven approach to the QRP, which compiles logical quantum circuits into physical ones, while reducing the overall depth overhead. We devised a generalized distance function, able to quantify the impact of SWAP insertions on the final circuit depth. We also introduced DDPlace, a depth-driven mapping procedure to generate convenient initial conditions for the routing algorithm. Experimental results show that our approach is able to effectively reduce the depth of the routed circuits, achieving up to 70% improvement over state-of-the-art solutions.

As the next steps, we aim to tune the definition of the generalized distance function to keep into consideration additional factors, such as the hardware-specific tradeoff between SWAP fidelity and coherence time.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, ser. SFCS '94. USA: IEEE Computer Society, 1994, p. 124–134. [Online]. Available: https://doi.org/10.1109/SFCS.1994.365700

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[3] IBM Quantum. (2016) IBM Quantum. IBM. Accessed: 2024-08-21. [Online]. Available: https://www.ibm.com/quantum

[4] Rigetti Computing. (2013) Rigetti Computing. Rigetti Computing. Accessed: 2024-08-21. [Online]. Available: https://www.rigetti.com/

[5] Google Quantum AI. (2013) Google Quantum AI. Google Quantum AI. Accessed: 2024-08-21. [Online]. Available: https://quantumai.google

[6] J. Kusyk, S. M. Saeed, and M. U. Uyar, "Survey on quantum circuit compilation for noisy intermediate-scale quantum computers: Artificial intelligence to heuristics," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–16, 2021.

[7] S. Resch and U. R. Karpuzcu, "Benchmarking quantum computers and the impact of quantum noise," *ACM Comput. Surv.*, vol. 54, no. 7, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3464420

[8] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, and Y. Okamoto, "Algorithmic theory of qubit routing," in *Algorithms and Data Structures*, P. Morin and S. Suri, Eds. Cham: Springer Nature Switzerland, 2023, pp. 533–546.

[9] D. B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Transactions on Computers*, vol. 70, no. 09, pp. 1363–1373, 9 2021.

[10] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1001–1014. [Online]. Available: https://doi.org/10.1145/3297858.3304023

[11] Qiskit. (2017) StochasticSwap. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.StochasticSwap

[12] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, 2020. [Online]. Available: https://doi.org/10.1088/2058-9565/ab8e92

[13] Quantum AI team and collaborators, "Cirq," 10 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4062499

[14] R. Wille and L. Burgholzer, "Mqt qmap: Efficient quantum circuit mapping," in *Proceedings of the 2023 International Symposium on Physical Design*, ser. ISPD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 198–204. [Online]. Available: https://doi.org/10.1145/3569052.3578928

[15] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[16] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, pp. 3457–3467, 11 1995.

[17] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 international symposium on code generation and optimization*, 2018, pp. 113–125.

[18] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, "Optimal Qubit Assignment and Routing via Integer Programming," *ACM Transactions on Quantum Computing*, vol. 4, no. 1, 10 2022. [Online]. Available: https://doi.org/10.1145/3544563

[19] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via MaxSAT," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1078–1091.

[20] P. Zhu, X. Cheng, and Z. Guan, "An exact qubit allocation approach for NISQ architectures," *Quantum Information Processing*, vol. 19, no. 11, p. 391, 11 2020. [Online]. Available: https://doi.org/10.1007/s11128-020-02901-4

[21] A. Shafaei, M. Saeedi, and M. Pedram, "Qubit placement to minimize communication overhead in 2D quantum architectures," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 495–500.

[22] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 360–374. [Online]. Available: https://doi.org/10.1145/3445814.3446706

[23] IBM Quantum. (2024) Development Roadmap. IBM Quantum. Accessed: 2024-09-12. [Online]. Available: https://www.ibm.com/quantum/technology#roadmap

[24] Qiskit. (2017) DenseLayout. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.DenseLayout

[25] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," *Quantum*, vol. 7, p. 1062, Jul. 2023. [Online]. Available: https://doi.org/10.22331/q-2023-07-20-1062

[26] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, feb 2023. [Online]. Available: https://doi.org/10.1145/3550488

[27] K. Chen, W. Fang, J. Guan, X. Hong, M. Huang, J. Liu, Q. Wang, and M. Ying, "VeriQBench: A Benchmark for Multiple Types of Quantum Circuits," 2022. [Online]. Available: https://arxiv.org/abs/2206.10880

[28] Qiskit. (2017) Qiskit Circuit Library. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/circuit_library

[29] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[30] Rigetti Computing. (2022) Rigetti Aspen. Rigetti Computing. Accessed: 2024-08-21. [Online]. Available: https://investors.rigetti.com/news-releases/news-release-details/rigetti-computing-announces-commercial-availability-80-qubit

[31] IBM Quantum. (2021) IBM Eagle. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://www.ibm.com/quantum/blog/127-qubit-quantum-processor-eagle

[32] ——. (2022) IBM Osprey. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two

[33] ——. (2023) IBM Condor. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://www.ibm.com/quantum/blog/quantum-roadmap-2033

[34] H. Deng, Y. Zhang, and Q. Li, "Codar: a contextual duration-aware qubit mapping for various nisq devices," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[35] Qiskit. (2017) BackendProperties. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/qiskit.providers.models.BackendProperties

[36] ——. (2017) RZGate. IBM Quantum. Accessed: 2024-08-21. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.RZGate

McPAL: Scaling Unstructured Sparse Inference with Multi-Chiplet HBM-PIM Architecture for LLMs

Shiwei Liu[†], Zhirui Huang[†], Jiangnan Yu[§], Qi Liu[†], Chixiao Chen[†‡*]
[†]State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China [‡]Shaoxin Laboratory, Zhejiang, China
[§]The Hong Kong University of Science and Technology, Hong Kong SAR, China

***Abstract*—Large language models (LLMs) have gained significant attention recently. However, executing LLM is memory-bound due to the extensive memory accesses. Process-in-memory (PIM) emerges as an energy-efficient solution for LLMs, delivering high memory bandwidth and compute parallelism. Nevertheless, the trend towards larger LLMs introduces escalating memory footprint challenges for monolithic PIM chips. This paper proposes McPAL, which tackles this challenge by emphasizing unstructured sparse compute within PIM and hierarchical multi-chiplet scaling. McPAL decomposes arbitrary sparse weight matrix into multiple irregular sparse vectors. The non-skipped computations in each vector are then routed via an in-memory butterfly network to the standard PIM array, enhancing the PIM utilization. In addition, we scale McPAL vertically by strategically organizing the 3D-HBM hierarchy to minimize the internal long-distance data travel. Meanwhile, a 2.5D IO chiplet scales McPAL horizontally, reducing die-to-die (D2D) data transfer and ensuring sparse workload balance. We conducted extensive experiments from Llama-7B to Llama-70B. The results show that McPAL achieves 1.57× to 3.12× speedup and 10.43× to 35.66× energy efficiency over Nvidia A100 GPU. Compared to SOTAs, McPAL also achieves 1.08× to 2.15× speedup and 1.65× to 5.14× energy efficiency.**

***Index Terms*—HBM, Chiplet, Large Language Model, Unstructured Sparsity, Processing-in-Memory**

I. Introduction

LLMs exhibit superior capability to learn long-range contextual dependencies and remarkable scalability as parameters grow, from GPT-2 1.5 billion [1] to GPT-4 1.76 trillion [2] in just 5 years. As LLMs continue to expand, a simultaneous need arises to scale up compute throughput, memory capacity, and inter-chip bandwidth. Conventional architectures like GPUs typically store all parameters in an external memory chip (e.g. DRAM) and perform LLMs on another separate processor chip, moving large-scale weights back and forth. Unfortunately, CMOS compute logic scales much faster than inter-chip links and memory capacity [4]. As a result, the performance of GPU-DRAM systems is often constrained by the limited bandwidth between memory and processor chips.

PIM is a promising solution for memory-intensive LLMs. It features ultra-tight coupling between compute and storage cells, avoiding the performance bottleneck limited by the memory-processor bandwidth. PIM pioneers leverage dataflow optimization [10]–[12] and attention sparsity [13]–[15] to accelerate self-attention in LLMs, but leave linear layers unexplored. These linear layers frequently become bottlenecks during LLM inference. For example, the Llama-7B model [3] with 4K tokens requires 6.47 billion weights for its linear layers, compared to just 1.07 billion for self-attention's key-value caching. The extensive weight parameters take up 85.8% of model storage, placing significant memory demands on LLM systems.

Sparsification Techniques [16]–[18] are widely adopted to reduce the memory footprint of linear layers. However, the irregular memory access and compute pattern of sparse weight matrices are inherently incompatible with the regular PIM structure. Consequently, previous PIM designs primarily support structured sparsity, where weights are pruned in block-wise [21], [22] or channel-wise [23] patterns. However, SparseGPT [6] proves that structured sparsity could lead to performance degradation in LLMs. In addition, ***Multi-Chiplet Integration*** emerges as another approach to manage memory footprint, breaking the monolithic limitation of a single PIM chip. Heterogeneous integration enables the creation of a virtual "*super-die*" by dividing it into multiple chiplets. Thanks to the flexibility and scalability, various chiplet architectures [24], [25] were proposed for high-performance computing. However, these works primarily optimize compute-bound tasks, such as matrix multiplication, but lack specific optimizations tailored for memory-bound LLMs, resulting in redundant storage and inefficient data transfer [19].

In summary, efficient LLM execution in PIM requires satisfying the following requirements: (1) The overall architecture should support the full scope of LLM operations, including not only self-attention but also linear layers and other components. (2) The PIM chiplet should accommodate unstructured weight sparsity to minimize the memory overhead without compromising LLM accuracy and hardware utilization. (3) The PIM-chiplet system meticulously optimizes LLM workflows, maximizing compute/storage utilization while minimizing inter-chiplet communication. This paper proposes McAPL to address these challenges through the following contributions,

- McPAL explores unstructured sparsity in HBM-PIM through a novel matrix decomposition algorithm. It decomposes arbitrary sparse weight matrices into irregular sparse vectors, which are then routed through an in-memory butterfly network for regular vector products in PIM. This method delivers nearly 2× speedup at 50% sparsity, significantly reducing memory footprint without compromising LLM accuracy or hardware utilization.
- McPAL introduces a hierarchical multi-chiplet scaling topology. We first scale McPAL vertically by strategically organizing the 3D-HBM hierarchy to minimize HBM's internal long-distance data travel by through-silicon vias (TSVs). A dedicated 2.5D IO chiplet furthers scales McPAL vertically to reduce D2D data synchronization and enables sparse workload balance.
- We conduct extensive experiments to test McPAL scalability from Llama-7B to Llama-70B models. The results show that McPAL achieves 1.57× to 3.12× speedup and 10.43× to 35.66× energy efficiency over Nvidia A100 GPU. Compared to SOTA accelerators, McPAL also achieves 1.08× to 2.15× speedup and 1.65× to 5.14× energy efficiency.

II. Background and Motivation

A. LLM Preliminaries

Fig. 1 illustrates the LLM structure, comprising an embedding layer, a stack of transformer layers and a language model (LM) head. The embedding converts discrete tokens into feature vectors, while

* Chixiao Chen is the corresponding author, cxchen@fudan.edu.cn.

This work was supported by the National Key Research and Development Program of China under Grant No. 2023YFB4404402 and the National Natural Science Foundation of China under Grant No. 62495101.

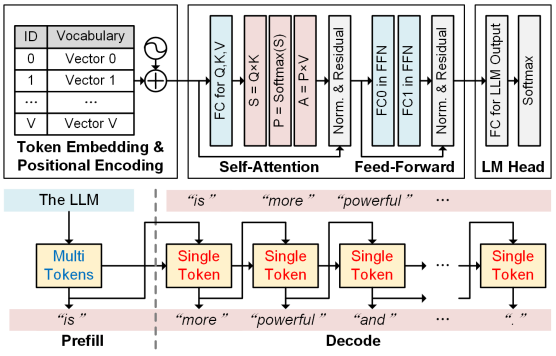


Fig. 1. LLM and transformer topology.

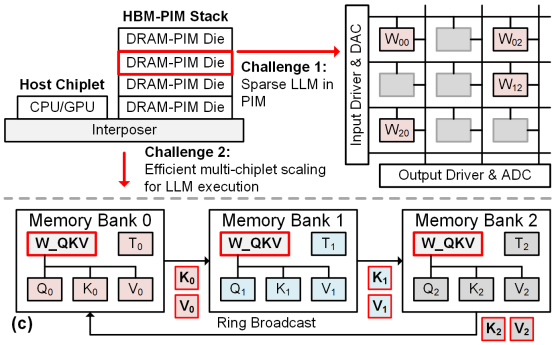


Fig. 2. Challenges for HBM-PIM chiplet architecture.

the LM head predicts the next token based on the transformer output. Between these lies the transformer layer, which is the core component of LLMs. It consists of two consecutive blocks: self-attention and feed-forward. The self-attention first projects input vectors into three intermediate features: Query (Q), Key (K) and Value (V). The Q×K yields attention scores (S), then normalized by a softmax to produce attention probabilities (P). Next, the P×V yields attention outputs. These outputs are fed into the feed-forward network, which comprises several linear layers separated by non-linear activation functions.

Most LLMs are pretrained for causal modelling, essentially as the next-token predictors. As shown in Fig. 1, LLMs typically contain two distinct phases. The prefill phase processes the input prompt in a single inference to generate the first new token. Since all tokens in the prompt are predetermined, LLMs perform matrix-matrix multiplication, making the prefill compute-bound. In contrast, the decode phase aggressively generates a single token per inference, serving as the input in the next iteration. The KV vectors of the input prompt and the generated tokens are stored to construct the "*KV Cache*", which is reused in the subsequent decode inference. The decode phase involves vector-matrix multiplication and is memory-bound. According to S3 [28], A100 GPU utilization reaches 68.4% during GPT-J model inference with a batch size of 512 (compute-bound), but drops to 0.4% with a single batch (memory-bound). It indicates that bandwidth limitation is the main bottleneck in LLM acceleration. Therefore, the PIM, offering high memory bandwidth, is a promising solution for accelerating memory-bound LLMs.

B. Motivations for HBM-PIM-Chiplet Architecture

The chiplet architecture integrates multiple smaller dies through 2.5D/3D advanced packaging technologies. This modular approach offers several advantages, including improved yield, reduced development time, and enhanced customization. Combining large-capacity HBMs, the HBM-PIM-chiplet architecture is developed. While HBM-PIM [19] and chiplet [24] are individually prevalent, their integration into a multi-chiplet HBM-PIM architecture raises new challenges:

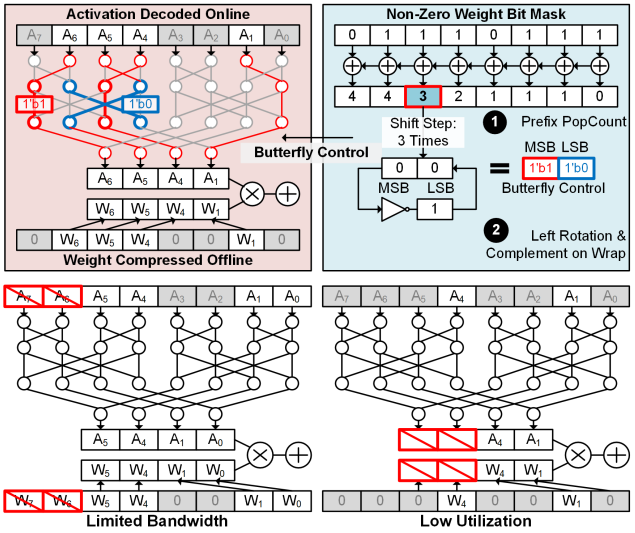


Fig. 3. (a) BFLY network to map M:N sparsity in PIM. and (b) The challenge of mapping unstructured sparsity in PIM using BFLY.

❶ **Efficient LLM storage in HBM-PIM.** The billions of parameters in LLMs create substantial memory demands, far exceeding the photolithographic limitation of a single HBM chiplet. While sparsification can minimize LLM size, most HBM-PIM designs struggle with under-utilization when mapping sparse weight matrices. Given the PIM crossbar in Fig. 2, the randomly distributed sparsity patterns lead to DRAM cells filled with zeros, resulting in low utilization. Consequently, most PIM designs only support block-wise [21], [22] or channel-wise [23] structured sparsity. SparseGPT [6] proves that structured sparsity could degrade LLM performance. Therefore, a PIM design that supports unstructured sparsity is necessary, distinguishing our work from other HBM-PIM works. ❷ **Efficient multi-chiplet scaling for LLM execution.** TransPIM [19] is the first PIM design to explore LLM scaling by distributing the input prompt across multiple HBM banks. As shown in Fig. 2, it duplicates all weight parameters in bank groups to enable parallel computing, which is impractical for larger LLMs. Moreover, it ring-broadcasts and synchronizes the KV cache for each input token. Since the KV cache size scales with the number of tokens, TransPIM incurs massive data transfers, rendering it inefficient for handling long input sequences, such as those with 4K tokens. Finally, TransPIM optimizes matrix multiplication within the HBM-PIM die, but overlooks auxiliary operations such as normalization, embedding, etc. In practical LLM applications, these remaining operations necessitate an additional master, which diminishes PIM efficiency by introducing substantial long-distance data travel within HBM or even inter-chip communication. To mitigate this issue, an efficient multi-chiplet scaling scheme tailored for LLMs is essential, setting McPAL apart from other chiplet-based architectures.

III. Sparsity-Aware McPAL PIM Design

A. Local Bank-Balance M:N Sparsity in PIM

Mapping unstructured sparsity to PIM is the knob to reduce memory footprint without sacrificing LLM performance. We first review the local-balanced M:N sparsity [8], [9], which lays the foundation for the subsequent exploration of the fully unstructured sparsity in McPAL HBM-PIM. In M:N sparsity, the weight matrix typically reserves ***M*** non-zero weights (NZWs) out of ***N*** elements within each weight vector. The sparse vector is then compressed and encoded by a bitmask. Guided by the bitmask, the activations

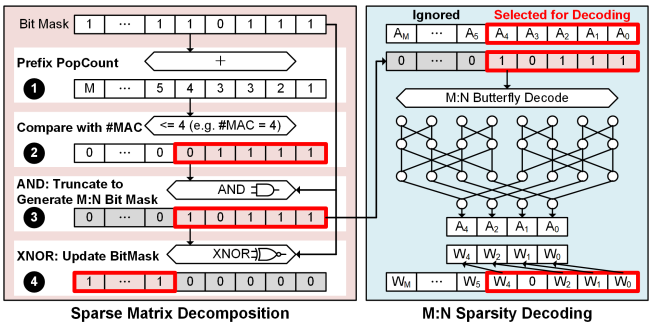


Fig. 4. Decompose sparse weight matrix to avoid limited bandwidth in BFLY.

associated with the NZWs are routed from the original inputs. The product between the compressed NZWs and the extracted activations effectively converts the M:N sparsity into regular vector products, which is well-supported in many PIM designs.

Fig. 3 depicts the M:N sparse compute flow and the butterfly network (BFLY) to route the non-skipped activations to NZWs [9]. The BFLY decoder contains two primary operations: prefix popcount and left-rotation complement-on-wrap (LRCW). The popcount performs cumulative sum on the given bitmask. The count of bit-1 indicates the number of NZWs up to a certain position. LRCW is a standard left rotation except that the MSB is complemented whenever it wraps around. Fig. 3 shows how to generate the control bits for two BFLY switches in stage 2. The two LSBs of the popcount result, PP[5], are used as the rotation step to execute LRCW on two zeros ('00'), which yields control bits '10'. Other switches follow a similar decoding method just with different popcount results. BFLY accurately directs the activations to the paired NZWs in M:N sparsity. However, due to the random distribution of NZWs in arbitrary unstructured sparsity, BFLY may cause limited bandwidth or low utilization. As shown in Fig. 3, the limited bandwidth arises when the number of NZWs exceeds BFLY's output bandwidth, preventing simultaneous extraction of all required activations. Conversely, low utilization happens when the weight vector is highly sparse, resulting in insufficient workloads to fully utilize the downstream MACs.

B. Unstructured Sparsity in McPAL PIM

McPAL leverages an algorithm-architecture co-design to enable arbitrary unstructured sparsity in PIM. ❶ Sparse weight matrix decomposition (SMD) and ❷ double-buffering BFLY (DB^{2}FLY) address the limited bandwidth and low utilization issues, respectively.

*1) **Sparse Weight Matrix Decomposition**:* The essence of SMD is to break down any sparse weight matrix into multiple M:N sparse vectors, which are then decoded as described in Sec. III-A. As shown in Fig. 4, SMD includes four steps to align the excess NZWs with both BFLY bandwidth and the downstream MACs (#MAC). ❶ Perform prefix popcount on the input bitmask to count the number of NZWs up to each position. ❷ Compare the popcount result with #MAC. The resulting bit-1s define the range of the first #MAC NZWs. ❸ Perform element-wise AND operation between the original bitmask and the comparison result. It truncates the bitmask for the first #MAC NZWs and sets the remaining to zero for the excess NZWs. The truncated bitmask is then forwarded to the standard BFLY decoder. With the bitmask for the excess NZWs set to zero, BFLY will only route the first #MAC NZWs. ❹ Finally, perform element-wise XNOR operation between the truncated bitmask and the original bitmask. It updates the bitmask for the next SMD iteration. In summary, the four-step SMD decomposes arbitrary sparse weight matrix to multiple M:N sparse vectors, enabling effective PIM mapping using the BFLY decoding logic.

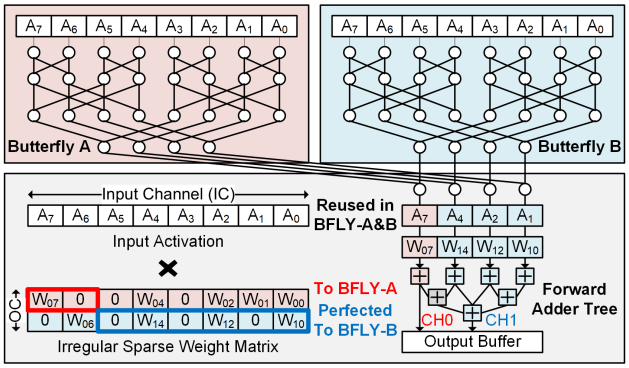


Fig. 5. Double-buffering BFLY to address low utilization issue.

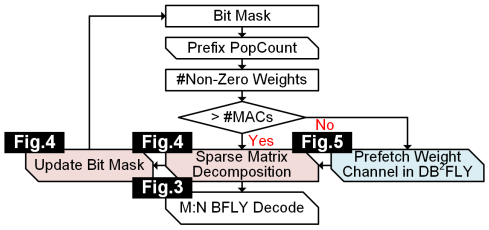


Fig. 6. The overall workflow of McPAL to map unstructured sparsity in PIM.

*2) **Double Buffering Butterfly Network**:* McPAL uses DB^{2}LFY to tackle the low utilization when handling highly sparse weight matrices. As shown in Fig. 5, the DB^{2}FLY comprises two independent regular BFLYs. If the sparse weight vector has enough NZWs to fill all MACs, only one BFLY is activated to reduce dynamic power. Otherwise, DB^{2}FLY activates both BFLYs to allocate adequate workload from a boarder region. Take the sparse weight matrix in Fig. 5 as an example. At the first cycle, McPAL employs SMD to produce M:N sparse compute for the first four NZWs, W_{00} to W_{04}. As a result, the remaining weight vector contains only one NZW, W_{07}, leading to insufficient data to fully utilize all MACs in the next iteration. To prevent low utilization, the DB^{2}FLY prefetches and then routes the NZWs from two consecutive output channels. Each BFLY handles M:N sparsity for its respective channel. In addition, both BFLYs reuse the same input activations, thereby reducing the input bandwidth requirement. The partial sums from these two channels cannot be accumulated directly. To resolve this issue, we leverage the forwarding adder tree in Sigma [17] for parallel accumulations within a single adder tree.

The integration of SMD and DB^{2}FLY enables McPAL to deal with unstructured sparsity in PIM. Fig. 6 summarizes the overall flow of the unstructured sparse computing. First, McPAL conducts prefix popcount on the input bitmask to calculate the number of NZWs and compares it with #MAC. As long as the NZWs can fill all MACs, SMD is then executed. Otherwise, the DB^{2}FLY prefetches the next weight channel to mitigate the low utilization. McPAL supports prefetching only two consecutive channels. That's because more channels would require a larger BFLY, forwarding adder tree and additional memory for partial sum storage, resulting in a higher hardware overhead. Although NZWs in these two channels may still fail to fully utilize the downstream MACs, the experimental results in Sec.V demonstrate that DB^{2}FLY is enough to achieve nearly a 2× latency reduction at 50% weight sparsity.

IV. Multi-Chiplet McPAL Scaling

A. 3D Vertical McPAL Scaling with Hierarchical HBM-PIM

To overcome the monolithic limitations of a single PIM chip, we first realize the sparsity-aware McPAL PIM and scale it vertically

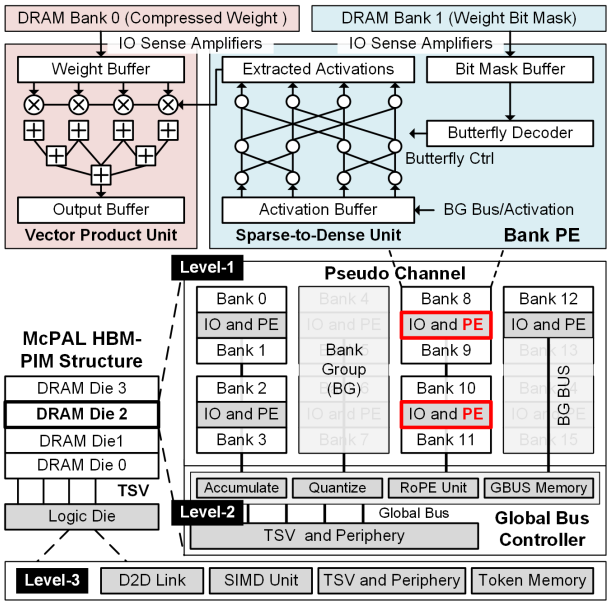


Fig. 7. 3D vertical McPAL scaling with hierarchical HBM-PIM.

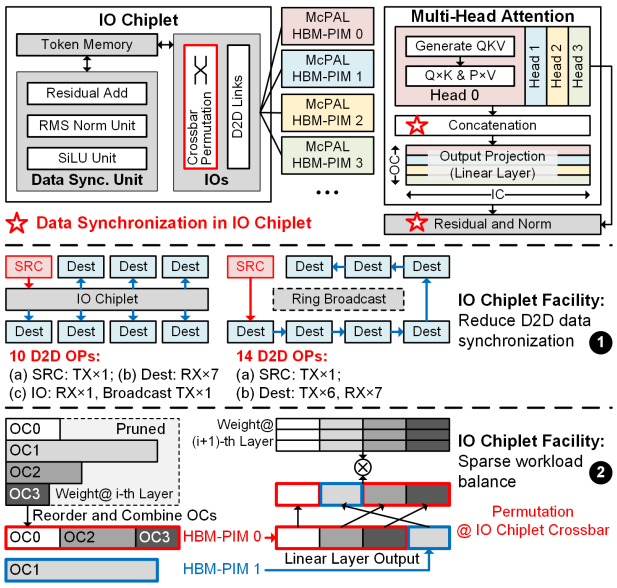


Fig. 8. 2.5D horizontal McPAL scaling with IO chiplet.

using 3D HBM technology. Fig. 7 gives the integration of the McPAL process engine (PE) into the HBM architecture. McPAL includes four DRAM dies and a logic die, all 3D stacked via TSVs. Each DRAM die contains four pseudo channels (pCH) that operate independently, with each pCH comprising four bank groups (BGs). BG is further divided into four banks and every two banks share one McPAL PE. Each McPAL PE can execute 32 INT-8 MACs in parallel, leveraging the 256-bit local bank I/O. Once McPAL issues a sparse PIM operation, the PE reads 32 NZWs and a 128-bit bitmask from two adjacent banks via I/O sense amplifiers, while activations are delivered from the logic die through TSVs, global bus and BG bus. The bitmask is decoded to control a 128×2-input-32-output DB^{2}FLY, which dispatches the required activations to NZWs. For circuit implementation, the embedded MACs and transmission-gate-based BFLY switches are derived from IMPERA [9].

FGDRAM [27] reveals that data transfer from BGs to the logic die via TSVs accounts for 56.6% of the HBM power consumption. To minimize long-distance travel via TSVs, we strategically organize the hardware hierarchy within HBMs to align with the distinct operational characteristics in LLMs. Besides the bank-level PEs, McPAL has two other hierarchical hardwares in the global bus controller (GBUS CTRL) of the DRAM die and the logic die. For linear layers, the activations from the logic die are broadcast to multiple BGs, which then process different weight output channels in parallel. For self-attention, the KV cache is stored in DRAM dies. Different BGs reuse the same Q to calculate Q×K or P×V for various KV vectors. This approach enables McPAL to fully leverage the abundant internal BGs of HBMs for vector-matrix multiplication, maximizing PIM utilization during the LLM decode phase. As shown in Fig. 7, the partial sums are accumulated and quantized in the GBUS CTRL per BG before writing back to the logic die through TSVs. It consumes less power as partial sums travel a shorter distance. The GBUS CTRL also includes a memory to buffer activations from the logic die and a RoPE unit for positional embedding. The activations are reused locally to avoid direct access to token memory in the logic die, while the RoPE unit locally embeds the positional information into the QK vectors. Then the prepared KV vectors are directly written back to the DRAM banks. Therefore, the GBUS CRTL further minimizes the data transfer through TSVs, improving the power efficiency of McPAL HBM. Finally, we place a 128-way SIMD in the logic die to compute softmax, where the exponential and division are approximated by Taylor expansion. That's because the softmax collects Q×K results from all BGs. Placing softmax at the GBUS or bank level would involve complex transfers among pCHs. Compared to HAIMA-like [11] HBM-PIM, which assigns matrix multiplication to the bank-level PE and allocates all other operations in the host logic die, the hierarchical McPAL reduces the power consumption by 34.3%.

B. 2.5D Horizontal McPAL Scaling with IO Chiplet

We further scale McPAL HBM-PIM horizontally to fit larger LLMs using an IO chiplet. The IO chiplet minimizes D2D communication during data synchronization and ensures balanced sparse workload across various McPAL devices. As shown in Fig. 8, the IO chiplet comprises a token memory, a synchronization unit and abundant IO functionalities. The token memory stores input tokens which are then dispatched to McPAL HBM-PIMs. The synchronization unit receives results from HBM-PIMs, and concurrently performs auxiliary operations in LLMs such as residual addition, RMS normalization or SiLU. Similarly, the non-linear functions are approximated by Taylor expansion. Following tensor parallel [20], McPAL divides self-attention by heads and linear layers by output channels across multiple McPAL HBM-PIMs. Each device independently processes the designated workload segment, contributing to a portion of the entire output. Therefore, output synchronization is essential for subsequent operations. For instance, the attention results from different heads must be concatenated before proceeding to the output projection. The synchronization is also required after each linear layer.

In contrast to ring broadcast [19], [20], the IO chiplet can minimize the D2D communication during data synchronization. Take 8 McPAL HBM-PIMs in Fig. 8 as an example. To move data from one source to the others, HBM-PIMs issue one transmission (TX from source) and 7 receptions (RX to destinations), while the IO chiplet receives data from the source and broadcasts it to the destinations. Consequently, the IO chiplet employs 7 distinct RX channels, which encompass the physical layer, D2D adapter and protocol layer, while reusing a single TX channel. Therefore, a total of 10 D2D transmissions are required. In contrast, the ring broadcast necessitates 7 TX and 7 RX transmissions, 1.4× higher than McPAL. In addition, the IO chiplet enables workload balance across different McPAL HBM-PIMs. Due

TABLE I
HBM CHARACTERISTICS.

HBM Organization	DRAM Die=4, pCHs/Die=4, Banks/pCH=16, Rows/Bank=32k, Row Size=1KB, IOs/pCH=64bits
HBM Timing (ns)	t_{RC}=45, t_{RCD}=16, t_{RAS}=29, t_{CL}=16, t_{RRD}=2, t_{WR}=16, t_{CCD}=2
HBM Energy (pJ)	e_{ACT}=909, $e_{Pre-GSA}$=1.51, $e_{Post-GSA}$=1.17, $e_{I/O}$=0.80

TABLE II
POWER AND AREA CHARACTERISTICS OF MCPAL.

	Module	Parameter	Power (mW)	Area (mm^2)
Bank PE	DB^{2}FLY	128×2-Input, 32-Output	4.00	27080.34
	Multiplier	INT-8 ×32	5.60	33195.99
	Adder Tree	×1	3.21	19294.95
	Output Buffer	32b×128	3.68	17588.08
GBUS CTRL	Accumulator	INT-16 ×16	5.46	29273.13
	Quantization	×16	14.98	82532.26
	RoPE MAC	INT-16 ×16	11.46	67967.75
	GBUS Memory	64KB ×16	27.23	347042.07
Logic Die	SIMD	INT-8 MACs ×128	35.25	59942.40
	Token Memory	16MB SRAM	108.94	27423405.82
	D2D Link	256Gb/s UCIE	195.94	470831.04
IO Chiplet	SIMD	INT-8 MACs ×128	35.25	59942.40
	Token Memory	16MB SRAM	108.94	27423405.82
	D2D Link	256Gb/s UCIE ×8	1567.48	3766648.32

to the unstructured sparsity within McPAL, directly distributing the linear layers to various McPAL devices could result in unbalanced workloads. The HBM-PIM, which is assigned the heaviest workload, dominates the total latency, leading to low utilization of both PIM logic and D2D bandwidth. To address this issue, as shown in Fig. 8, we combine several highly sparse output channels and assign them to McPAL-0, while a single dense channel is assigned to McPAL-1. Note that the reordered outputs are misaligned and thus cannot directly multiply with the weight in the next layer. The IO chiplet utilizes a crossbar that permutes the output during data synchronization, ensuring correct LLM execution.

V. EXPERIMENTAL RESULTS

We developed a McPAL prototype using 28nm technology. The 8GB HBM used in McPAL follows the standard HBM2 protocol. Its hardware characteristic is drawn from FGDRAM [27] and summarized in Table. I. The embedded multiplier and transmission-gate BFLY switches are derived from IMPERA [9], and redesigned with Cadence Virtuoso. The digital modules in McPAL PE and logic die are synthesized using Synopsys Design Compiler and then placed and routed with Synopsys IC Compiler. The post layout provides area and power estimations. Similar to TransPIM [19], we consider the process difference between the CMOS logic and DRAM, where the DRAM process incurs about 50% area overhead to the logic process. We evaluate McPAL across various Llama models (7B, 33B and 70B). To facilitate McPAL execution, these models are pruned to 50% sparsity by Wanda [5] and quantized to INT-8 by SmoothQuant [7]. The perplexities on the WikiText dataset are 7.45, 5.86 and 4.07, respectively, resulting in perplexity losses of 1.77, 1.09 and 0.95 compared to the original models. We run Llama models on Nvidia A100 40GB GPU, serving as the baseline and compare with McPAL. To fit the entire parameters on McPAL, we utilize 1, 4 and 8 McPAL HBM-PIMs to run Llmam-7B, 33B and 70B models, respectively. Finally, **McPAL-S** enables sparse compute, whereas **McPAL-D** stands for the dense counterpart without sparsity.

A. McPAL Performance and Design Space Exploration

*1) **Power, Area and Latency Breakdown**:* Table II reports McPAL specification. To match the column access time of HBM ($t_{CCD} = 2ns$), McPAL PEs and other digital modules run at 500MHz frequency and 1.0V power supply. As described in Sec. IV, every

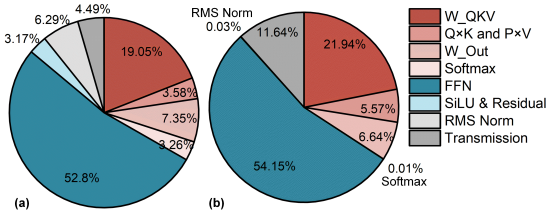


Fig. 9. (a) Latency and (b) Energy breakdown of McPAL with 8 McPAL HBM-PIMs on Llama-70B model.

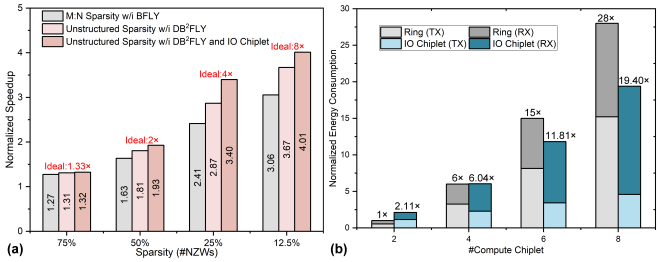


Fig. 10. Design space exploration: (a) The impact of unstructured weight sparsity on McPAL performance, and (b) Transmission energy reduction in IO-chiplet-based McPAL compared to ring broadcast.

two DRAM banks share a single 32-MAC McPAL PE. Therefore, the total of 32 PEs and the GBUS CTRL occupy a 3.64mm^2 area, incurring 6.8% overhead to the original HBM die (53.15mm^2) [19]. As shown in Table I, each pCH provides a 64-bit data bus to transfer 16-bit partial sums. To avoid congestion, only half of the McPAL PEs are activated at the same time. As a result, a single McPAL HBM-PIM yields a peak throughput of 2.048TOPs. Both the logic die and IO chiplet contain a 16MB token memory, capable of storing up to 2K input tokens for the Llama-70B model. Finally, the D2D link adheres to the standard UCIe protocol [26], providing a total transmission speed of 256Gb/s with 16 lanes. Fig. 9 gives the latency and energy breakdown of running the Llama-70B model on 8 McPAL HBM-PIMs with sparsity enabled. The model processes an input sequence of 2K input tokens and sequentially generates an additional 2K tokens. The matrix multiplication in HBM-PIM dominates the inference, accounting for 82.78% latency and 88.31% energy cost. Consequently, the average throughput is 13.56TOPS (82.78%×2.048TOPS×8 McPAL HBM-PIMs). The IO chiplet orchestration reduces D2D communication, which only contributes to 4.49% latency and 11.64% energy consumption. The remainder is for the non-linear functions.

*2) **McPAL Speedup with Sparsity**:* Fig. 10(a) shows the impact of unstructured weight sparsity on McPAL performance, with comparison between BFLY and DB^{2}FLY. In this experiment, we exclusively compute the feed-forward network of the Llama-70B model. With a lower sparsity where 75% of NZWs are reserved, the BFLY and DB^{2}FLY shows similar speedup. At a higher sparsity of 50%, BFLY fails to fully exploit the sparsity benefit, resulting in only 1.63× speedup. In contrast, DB^{2}FLY reduces the data congestion during sparsity decoding and achieves 1.81× speedup. The IO chiplet further enables sparse workload balance across various McPAL HBM-PIMs. As a result, the ultimate McPAL achieves 1.93× speedup, approaching the 2× speedup upper bound. It even achieves 3.40× speedup with only 25% of NZWs preserved. Finally, McPAL struggles to maintain utilization at higher sparsity. Nevertheless, McPAL well supports 50% sparsity, which is the highest compression ratio attainable by the prevalent sparsification algorithms [5], [6] without compromising LLM accuracy.

*3) **IO Chiplet Efficiency**:* We evaluate the efficiency of the IO chiplet for data synchronization and compare it with a baseline,

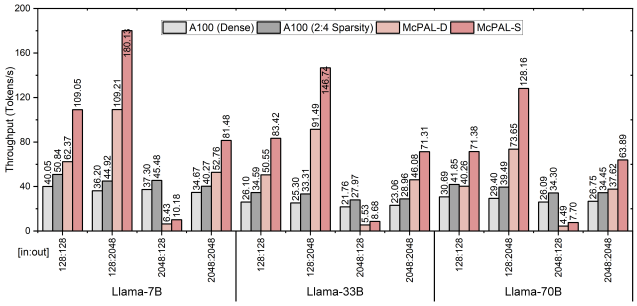


Fig. 11. McPAL throughput improvement over Nvidia A100 GPU.

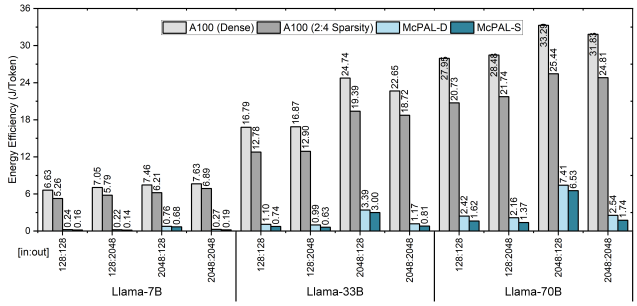


Fig. 12. McPAL energy efficiency improvement over Nvidia A100 GPU.

where McPAL HBM-PIMs are organized as a ring [20]. To fit all parameters on fewer McPAL devices, a smaller Llama-7B model is used to generate 2K tokens with another 2K input tokens. Fig. 10(b) reports the normalized energy cost of D2D communication. The ring topology exhibits better energy efficiency when configured with 2 or 4 McPAL HBM-PIMs. It indicates that direct communication among HBM-PIMs is more efficient in such cases. Nevertheless, the IO chiplet proves to be more efficient for coordinating more McPAL HBM-PIMs, making it more compatible with LLM evolution towards large model sizes. Specifically, the IO chiplet consumes 1.27× less energy to coordinate 6 McPAL HBM-PIMs and 1.44× less energy for 8 McPAL HBM-PIMs. As described in Sec. IV-B, the I/O chiplet broadcasts data to all destinations simultaneously, thereby reducing TX transmission requirement with a minimal RX transmission overhead.

B. Comparison to Baseline

1) ***Comparison to Nvidia A100 GPU****:* We first compare McPAL with the Nvidia A100 GPU. To accommodate the entire model on devices, we use 1, 4 and 8 8GB HBM-PIMs to run Llama-7B, 33B and 70B models, respectively, while utilizing 1, 2 and 4 A100 GPUs to run the same models. Even using fewer GPUs, the peak TOPS of a single A100 GPU remains 76.2× higher than that of 8 McPAL HBM-PIMs. Fig. 11 and Fig. 12 show McPAL's improvements in throughput and energy efficiency across various prefill and decode sequence lengths, [in:out]. For throughput comparison, the latency of both prefill and decode phases are considered and divided by the number of output tokens. The approach to calculate energy efficiency is similar. Compared to the A100 GPU, McPAL-D shows an average speedup of 1.57×, 1.95× and 1.35× on Llama-7B, 33B and 70B models, respectively. It also achieves 24.50×, 14.75× and 10.43× energy efficiency. Besides the inherent advantages of HBM-PIM, the proposed hierarchical multi-chiplet scaling method ensures >80% utilization under various workloads, while the A100 GPU achieves only about 3.2% utilization under the same scenarios. By pruning the weights to 50% sparsity, McPAL-S results in further speedup of 2.58×, 3.12× and 2.34×, along with energy efficiency improvements of 35.66×, 21.50× and 15.34×. For a fair comparison, we also

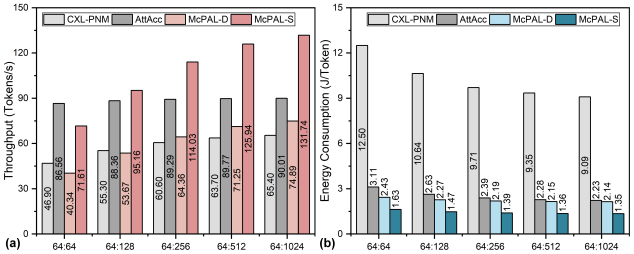


Fig. 13. Comparison with CXL-PNM and AttAcc: (a) Throughput and (b) Energy Consumption. McPAL and AttAcc run Llama-70B model and CXL-PNM runs OPT-65B model.

explore 2:4 sparsity (described in Sec. III-A) on A100 GPUs, which is inherently supported by Nvidia sparse tensor core. Nevertheless, McPAL-S still achieves 2.10×, 2.39× and 1.76× average speedup, and 29.86×, 16.90× and 11.69× energy efficiency at the same sparsity. Finally, McPAL shows better performance with longer input sequences, but worse performance with fewer input tokens. Take Llama-70B model as an example. Compared to A100 GPU, McPAL-S achieves 3.24× speedup and 15.88× energy efficiency when [in:out] is [128:2048]. That's because the memory-bound decode stage exacerbates the overhead caused by the back-and-forth weight movement in GPUs. In contrast, McPAL-S demonstrates only 0.21× throughput and 3.89× energy efficiency with [2048:128] workload, where the compute-bound prefill phase saturates GPU utilization.

2) ***Comparison to SOTA LLM Accelerators****:* We also compare McPAL with SOTAs LLM accelerators. CXL-PNM [29] is a near-memory architecture based on LPDDR5, utilizing 8 devices to run the OPT-65B model. Each device delivers the same peak throughput as one McPAL HBM-PIM. AttAcc [30] is a hybrid attention accelerator for the Llama-70B model, where A100 GPU handles the linear layer and HBM-PIM processes the self-attention. Fig. 13 compares the performance of McPAL HBM-PIM with CXL-PNM and AttAcc. AttAcc's results are obtained from its open-sourced simulator, while CXL-PNM provides explicit performance. McPAL-D achieves a similar throughput as CXL-PNM, but delivers 4.25× to 5.14× energy efficiency. In addition, McPAL-S further achieves 1.53× to 2.01× speedup and 6.75× to 7.68× energy efficiency. AttAcc demonstrates a speedup of 1.20× to 2.15× over McPAL-D across all workloads, and even a speedup of 1.21× over McPAL-S under [64:64] workload. That's because AttAcc allocates more compute resources, using A100 GPUs and 40 HBM3-PIMs to process LLMs. Nevertheless, McPAL-S still achieves 1.08× to 1.46× speedup with very few compute logic, alongside energy efficiency improvements of 1.65× to 1.91×. The data communication of self-attention between A100 GPU and HBM-PIM negatively impacts the energy efficiency of AttAcc, accounting for about 43.3% of energy consumption. In contrast, McPAL eliminates the need for inter-chip transfer for the KV cache by the three-level hierarchical HBM-PIM design.

VI. Conclusion

The paper introduces McPAL, a novel HBM-PIM-chiplet architecture that enhances sparse compute and multi-chiplet scalability for LLM inference. The sparse matrix decomposition enables unstructured sparse compute in PIM without compromising LLM accuracy or hardware utilization. Further, the chiplet scaling reduces D2D data transferor and enables sparse workload balance. The experimental results show that McPAL achieves 1.57× to 3.12× speedup and 10.43× to 35.66× energy efficiency over Nvidia A100 GPU. Compared to SOTA accelerators, McPAL also achieves 1.08× to 2.15× speedup and 1.65× to 5.14× energy efficiency.

REFERENCES

[1] A. Radford, et al. "Language models are unsupervised multitask learners." OpenAI blog, 2019.

[2] J. Achiam, et al. "Gpt-4 technical report." arXiv preprint arXiv:2303.08774, 2023.

[3] H. Touvron, et al. "Llama 2: Open foundation and fine-tuned chat models." arXiv preprint arXiv:2307.09288, 2023.

[4] A. Gholami, et al. "AI and memory wall." IEEE Micro, 2024.

[5] M. Sun, et al. "A simple and effective pruning approach for large language models." arXiv preprint arXiv:2306.11695, 2023.

[6] E. Frantar and A. Dan. "Sparsegpt: Massive language models can be accurately pruned in one-shot." International Conference on Machine Learning, 2023.

[7] G. Xiao, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." International Conference on Machine Learning (ICML), 2023.

[8] M. Zhang, et al. "FNM-Trans: Efficient FPGA-based Transformer Architecture with Full N: M Sparsity." ACM/IEEE Design Automation Conference (DAC), 2024.

[9] S. Liu, et al. "A 28nm 53.8 TOPS/W 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned NN and CIM-based local-attention-reusable engine." IEEE International Solid-State Circuits Conference (ISSCC), 2023.

[10] X. Yang, et al. "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration." International Conference on Computer-Aided Design (ICCAD), 2020.

[11] Y. Ding, et al. "HAIMA: A hybrid SRAM and DRAM accelerator-in-memory architecture for transformer." ACM/IEEE Design Automation Conference (DAC), 2023.

[12] Z. Chen, et al. "An in-memory computing accelerator with reconfigurable dataflow for multi-scale vision transformer with hybrid topology." ACM/IEEE Design Automation Conference (DAC), 2024.

[13] F. Tu, et al. "TranCIM: Full-digital bitline-transpose CIM-based sparse transformer accelerator with pipeline/parallel reconfigurable modes." IEEE Journal of Solid-State Circuits (JSSC), 2022.

[14] F. Tu, et al. "Multcim: Digital computing-in-memory-based multimodal transformer accelerator with attention-token-bit hybrid sparsity." IEEE Journal of Solid-State Circuits (JSSC), 2023.

[15] A.F. Laguna, et al. "In-memory computing based accelerator for transformer networks for long sequences." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021.

[16] G. Li et al. "Ristretto: An atomized processing architecture for sparsity-condensed stream flow in cnn." IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

[17] E. Qin, et al. "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training." IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020.

[18] Z. Li, et al. "Spada: Accelerating sparse matrix multiplication with adaptive dataflow." ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2023.

[19] M. Zhou, et al. "Transpim: A memory-based acceleration via software-hardware co-design for transformer." IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2022.

[20] S. Hong, et al. "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation." IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

[21] J. Yue, et al. "An energy-efficient computing-in-memory NN processor with set-associate blockwise sparsity and ping-pong weight update." IEEE Journal of Solid-State Circuits (JSSC), 2023.

[22] C. Chu, et al. "PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture." ACM/IEEE Design Automation Conference (DAC), 2020.

[23] J. Kim, et al. "Z-PIM: A sparsity-aware processing-in-memory architecture with fully variable weight bit-precision for energy-efficient deep neural networks." IEEE Journal of Solid-State Circuits (JSSC), 2021.

[24] J. Cai, et al. "Gemini: Mapping and architecture co-exploration for large-scale DNN chiplet accelerators." IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2024.

[25] Y.S. Shao, et al. "Simba: Scaling deep-learning inference with multi-chip-module-based architecture." IEEE/ACM International Symposium on Microarchitecture (MICRO). 2019.

[26] D. Sharma, et al. "Universal chiplet interconnect express (UCIe): An open industry standard for innovations with chiplets at package level." IEEE Transactions on Components, Packaging and Manufacturing Technology, 2022.

[27] M. O'Connor, et al. "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems." IEEE/ACM International Symposium on Microarchitecture (MICRO), 2017.

[28] Y. Jin et al. "S3: Increasing GPU utilization during generative inference for higher throughput." Advances in Neural Information Processing Systems (NIPS), 2023.

[29] S. Park, et al. "An LPDDR-based CXL-PNM platform for TCO-efficient inference of transformer-based large language models." IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2024.

[30] J. Park, et al. "AttAcc! Unleashing the power of PIM for batched transformer-based generative model inference." ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2024.

EdgeMM: Multi-Core CPU with Heterogeneous AI-Extension and Activation-aware Weight Pruning for Multimodal LLMs at Edge

Kangbo Bai
School of Integrated Circuits
Peking University
Beijing, China
baikangbo@pku.edu.cn

Le Ye
School of Integrated Circuits
Peking University
Beijing, China
yele@pku.edu.cn

Ru Huang
School of Integrated Circuits
Peking University
Beijing China
ruhuang@pku.edu.cn

Tianyu Jia*
School of Integrated Circuits
Peking University
Beijing, China
tianyuj@pku.edu.cn

***Abstract*—Emerging multimodal LLMs (MLLMs) exhibit strong cross-modality perception and reasoning capabilities and hold great potential for various applications at edge. However, MLLMs typically consist of a compute-intensive modality encoder and a memory-bound LLM decoder, leading to distinct bottlenecks for hardware designs. In this work, we present a multi-core CPU solution with heterogeneous AI extensions, which are based on either the compute-centric systolic array or memory-centric digital compute-in-memory (CIM) coprocessors. In addition, dynamic activation-aware weight pruning and bandwidth management are developed to enhance bandwidth efficiency and core utilization, improving overall performance. We implemented our solution using commercial 22nm technology. For representative MLLMs, our evaluations show EdgeMM can achieve 2.84× performance speedup compared to laptop 3060 GPU.**

***Index Terms*—Multimodal Large Language Model, Multi-Core CPU, Heterogeneous Architecture, CPU Extension.**

I. Introduction

Multimodal large language models (MLLMs) exhibit strong perception, reasoning, and planning abilities, with promising applications in autonomous driving [1], embodied AI [2], mobile [3], and AR/VR. As shown in Fig. 1 (a), MLLMs take in various modalities of input, such as image, audio, and LiDAR, which are processed by modality encoder to generate token representations. Following LLM and post-processing are performed for tasks such as comprehension, object segmentation [4], or trajectory planning [5]. Due to the application demands for privacy and real-time responses, deploying MLLMs on edge devices is highly desired.

Achieving low-latency inference of MLLMs at edge is challenging. First, the hardware resource is quite constrained at edge. Second, different phases in MLLMs exhibit opposite characteristics and performance bottlenecks. For instance, the encoder network is compute-intensive, while the LLM-decoding phase is bounded by the memory bandwidth. Such workload behaviors limit performance improvements of certain accelerators tailored for particular Transformers [6], [7]. Meanwhile, the prior optimization techniques for long-context

*Corresponding author.

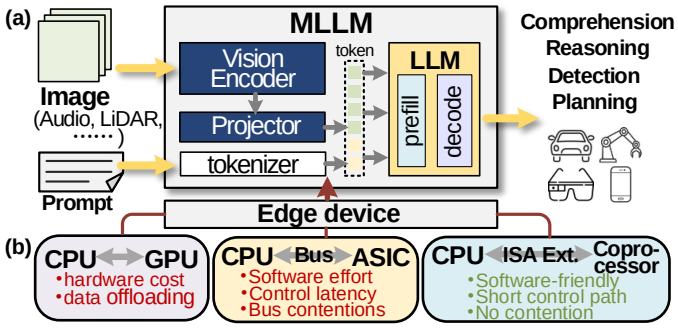


Fig. 1. MLLMs and hardware architecture design choices.

LLMs are not suitable for edge MLLMs, which process short token sequences with streaming inputs.

There are multiple hardware architecture choices for edge devices, as shown in Fig. 1(b). CPU remains the most common hardware in edge devices, while it has limited parallelism for AI tasks. The use of GPU on edge devices is severely limited by cost, and data offloading from CPU to GPU may become a system bottleneck [8]. ASIC accelerator is also a popular solution [9], which is connected to host CPU through memory-mapped interface. However, ASIC accelerators require corresponding compiler development, adding huge engineering effort for software developers. They also introduce bus access with increased control latency and bus contention [10].

In recent years, AI-target ISA extension with in-CPU coprocessors has emerged as a promising solution for AI tasks. For example, Intel introduced its advanced matrix extensions (AMX) [11] in the Xeon processors [12], [13]. Arm developed scalable matrix extension (SME) in Armv9-A [14]. T-head proposed open-source RISC-V matrix extension [15]. At architecture level, the host CPU cores decode and dispatch these extended instructions to coprocessors for execution via direct-linked interfaces. Compared to conventional bus-attached accelerators, coprocessors have a shorter control path and avoid bus contention. Moreover, this architecture can be merged into existing CPU ISA and inherit the compiler stack [13].

In this work, we explore a multi-core CPU solution for

TABLE I
REPRESENTATIVE MLLMS AND EFFICIENT EDGE MLLMS.

MLLMs	Visual Encoder	Projector	Language Model
Emu2-Chat [16]	EVA [17]	MLP	LLaMa (33B)
LLaVA [18]	CLIP ViT/L-14 [19] (0.3B)	MLP	Vicuna [20] (7B, 13B)
MobileVLM [21]	CLIP ViT/L-14 [19] (0.3B)	LDP	MobileLLaMA [21] (2.7B)
TinyGPT-V [22]	EVA	Q-former [23]	Phi-2 [24] (2.7B)
SPHINX-Tiny [25]	CLIP ViT/L-14 [19], CLIP ConvNeXt [19] + DINOv2(0.4B) [26]	MLP	TinyLlama [27] (1.1B)
DeepSeek-VL [28]	SigLIP-L [29] (0.4B)	MLP	DeepSeek-LLM [28] (1.3B)
Karmavlm [30]	SigLIP-so [29] (0.4B), CLIP ViT/L-14 [19] (0.3B)	MLP	Qwen1.5 [31] (0.5B)

edge MLLMs with heterogeneous AI-extension. For compute-intensive general matrix multiplication (GEMM), systolic array is utilized as the coprocessor. Alongside, for memory-bound matrix-vector multiplication (GEMV), which dominates LLM-decoding phase, area- and power-efficient digital compute-in-memory (CIM) macros are adopted. These coprocessors have different dataflows and resource distributions, coupled with RISC-V ISA and programming model for easy deployment. Furthermore, a dynamic activation-aware weight pruning is developed along with hardware support to alleviate memory bandwidth pressure. At the system level, the DRAM bandwidth can be dynamically managed to fully utilize heterogeneous cores for diverse sequence lengths.

The contributions of this paper are summarized as follows:

- We analyze the attributes of edge MLLMs, revealing that different phases encounter distinct bottlenecks in computation and memory access.
- We develop the heterogeneous extension for multi-core CPU, employing the systolic array and CIM macro tailored for GEMM and GEMV.
- We present an activation-aware weight pruning and bandwidth management, leveraging the limited bandwidth and heterogeneous cores to accelerate edge MLLM.
- EdgeMM is implemented and evaluated at 22nm, achieving a 0.217 token/J efficiency and 138 tokens/s throughput, which is 2.84× better than RTX 3060 GPU.

II. BACKGROUND AND MOTIVATIONS

A. Multimodal Large Language Models

MLLM builds on standard LLM by incorporating multimodal data for input and output. In recent two years, leading MLLMs such as Gemini [32], GPT-V [33], LLaVA [18], InstructBLIP [34] have emerged, impacting both industry and academia. Combining LLMs' reasoning with multimodal perception, they exhibit remarkable performance.

Fig. 1(a) illustrates the typical backbone architecture of MLLM, which is composed of modality encoder, projector, and LLM. Take visual modality as an example. The features are extracted from input images by Transformer-based vision encoder, e.g. CLIP [19], DINOv2 [26] and then aligned with language model via a projector, e.g. MLP, Q-former [23], to generate vision tokens. The vision tokens and text tokens from the prompt are jointly fed into the LLM. The inference of LLM consists of two phases, i.e. prefill and decoding. In the prefill phase, attention is applied between all tokens, generating the first output token and key-value (KV) cache. In the decoding phase, only the last token is taken as input, autoregressively generating the next token and updating the KV cache. The

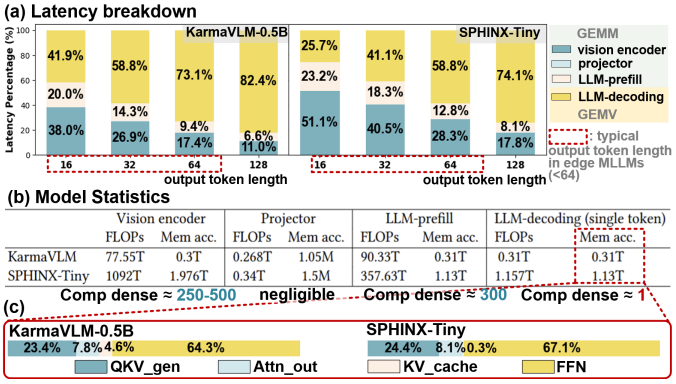


Fig. 2. Workload analysis of two MLLMs: (a) latency breakdown, (b) model statistics, (c) memory accesses.

generated tokens can be converted into language as output or further processed by downstream tasks, e.g. AD [1].

Tab. I lists a few representative open-source MLLMs and their model architectures. MLLMs [16], [18] often adopt LLMs with parameter amounts surpassing 7B. Meanwhile, there is also a model development trend for lightweight MLLMs to assist the deployment at edge. These edge MLLMs adopt compressed LLMs with less than 3B parameters, such as Phi2-2.7B [24], TinyLlama-1.1B [27], Qwen1.5-0.5B [31]. It is also worth pointing out that these edge MLLMs also demonstrate comparable performance compared to large-scale MLLMs in many benchmarks [35]–[38].

B. Model Profiling and Analysis

To better understand the workload characteristics, we conduct profiling on two representative MLLMs: SPHINX-Tiny [25] and Karmavlm [30]. The input token length is about 300, primarily made up of vision tokens. The inference latency breakdown on Nvidia 3060 GPU with varying output token lengths is shown in Fig. 2(a). An increase in output tokens leads to more LLM-decoding iterations, enlarging the proportion of this phase in overall latency. Alongside, the vision encoder and LLM-prefill also contribute to the overall latency notably in edge applications with short token lengths. The latency of the projector is negligible.

Fig. 2(b) also indicates the computing and memory attributes differ significantly during different phases. In vision encoder and LLM-prefill, multiple token embedding vectors share weight matrices forming computation-intensive GEMM. Meanwhile, LLM-decoding performs the operations between the entire weight matrix and a single embedding vector, resulting in low-computational-density GEMV. The LLM-decoding employs the same model parameters as LLM-prefill but involves two orders of magnitude fewer FLOPs, revealing an obvious memory-bound bottleneck. As shown in Fig. 2(c), weight matrices dominate the memory access, with FFN consuming the largest portion as the channel dimension is typically several times larger than the model dimension [39]. The KV cache only contributes a small portion of memory access due to the short token length in edge MLLMs.

Since the majority of DRAM access is occupied by FFN weight matrices, we further analyze the characteristics of this

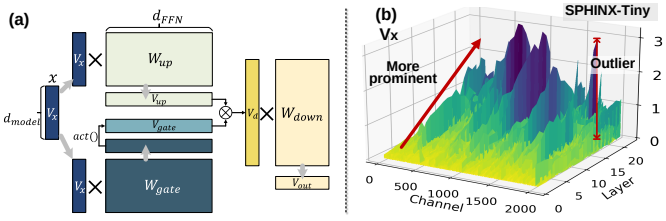


Fig. 3. Gated-MLP and the activation vector sparsity in FFN.

portion, e.g. sparsity. Currently the mainstream LLMs, e.g. Llama series [40], Mistral [41], Qwen [31], use the gated-MLP as FFN. Fig 3 (a) shows the detailed process, which contains three GEMV, and the calculation formula is as follows:

$$FFN(V_x) = ((V_x * W_{up}) \circ (act(V_x * W_{gate}))) * W_{down} \quad (1)$$

where V_x is the input vector, W_{up}, W_{gate} are $d_{FFN} \times d_{model}$ weight matrices. $Act()$ is the activation function. In fact, the activation vectors (V_x, V_d) in GEMV exhibit notable sparsity across channels. Fig. 3(b) shows profiled magnitudes of V_x during a token generation in SPHINX-Tiny [25]. The x, y, and z axes denote the decoder layer index, vector channels, and absolute magnitudes. As presented, these activation vectors exhibit notable sparsity for most channels, with a few outlier channels that can be easily masked out. Moreover, as the layer index increases, these outliers become more prominent.

The above analysis guides us to rethink the optimal architecture for edge MLLMs with the following design insights. **Insight 1: Heterogeneous coprocessors for GEMM and GEMV are desired.** The performance of compute-bound GEMM can be enhanced by increasing computing resources, while memory-bound GEMV requires addressing bandwidth. **Insight 2: Activation-aware weights pruning is promising.** Many prior work focuses on long-context KV cache optimization [42]. For short-context edge MLLMs, the weight matrices of FFN are key constraints, while the sparsity in most activation channels offers great potential for optimization. **Insight 3: Scheduling for output token lengths is needed.** Since the workload proportion varies with the output token length, dynamic scheduling is also strongly desired to maintain the balanced workload latency of heterogeneous cores.

III. EdgeMM Architecture Design

A. Architecture Overview

To accommodate the mentioned characteristics of edge MLLMs, we present a multi-core CPU architecture, i.e. EdgeMM, with heterogeneous AI extensions, as shown in Fig. 4. EdgeMM adopts a hierarchical multi-core architecture based on the open-source Snitch_cluster [43]. Two types of accelerator architecture are incorporated as coprocessors for compute-centric (CC) cores and memory-centric (MC) cores. Specifically, CC-cores utilize compute-dense systolic array (SA)-based coprocessors for GEMM. While MC-cores utilize memory-abundant digital CIM, which integrates the compute units within SRAM banks, as coprocessors for GEMV.

For CC-cores, cores in a cluster share the instruction memory and data memory. In contrast, the data memory and

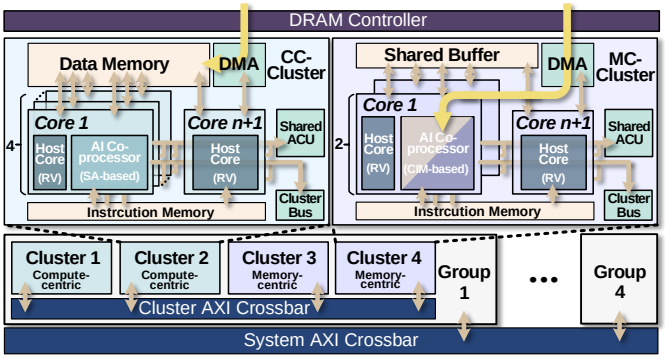


Fig. 4. Architecture overview of EdgeMM design.

compute array in MC-cores are integrated as a CIM macro in each core, and a small shared buffer is equipped for inter-core data transfer. Every cluster has a distributed DMA module, all of which are linked to the DRAM controller for data transfer to on-chip data memory. Auxiliary compute units (ACU), e.g. 32-bit multiplier and divider, are also shared among cores in a cluster for uncommon calculations.

The entire chip contains 4 groups, each consisting of 2 CC-clusters and 2 MC-clusters. Each CC-cluster and MC-cluster contains 4 CC-cores and 2 MC-cores respectively. These cores integrate an area-efficient RISC-V host core [43] for control and an extended coprocessor for AI computing. Each cluster also has a dedicated host core to control the DMA module. The groups and clusters are connected by the hierarchical AXI crossbars. The hardware architecture can also be scaled by changing architecture parameters.

B. CPU Extension for AI Acceleration

Fig. 5 and Fig. 6 illustrate the design details of CC and MC cores. Both core types consist of a host core and a coprocessor. The extended instructions are decoded by host core and dispatched to coprocessor via direct-linked interface.

Compute-Centric core: As shown in Fig. 5, the bulk of compute-centric GEMM coprocessor is a systolic array, which contains $R \times C$ multiply-accumulate processing elements (PE). For GEMM operations, the weights stored in the PE array remain stationary, while the input activations are sent into the array in a systolic manner. Four $R \times C$ matrix registers are equipped to store operands. The vector units are employed to execute vector instructions for element-wise computations. Vector instructions share the matrix registers and have an element width of C, enabling parallel operation on a row of a matrix register by one instruction. The coprocessor has an independent load/store unit for matrix registers, avoiding the data bottleneck caused by the narrow port of the host core. For multiplication between a $R \times C$ and an $M \times R$ matrix, the cycles of loading and computing equals:

$$L_{SA} = R + (R-1) + (C+M-1) - 1 = 2R + C + M - 3 \quad (2)$$

Systolic dataflow enables data transfer solely between neighboring PEs, reducing wiring overhead. However, there is only one column of activation (circled in red in Fig. 5) in GEMV, causing PEs idleness and inefficiency.

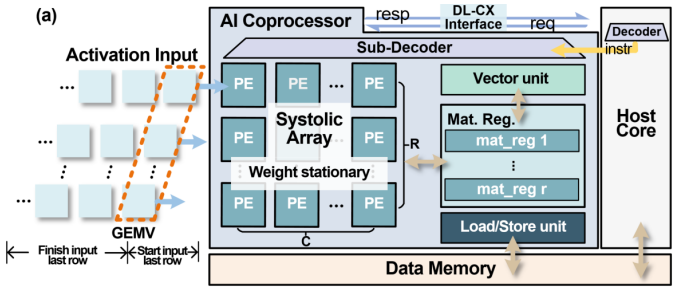


Fig. 5. Compute-centric core with Systolic Array.

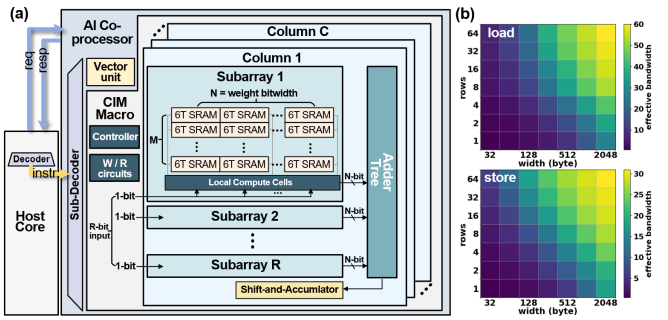


Fig. 6. Mem-centric core with CIM macro design.

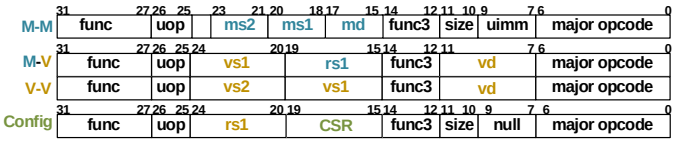


Fig. 7. Instruction format of the extended instructions.

Memory-Centric core: As shown in Fig. 6, the main feature of an MC-core is utilizing digital CIM for the coprocessor, which integrates compute cells within SRAM macro to avoid reg-to-SRAM load/store. A CIM macro consists of a controller, wordline decoder, write-and-read circuits, and C columns. Each column contains R subarrays, an adder tree, and a shift-and-accumulator. Each subarray consists of $M \times N$ 6T SRAM bit-cells. Here N equals to the bit-width of a weight data. During GEMV, the W-bit activation data is broadcasted into the columns in bit-serial. One of the weights in each subarray is read out and multiplied with 1-bit activation every cycle. Adder tree sums the R product and sends the result to shift-and-accumulator, in which the sum is shifted and accumulated to finish the entire W-bit multiplication. The broadcast dataflow fully utilizes the compute cells for GEMV, which is completed in $W+1$ cycles. The cycle number of a M rows GEMM is:

$$L_{CIM} = M * W + 1 \tag{3}$$

Compared to SA, CIM requires fewer cycles for GEMV, i.e. $M = 1$ in (2)(3). But for GEMM, CIM is less efficient due to the bit-width factor W. Moreover, MC-clusters have significantly larger data memory than CC-clusters, which allows larger matrix block to be transferred at once. We assess the effective bandwidth (data amount/cycles) for transferring matrices with various sizes. Fig. 6 (b) shows that the effective bandwidth drops notably for small matrices, but nears the ideal bandwidth as matrix size increases. This indicates the ample on-chip memory in MC-cluster can alleviate the bandwidth pressure and enhance DMA and DRAM efficiency.

C. ISA Extension and Programming Model

We adopt RISC-V as our main ISA and extend instructions for AI acceleration as shown in Fig 7. The first row illustrates the matrix instruction (i.e. mm_mul for GEMM) supported by CC-core. The matrix instructions differ from scalar instructions by employing matrix registers for both the source and destination operands. The CIM-based MC-core supports matrix-vector instructions, in which the *vs1* and *vd* are vector registers that store source and destination operands, and the *rs1* store the base address of matrix operand. The vector units in all cores support a subset of RISC-V vector instructions for activation functions and data precision conversion. Configure instructions can modify the control and status registers (CSR), storing the runtime parameters such as vector/matrix sizes.

The programming model of our EdgeMM is similar to snich_cluster [43]. The computing tasks, e.g. GEMM/GEMV, are allocated across cores with tensor partitioning. Specifically, each core and cluster contains read-only CSRs that store its index and type, allowing cores to calculate address offsets to access the allocated tensor shards. The core synchronization function is also supported. For ISA extension, the extended instructions can be utilized by customized kernel functions, enabling the use of the RISC-V toolchain without the internal modification of the compiler.

IV. Bandwidth Optimizations

To effectively utilize the limited bandwidth, we optimize memory access reduction by activation-aware weight pruning and perform token-length-driven bandwidth management.

A. Activation-aware Weight Pruning for GEMV

As Sec. II-B mentioned, the FFN weight matrices dominate DRAM access, and the activation vectors exhibit significant sparsity and outliers across channels. This motivates us to perform channel-wise activation-aware pruning. As shown in Fig. 8, GEMV can be performed by selecting the Top-k outliers and pruning the rest channels with corresponding rows in weight matrices. Recent works [44], [45] adopt the similar concept, but they all regard k as a fixed empirical input. In fact, k can vary along different decoder layers for better pruning. For example, early decoder layers should retain most weights to avoid errors accumulating and the latter layers have more distinct outliers, allowing for more efficient pruning.

As shown in Alg. 1, a layer-wise dynamic Top-k pruning scheme is presented. The update of k is based on straightforward yet effective principles: a channel that is t times smaller than the max channel can be considered negligible, where t is a fixed parameter set to 16 in our design. Moreover, k should decrease progressively with layer depth. If the number of retained channels n is less than k, it indicates that the current k is too large. Thus, k is set to n.

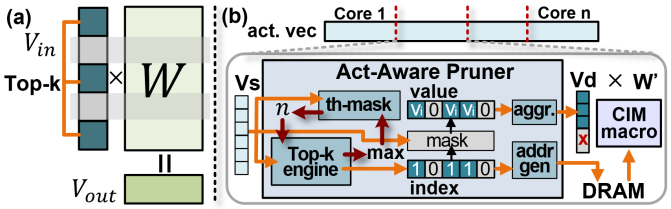


Fig. 8. Activation-aware weight pruning and hardware pruner.

Algorithm 1: Layer-wise Dynamic Top-k Pruning

input : Threshold t, Vector dim d
for *layers* in model **do**
  if $layer_index == 1$ **then**
    $k = d$; // no pruning
  $V_x^{'}$ = top-k(V_x, k) ; // select channels
  $W^{'}$ = pruning(W, index($V_x^{'}$)); // pruning
  GEMV($W^{'}$, $V_x^{'}$)
  n = count($V_x[i]$ ¿ $\frac{max(V_x[i])}{t}$)
  if $n < k$ **then**
    $k = n$; // update k
  /* remaining computation */
generate an output token

The hardware pruner in the MC-core facilitates the execution of GEMV with Alg. 1, as shown in Fig 8 (b). In practice, the activation vector is allocated to cores by channels. Each core focuses on its assigned local channels, avoiding complex global Top-*k* selections. The core first loads a slice of the activation vector into the vector register *vs*, then calls the hardware Act-Aware pruner for processing via specific instruction. In pruner, the Top-*k* engine finds the *max* value in *vs* and marks the selected elements with 1 in the index register. The th-mask receives the *max* value and *vs*, generating *n* in Alg. 1 for *k* update. The index is fed into the address generator, which calculates the address of the non-pruned weight rows for DRAM read requests. Following that, the *vs* is masked by the index and aggregated, and then output to *vd* register. Then the CIM macro computes the GEMV between *vd* and stored weights. The above process reduces DRAM access without wasting on-chip computation or memory resources.

B. Token-Length-Driven Bandwidth Management

The heterogeneity of EdgeMM complicates workload distribution. For MLLMs, it is optimal to run modality encoder and LLM-prefill on CC-clusters, with LLM-decoding on MC-clusters. In real-time applications (e.g. AD, robot, AR/VR), continuous streaming input enables pipelined execution. But the latency of LLM-decoding increases with the output token length, which can result in an unbalanced pipeline.

To maintain a balanced pipeline for various token lengths *l*, a throttling-based dynamic bandwidth allocation is adopted. For a model with default equal bandwidth sharing among clusters, an expected token length l_e can balance the latency of CC and MC clusters. As shown in Fig. 9(a), if l exceeds

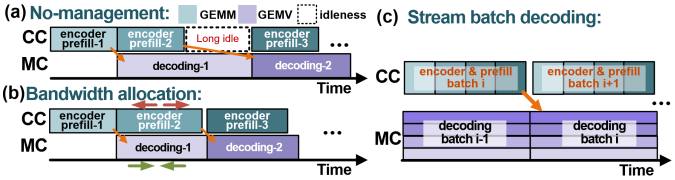


Fig. 9. Pipeline of bandwidth and workload management.

(a)

	CC-cluster	MC-cluster
Cores	4	2
Array(R,C)	(16,16)	(32,16)
OPs/s	2T	0.25T
Data Mem	32kB	128kB
DMA width	512bit	512bit
Crossbar width	1024bit	512bit
Area	$0.619mm^2$	$0.652mm^2$
Clusters in a group	2	2

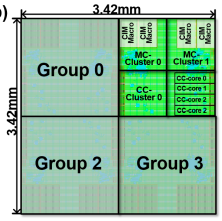


Fig. 10. Design configurations and the 22nm layout of EdgeMM.

l_e, CC-cluster will enter waiting idle. In such scenarios in Fig. 9(b), limiting CC-cluster bandwidth and reallocating it to MC-clusters can accelerate LLM-decoding, reducing overall latency. To implement bandwidth allocation, each cluster is assigned a memory access budget *B*. Within an interval *T*, the performance monitoring counter (PMC) in DMA accumulates the memory access usage d of each cluster. If $d > B$, subsequent DMA requests from this cluster are blocked until T elapses and PMCs are reset. We adjust the cluster budget B_c and B_m to enable dynamic bandwidth allocation.

Note the bandwidth allocation can only balance the pipeline with l in a certain range. For larger l, stream-based batch processing can be employed. As shown in Fig 9 (c), CC-clusters can encode and prefill a batch of streaming input without waiting for LLM-decoding. The MC cluster performs decoding concurrently on this batch. Larger batch multiplies the latency of CC-clusters, while decoding latency of MC-clusters increases slightly for the weight reuse. Hence the overall throughput can be boosted with minimal latency loss.

V. Evaluations

A. Implementation and Evaluation Setup

We leverage open-source Snitch_cluster [43] as the foundation platform for our EdgeMM development. The CC-cluster and MC-cluster are implemented using RTL flow and in-house 22nm digital CIM macro IP. We utilize Cadence Genus and Cadence Innovus for synthesis and physical implementation at 1GHz using TSMC 22nm technology. Fig. 10 demonstrates the layout view and design configurations of EdgeMM. The chip power is 112mW according to post-P&R report. The SA-based coprocessor occupies 62% area of a CC-core, and the CIM macro occupies 81% area of an MC-core. The performance is evaluated by RTL simulation and the in-house simulator with a dedicated mapping explorer.

B. Benefits of Heterogeneous Extensions

To evaluate the benefits of our heterogeneous AI-extensions, we compare our heterogeneous EdgeMM with homogeneous cluster designs: homo-CC and homo-MC, which consist solely of CC-clusters and MC-clusters. The original snitch_cluster

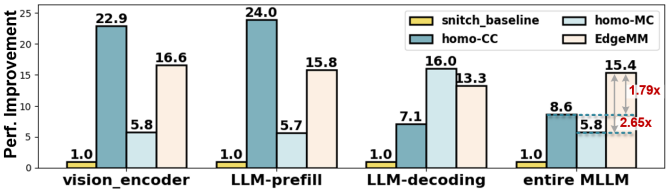


Fig. 11. Perf. improvements of homo. and hetero. designs

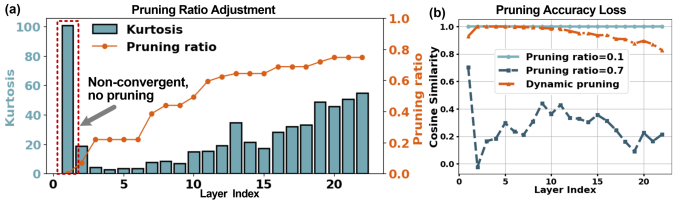


Fig. 12. Evaluation of activation-aware dynamic Top-k weight pruning.

[43] including SIMD cores is set as the baseline. The SPHINX-Tiny [25] MLLM and its inner phases with averaged input and output token length are used as the workload.

As shown in Fig. 11, all extended designs have significant performance boosts compared to the baseline since matrix extensions and coprocessors enhance computing power and reduce redundant load/store. Homo-CC and homo-MC perform the peak speedup in their adapted phases: a CC-cluster shows 4.3× better GEMM performance than an MC-cluster, whereas an MC-cluster is 2.42× faster in GEMV due to more efficient memory access. However, homo-CC and homo-MC encounter bottlenecks in non-adapted phases due to hardware mismatch. For the entire MLLM, the heterogeneous EdgeMM outperforms them by 1.79× and 2.65× respectively.

C. Benefits of Bandwidth Optimizations

Fig. 12 (a) presents the evaluation results for activation-aware dynamic TOP-k weight pruning during a token generation, where *Kurtosis* is a mathematical metric to represent channel-wise data distribution. Higher *Kurtosis* indicates more distinct outliers, i.e. more channels can be pruned. Results show that with *Kurtosis* increasing with layer depth, with pruning ratio of a randomly chosen core increases dynamically, proving the activation-awareness of dynamic TOP-k mechanism. Note that although the first layer exhibits a high *Kurtosis*, its data distribution remains unstable. Our experiment reveals that pruning this layer leads to significant accuracy degradation. Thus Alg. 1 skips the first layer.

To validate the accuracy of dynamic TOP-k pruning, we assess the cosine similarity between pruned and unpruned output vectors in FFN as depicted in Fig.12 (b). It is observed that dynamic pruning achieves comparable accuracy as a mild fixed pruning ratio of 0.1 while fixing pruning ratio at 0.7 causes irreversible accuracy loss in the shallow layers. Overall, our activation-aware pruning reduces the LLM-decoding latency of SPHINX-Tiny by 42% on average, with minimal score reduction in VQA [35].

Our dynamic bandwidth allocation can harness the heterogeneous cores under varying output token lengths l. As described in Fig. 13, when $l < l_e = 36$, bandwidth is not the critical

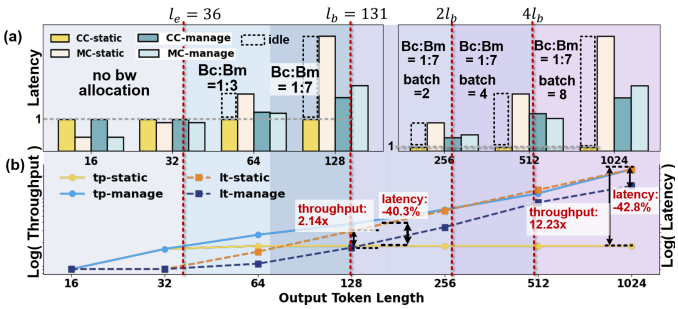


Fig. 13. (a) Latency and (b) throughput gains by bandwidth management.

bottleneck as the latency of CC-cores is large. As l rises, the extended duration of LLM-decoding slows down the pipeline. Thus the bandwidth budget of CC-cores B_c is reduced progressively, reallocating bandwidth to MC-cores. The B_c:B_m ratio ranges to 1:3 or even 1:7, accelerating MC-cores with tolerable latency increment of CC-cores. This optimization reduces latency by 40.3% and improves throughput 2.14× at l=128. When l increases to $l_b = 131$, bandwidth allocation nears its limit, where batch decoding is introduced to balance the pipeline. While batch decoding increases latency by 42% at $l = 1024$, it boosts throughput by 13.98×, compensating for the latency overhead.

We further compare EdgeMM with existing mobile GPU, as shown in Table II. Although SM cores in GPU provide massive parallelism, they often remain underutilized and not as effective as SA and CIM extensions in EdgeMM. Compared to laptop RTX 3060 GPU, EdgeMM achieves 2.15× performance for better resource utilization driven by heterogeneity. With our activation-aware weight pruning, the performance gain can be enhanced to 2.84×, reaching 138 tokens/s throughput. Overall, EdgeMM achieves a 0.28 token/J energy efficiency.

TABLE II
COMPARISON OF EDGEMM AND MOBILE GPU

	Compute Power	Bandwidth	MLLM perf.
RTX 3060 Laptop	13TFLOP/s (FP32)	GDDR6 336GB/s	1×
EdgeMM	18TFLOP/s (BF16)		2.15×
EdgeMM+Weight pruning	18TFLOP/s (BF16)		2.84×

VI. CONCLUSION

We present EdgeMM, a multi-core CPU with heterogeneous AI-extension for edge MLLM. To accommodate phases with diverse attributes in MLLM, coprocessors based on systolic arrays and digital CIM macros are integrated for GEMM and GEMV. To leverage limited bandwidth, activation-aware dynamic weight pruning is proposed with hardware support. Token-length-driven bandwidth allocation is employed to fully utilize heterogeneous cores. Overall, EdgeMM achieves 2.84× performance improvement compared to RTX 3060 GPU.

ACKNOWLEDGMENT

This work is supported in part by NSFC Grant No. 92464202 and CIE-Smartchip research fund.

REFERENCES

[1] C. Cui *et al.*, "A survey on multimodal large language models for autonomous driving," in *IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 958–979, 2024.

[2] Y. Liu, W. Chen, Y. Bai, J. Luo, X. Song, K. Jiang, Z. Li, G. Zhao, J. Lin, G. Li, *et al.*, "Aligning cyber space with physical world: A comprehensive survey on embodied ai," *arXiv preprint arXiv:2407.06886*, 2024.

[3] X. Chu, L. Qiao, X. Lin, S. Xu, Y. Yang, Y. Hu, F. Wei, X. Zhang, B. Zhang, X. Wei, *et al.*, "Mobilevlm: A fast, reproducible and strong vision language assistant for mobile devices," *arXiv preprint arXiv:2312.16886*, 2023.

[4] Y.-C. Chen, W.-H. Li, C. Sun, Y.-C. F. Wang, and C.-S. Chen, "Sam4mllm: Enhance multi-modal large language model for referring expression segmentation," *arXiv preprint arXiv:2409.10542*, 2024.

[5] H. Shao *et al.*, "Lmdrive: Closed-loop end-to-end driving with large language models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15120–15130, 2024.

[6] B. Keller *et al.*, "A 17–95.6 tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5nm," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI)*, 2022.

[7] T. Tambe *et al.*, "22.9 a 12nm 18.1tflops/w sparse transformer processor with entropy-based early exit, mixed-precision predication and fine-grained power management," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023.

[8] Z. XUANLEI, B. Jia, H. Zhou, Z. Liu, S. Cheng, and Y. You, "Hetegen: Efficient heterogeneous parallel inference for large language models on resource-constrained devices," in *Machine Learning and Systems* (P. Gibbons, G. Pekhimenko, and C. D. Sa, eds.), vol. 6, pp. 162–172, 2024.

[9] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore, *et al.*, "A survey on deep learning hardware accelerators for heterogeneous hpc platforms," *arXiv preprint arXiv:2306.15552*, 2023.

[10] Z. Jiang, N. Fisher, N. Guan, and Z. Dong, "Blueface: Integrating an accelerator into the core's pipeline through algorithm-interface co-design for real-time socs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023.

[11] Online in *https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/advanced-matrix-extensions/overview.html*.

[12] N. Nassif *et al.*, "Sapphire rapids: The next-generation intel xeon scalable processor," in *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 44–46, 2022.

[13] Y. Yuan *et al.*, "Intel accelerators ecosystem: An soc-oriented perspective," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 848–862, 2024.

[14] Online in *https://developer.arm.com/documentation/ddi0616/latest/*.

[15] Online in *https://github.com/T-head-Semi/riscv-matrix-extension-spec*.

[16] Q. Sun, Y. Cui, X. Zhang, F. Zhang, Q. Yu, Y. Wang, Y. Rao, J. Liu, T. Huang, and X. Wang, "Generative multimodal models are in-context learners," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14398–14409, 2024.

[17] Q. Sun, Y. Fang, L. Wu, X. Wang, and Y. Cao, "Eva-clip: Improved training techniques for clip at scale," *arXiv preprint arXiv:2303.15389*, 2023.

[18] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, 2024.

[19] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning (ICML)*, pp. 8748–8763, 2021.

[20] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, *et al.*, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, march 2023," *URL https://lmsys. org/blog/2023-03-30-vicuna*, vol. 3, no. 5, 2023.

[21] X. Chu, L. Qiao, X. Zhang, S. Xu, F. Wei, Y. Yang, X. Sun, Y. Hu, X. Lin, B. Zhang, *et al.*, "Mobilevlm v2: Faster and stronger baseline for vision language model," *arXiv preprint arXiv:2402.03766*, 2024.

[22] Z. Yuan, Z. Li, and L. Sun, "Tinygpt-v: Efficient multimodal large language model via small backbones," *arXiv preprint arXiv:2312.16862*, 2023.

[23] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *International Conference on Machine Learning (ICML)*, pp. 19730–19742, 2023.

[24] M. Javaheripi, S. Bubeck, M. Abdin, J. Aneja, S. Bubeck, C. C. T. Mendes, W. Chen, A. Del Giorno, R. Eldan, S. Gopi, *et al.*, "Phi-2: The surprising power of small language models," *Microsoft Research Blog*, 2023.

[25] P. Gao, R. Zhang, C. Liu, L. Qiu, S. Huang, W. Lin, S. Zhao, S. Geng, Z. Lin, P. Jin, *et al.*, "Sphinx-x: Scaling data and parameters for a family of multi-modal large language models," *arXiv preprint arXiv:2402.05935*, 2024.

[26] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.

[27] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," *arXiv preprint arXiv:2401.02385*, 2024.

[28] H. Lu, W. Liu, B. Zhang, B. Wang, K. Dong, B. Liu, J. Sun, T. Ren, Z. Li, Y. Sun, *et al.*, "Deepseek-vl: towards real-world vision-language understanding," *arXiv preprint arXiv:2403.05525*, 2024.

[29] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, "Sigmoid loss for language image pre-training," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11975–11986, 2023.

[30] Online in *Karmavlm: A family of high efficiency and powerful visual language model. https://github.com/ thomas-yanxin/KarmaVLM, 2024.*

[31] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[32] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[33] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[34] H. Liu, C. Li, Y. Li, and Y. J. Lee, "Improved baselines with visual instruction tuning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26296–26306, 2024.

[35] Y. Goyal, T. Khot, A. Agrawal, D. Summers-Stay, D. Batra, and D. Parikh, "Making the v in vqa matter: Elevating the role of image understanding in visual question answering," 2019.

[36] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan, "Learn to explain: Multimodal reasoning via thought chains for science question answering," *ArXiv*, vol. abs/2209.09513, 2022.

[37] B. Li, R. Wang, G. Wang, Y. Ge, Y. Ge, and Y. Shan, "Seed-bench: Benchmarking multimodal llms with generative comprehension," *ArXiv*, vol. abs/2307.16125, 2023.

[38] Y. Liu, H. Duan, Y. Zhang, B. Li, S. Zhang, W. Zhao, Y. Yuan, J. Wang, C. He, Z. Liu, K. Chen, and D. Lin, "Mmbench: Is your multi-modal model an all-around player?," in *European Conference on Computer Vision*, 2023.

[39] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, *et al.*, "Full stack optimization of transformer inference: a survey," *arXiv preprint arXiv:2302.14017*, 2023.

[40] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[41] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[42] Z. A. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. W. Barrett, Z. Wang, and B. Chen, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *ArXiv*, vol. abs/2306.14048, 2023.

[43] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Transactions on Computers*, 2020.

[44] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," *arXiv preprint arXiv:2306.11695*, 2023.

[45] J.-Y. Lee, D. Lee, G. Zhang, M. Tiwari, and A. Mirhoseini, "Cats: Contextually-aware thresholding for sparsity in large language models," *arXiv preprint arXiv:2404.08763*, 2024.

Anchor First, Accelerate Next: Revolutionizing GNNs with PIM by Harnessing Stationary Data

Jiaxian Chen[1], Yuxuan Qi[1], Yongbiao Zhu[1], Jianan Yuan[1], Kaoyi Sun[1], Tianyu Wang[1], Chenlin Ma[1], Yi Wang[1,2]
1. College of Computer Science and Software Engineering, Shenzhen University
2. State Key Laboratory of Radio Frequency Heterogeneous Integration

***Abstract*—Substantial data movement caused by irregular graph topologies hinders the efficient processing of graph neural networks (GNNs). Although the emerging near-bank processing-in-memory (PIM) architecture offers a promising solution to reduce data transfer between memory and computing units, cross-bank communication remains a critical challenge, limiting the benefits of PIM architectures. Our findings indicate that only 35.6% of the data can stay stationary within PIM units on average, with the rest requiring movement due to graph dependencies. This situation worsens as the number of PIM units increases, reducing the ratio to 18.7%.**

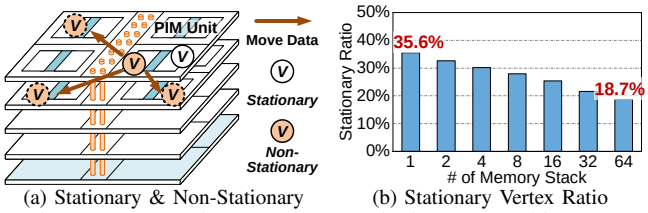
In this paper, we argue that to fully leverage PIM architectures, systems must *maximize stationary data and minimize the movement of non-stationary data*. Following this principle, we propose *Anchor*, a scalable PIM architecture that exploits stationary data for GNNs through a hardware-software co-design approach. To maximize stationary data, we introduce the graph partitioning algorithm Mastav, which carefully allocates vertices and edges to preserve data locality. To minimize the movement of non-stationary data, we employ a two-step strategy. First, a customized dataflow ensures that non-stationary data is accessed and distributed exactly once. Second, an optimized communication mechanism reduces redundant data transfers through critical paths. Our extensive experiments demonstrate that Anchor significantly reduces processing latency and data movement compared to representative schemes.

***Index Terms*—GNN, Processing-in-Memory, Stationary Data**

I. Introduction

Graph neural networks (GNNs) have emerged as a powerful learning paradigm for graph-structured data [11], [14], [23]. Computational efficiency is a primary concern in such deep learning systems, where vast numbers of nodes and edges have to be simultaneously processed. GNNs mainly consist of two time-consuming phases: (i) *combination* and (ii) *aggregation*. The sparse and uncertain topology of the input graph results in significant data movement, posing a severe challenge to the efficient processing of GNNs. The emerging 3D-stacked processing-in-memory (PIM) architecture [15], [22] allows computing units to be integrated near memory banks. The PIM architecture has proven to be an effective approach for addressing the data movement issue in GNNs [6], [7]. While PIM-based accelerators are already in place, the data movement problem still persists, leading to performance degradation. We observe that an unsolved key challenge is how to minimize unnecessary data movement across different PIM units.

Fig. 1 illustrates two types of vertices: *stationary* and *non-stationary*. Stationary vertices stay within their local processing units (PIM units) during aggregation, while non-stationary vertices have to migrate to remote PIM units due to graph dependencies. From the preliminary test results, only 35.6% of vertices are stationary vertices. Over 60% of the vertices are migrated to other PIM units. As the number of PIM units increases, the amount of stationary data drops quickly, indicating significant opportunities for optimization. To address this issue, we propose the ***Max-Min principle****: a PIM system must maximize stationary data and minimize the movement of non-stationary data.*



(a) Stationary & Non-Stationary (b) Stationary Vertex Ratio

Fig. 1: Low stationary vertex ratio in PIM architectures.

Many domain-specific architectures [20], [25], [30] have been proposed to address the challenges faced by GNNs. Some works capture the memory access patterns of GNNs and alleviate irregular data access by employing dedicated buffers [12], [16], [24]. Other works focus on reducing redundant memory accesses through algorithmic and architecture co-optimization [5], [10], [29]. Existing works only partially solve the issue by reducing memory accesses. Considerable efforts on architectural and system-software support are required to reduce the significant data movement.

This paper presents *Anchor*, a scalable PIM architecture harnessing stationary data to efficiently process GNNs. Anchor follows the proposed Max-Min principle using a hardware-software co-design approach. The objective is to fully exploit the PIM architecture and reduce data movement. At the software level, we introduce the graph partitioning algorithm *Mastav*, which maximizes stationary data by strategically allocating vertices and edges to maintain data locality. At the hardware level, we design a customized PIM architecture with a two-step optimization to minimize the movement of non-stationary data. A hybrid dataflow ensures that each non-stationary vertex is distributed only once, while an optimized communication mechanism further minimizes redundant data transfers through critical paths. We evaluate the proposed Anchor using a set of extensive experiments. The experimental results show that Anchor can significantly reduce latency and

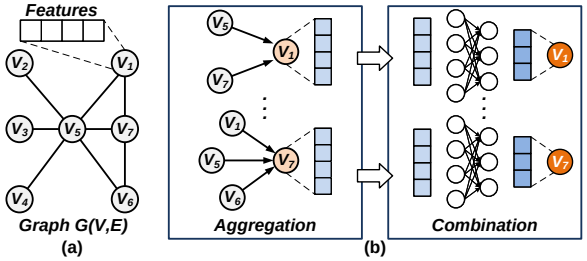


Fig. 2: (a) An input graph. (b) Illustration of a GNN model.

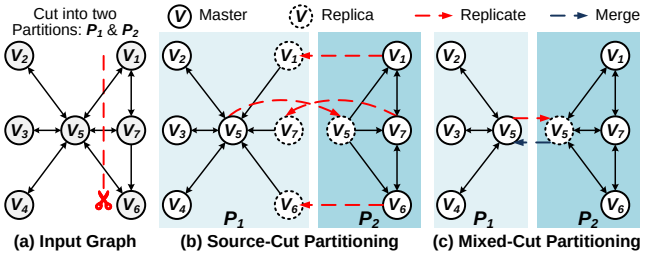


Fig. 3: Source-cut partitioning and mixed-cut partitioning.

data movement compared with representative schemes.

The main contributions of this work include:

- *Principle*: we propose the Max-Min principle to maximize stationary data and minimize the movement of non-stationary data.
- *Algorithm*: A novel graph partitioning algorithm *Mastav* is proposed to maximize stationary data within PIM units.
- *Dataflow*: A hybrid dataflow and an optimized communication mechanism are introduced to reduce redundant data movement.
- *Architecture*: A scalable PIM architecture with dedicated compute and control logic is carefully designed to implement the Max-Min principle.

II. Background

A. Graph Neural Networks (GNNs)

A GNN model generally consists of multiple GNN layers [11], [14], each with two main phases: aggregation and combination. As shown in Fig. 2(a), GNN models take a graph $G(V, E)$ as input, where V and E denote vertices and edges, respectively, and each vertex is represented by a feature vector. In the aggregation phase, each vertex aggregates features from neighboring vertices using an *aggregate* function, such as the mean function. In the combination phase, each aggregated feature vector is further transformed via a *combine* function, typically implemented as a multi-layer perceptron (MLP). Fig. 2(b) illustrates a typical GNN inference workflow. Vertex V_1 aggregates features from neighbors V_5 and V_7, generating a four-dimension feature vector. An MLP then transforms it into three dimensions. The k-th GNN layer is defined as:

$$X^{k+1} = \sigma(AX^kW^k) \tag{1}$$

Here, σ denotes the non-linear activation function, A is the sparse matrix derived from E, and X^k and W^k represent the feature and weight matrices for the k-th layer, respectively. The execution order of the combination and aggregation phases can be interchanged, and we adopt a combination-first approach to reduce computations, as suggested in prior work [9], [16].

B. Graph Partitioning for GNNs

Source-cut and mixed-cut are two typical edge partitioning methods. Source-cut partitioning [26] assigns a vertex and all its incoming edges to the same partition, making it suited for GNNs since they aggregate vertices using incoming edges. Fig. 3(b) presents an example for the source-cut partitioning. During the aggregation phase, V_5 pulls remote vertices V_1, V_6, and V_7 from P_2, creating their replicas in P_1. This allows V_5 to perform aggregation operations locally in P_1. Similarly, V_7 pulls V_5 from P_1 to P_2. Mixed-cut partitioning [27] allocates edges to the vertex with a lower degree, allowing low-degree vertices to remain stationary and reducing replicas. This approach requires an additional *merging* phase to gather updated replicas since incoming edges of a vertex may span multiple partitions. As shown in Fig. 3(c), V_5 creates a replica in P_2, with edges (V_1, V_5), (V_6, V_5), and (V_7, V_5) assigned to vertices in P_2 due to their lower degrees. Partial aggregation of V_5 occurs in P_2, and the replica is merged back to P_1. Mixed-cut partitioning requires only two communications, compared to four for source-cut partitioning.

C. Processing-in-Memory (PIM) Architecture

As shown in Fig. 4(a), a typical PIM system consists of multiple memory stacks connected via external links, with a dragonfly-based network widely adopted for its simple topology and high performance [8]. This network topology includes multiple interconnected groups, with each group fully connected internally and fully interconnected with other groups. Each stack is managed using a memory controller (MC). Fig. 4(b) illustrates a PIM architecture enabled by 3D-stacking technology. Multiple DRAM dies are stacked on a base die and interconnected by through-silicon vias (TSVs). Each DRAM die is organized into channels, with each channel comprising bank groups and each bank group comprising individual banks. To reduce off-chip data movement, many PIM systems [18] integrate compute logic directly on the base die. The emerging PIM architectures [3], [15], [19] further integrate compute and control logic near memory banks, forming a PIM unit, as depicted in Fig. 4(c).

D. Motivation: Rethinking Graph Partitioning for PIM

Graph partitioning for PIM architecture requires careful design to reduce data movement. Since data movement is

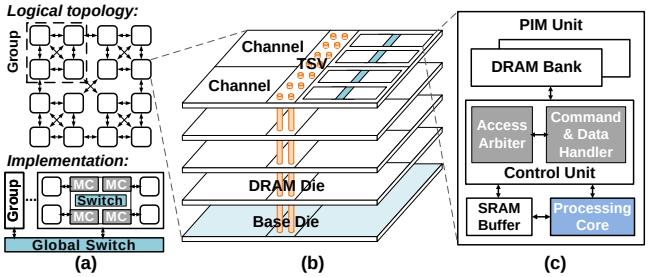


Fig. 4: A PIM system. (a) Dragonfly-based topology. (b) 3D-stacked PIM architecture. (c) PIM-enabled bank group.

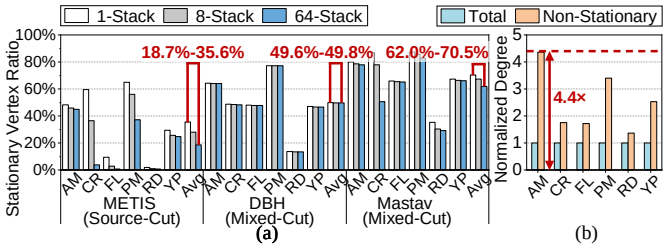


Fig. 5: (a) Ratio of stationary vertices. (b) Normalized average degrees of total and non-stationary vertices.

primarily caused by non-stationary data, maintaining data stationary within local PIM units can prevent data from traveling to remote PIM units. Therefore, we argue that maximizing stationary data is a promising direction to fully exploit the potential of PIM architecture. To validate our idea, we conduct a series of preliminary experiments. Fig. 5(a) presents the ratio of stationary vertices using different partitioning methods. The source-cut-based partitioning method METIS [13] demonstrates an unsteady ratio as the number of PIM stacks increases, declining from 35.6% to 18.7%. In contrast, the mixed-cut-based method DBH [21] maintains a steady ratio, ranging between 49.6% and 49.8%. There is still significant potential to increase stationary vertices. Our proposed Mastav achieves higher ratios, ranging from 62.0% to 70.5%.

Data movement can be further reduced by optimizing the distribution of non-stationary vertices. Fig. 5(b) presents the normalized average degrees of all vertices and non-stationary vertices using our Mastav algorithm. The degrees of non-stationary vertices are notably higher, ranging from 1.3× to 4.4× the average. This indicates that non-stationary vertices tend to have more connections. These vertices often need to move data repeatedly, requiring further optimization. Based on these observations, we are motivated to design a new PIM architecture for GNNs through a hardware-software co-design approach, following the Max-Min principle.

III. Anchor: A Scalable PIM Architecture Harnessing Stationary Data for GNNs

The proposed Anchor is built based on the Max-Min principle for efficiently processing GNNs via a hardware-software co-design approach. The objective is to take advantage of the near-bank PIM architectures and minimize data movement. The overview of Anchor is presented in Fig. 6. At the software level, a graph partitioning algorithm, *Mastav*, is proposed maximize stationary data by exploiting locality. At the hardware level, a novel architecture with two-step optimization, including *hybrid dataflow* and *optimized communication*, is implemented to minimize data movement through critical paths. As shown in Fig.6, the PIM unit is redesigned to support hybrid dataflow, while additional base die logic is integrated to support optimized communication.

A. Mastav: Maximizing Stationary Vertices

To maximize stationary vertices, the graph partitioning algorithm Mastav is proposed to optimize for both edges and vertices allocation. For edge allocation, Mastav employs mixed-cut partitioning to enhance vertex stationarity, as discussed in

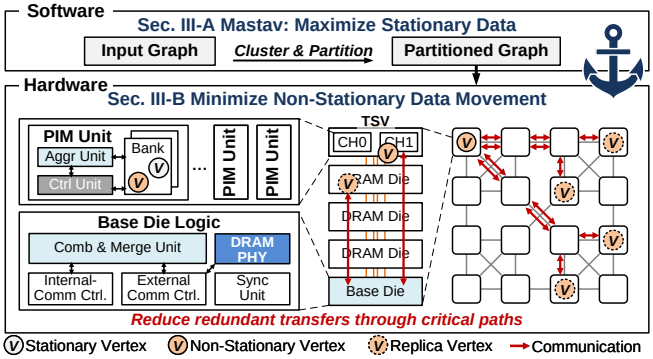


Fig. 6: Overview of Anchor.

Section II-D. For vertex allocation, Mastav uses modularity-based vertex clustering to group vertices with strong localities within the same PIM units, thereby minimizing data movement across PIM units.

$$\Delta Q(c,t) = 2\left\{\frac{w_{ct}}{2|E|} - \frac{\text{Degree}(c)\cdot\text{Degree}(t)}{(2|E|)^2}\right\} \tag{2}$$

As illustrated in Algorithm 1, the Mastav algorithm consists of two main stages: edge partitioning and vertex allocation. It begins with mixed-cut edge partitioning, where each edge is assigned to the vertex with the lower degree (Lines 1-6). Then, vertex clustering is performed. Mastav initializes individual clusters for each vertex and sorts these clusters by vertex degrees (Line 7). For each cluster c, a target vertex t among its neighbors is selected to maximize the modularity increment ΔQ (Line 9), as defined in Equation (2) [4]. Here, E is the edge set, and w_{ct} is the weight of edges between clusters. A higher ΔQ indicates a stronger connection between clusters. If joining c with t results in a positive modularity gain and the combined edge count remains below the threshold T_e, the clusters are joined (Lines 10-11). The edge threshold balances edge allocation to fully exploit the parallelism provided by PIM. The joining process repeats until no further beneficial joining operations can be performed, and the output cluster set C is obtained. The time complexities of Mastav is $O(|E|)$. The edge partitioning traverses all edges to allocate them based

Algorithm 1: Mastav: A Graph Partitioning Algorithm

Input: Vertex set V, and threshold of edges T_e.
Output: Cluster set C.
// Mixed-cut edge partitioning
1 **foreach** $u \in V$ **do**
2 **foreach** $v \in N(u)$ **do**
3 **if** Degree(u) < Degree(v) **then**
4 Allocate (u, v) to u
5 **else**
6 Allocate (u, v) to v
// Modularity-based clustering
7 Initialize $C \leftarrow \{\{u\} \mid u \in V\}$ and sort C by ascending order of degree.
8 **foreach** $c \in C$ **do**
9 $t \leftarrow \arg\max_{t \in \mathbf{N}(c)} \Delta Q(c,t)$
10 **if** $\Delta Q(c,t) > 0$ and Edge(t) + Edge(c) < T_e **then**
11 Join c to t and Delete c from C
12 **return** C

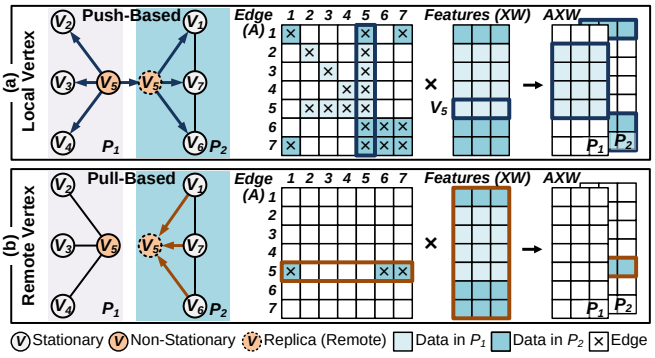


Fig. 7: Hybrid dataflow.

on vertex degrees, and it has a time complexity of $O(|E|)$. The vertex clustering calculates the maximum modularity increment by traversing the neighbors of each vertex, also resulting in a time complexity of $O(|E|)$.

B. Minimizing Non-Stationary Data Movement

Non-stationary vertices are often required by multiple PIM units, incurring repeated data transfers. To minimize data movement, Anchor introduces a two-step optimization with hardware support. A hybrid dataflow ensures each non-stationary vertex is distributed only once, and optimized communication reduces redundant movement during distribution.

*1) **Hybrid Dataflow with Hardware Support**:*

Hybrid Dataflow. Anchor employs a hybrid dataflow for aggregation, combining push-based and pull-based approaches. The push-based dataflow broadcasts features from local vertices to their neighbors along outgoing edges, as shown in Fig. 7(a), where vertex V_5 broadcasts to V_1, V_6, and V_7 in P_2 and to V_2, V_3, and V_4 in P_1. This ensures that features of non-stationary vertex V_5 are distributed once, minimizing redundant accesses and transfers. The push-based dataflow processes edges in column order, and thus a compressed sparse row (CSR) format is used for edges of local vertices. Processing remote vertices with a push-based dataflow can exhaust memory due to the need to store all intermediates during aggregation. Therefore, the pull-based dataflow is adopted to handle remote vertices. As shown in Fig. 7(b), remote vertex V_5 in P_2 pulls features from neighbors, V_1, V_6, and V_7, along incoming edges. The pull-based dataflow processes edges in row order using a compressed sparse column (CSC) format.

Hardware Support. As shown in Fig. 8, the PIM unit is redesigned with an aggregation unit, an aggregation queue, and a vertex locator to support the proposed hybrid dataflow. The aggregation unit includes MAC arrays and dedicated SRAM buffers for aggregation. The vertex locator manages the allocation of space and the search for vertices addresses, and the aggregation queue stores vertex IDs prepared for aggregation. Similar to previous works [19], the access arbiter manages memory access within and outside the PIM unit, and the handlers process external instructions to enable cross PIM unit communication.

Anchor organizes the bank space into three distinct zones: edge data zone, local vertex zone, and remote vertex zone,

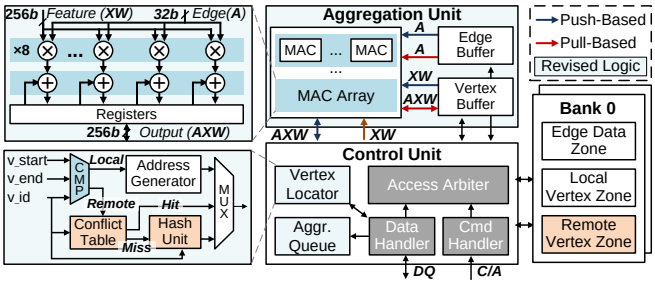


Fig. 8: Architecture design of the PIM unit.

to support the hybrid dataflow. The edge data zone stores the weights of edges assigned to the PIM unit. The local vertex zone holds the features of the vertices allocated to the PIM unit. The remote vertex zone temporarily caches features of non-stationary vertices from other PIM units, utilizing huge DRAM space to prevent blocking of remote vertex delivery. Anchor employs a hash unit and a conflict table to manage the space allocation of the remote vertex zone.

The aggregation unit supports both push-based and pull-based dataflows, as shown in Fig. 8. Local vertices are aggregated using the push-based dataflow. First, a portion of edge weights (A) is read from the edge data zone into the edge buffer. Second, IDs of prepared vertices are fetched from the head of the aggregation queue, and their features (XW) are retrieved from the banks to the vertex buffer. Third, the MAC arrays then perform matrix multiplication with edges and vertex features, accumulating the resulting output (AXW) back into the local vertex zone. Remote vertices are aggregated using the pull-based dataflow, following a similar process.

Anchor defines three types of extended instructions for GNNs, using standard DRAM commands with reserved DRAM rows and columns [15], [19]. The `Sync` instruction inserts vertex IDs into the aggregation queue, `Replic` stores remote vertex features to banks, and `Merge` fetches aggregated replicas from banks to the base die. The data and command handlers decode these instructions.

*2) **Optimized Communication with Hardware Support**:*

Optimized Communication. The proposed customized dataflow enables simultaneous processing of each non-stationary vertex, facilitating communication optimization. As shown in Fig. 9(a), Anchor uses a broadcast mechanism to optimize replicating operations during the aggregation phase. Local broadcast is used to reduce intra-stack data movement, where vertices updated after the combination operation are directly broadcast from the base die to PIM units via TSVs. To optimize inter-stack communication, Anchor utilizes the hierarchical structure of the dragonfly network [26], employing a tree-based global broadcast to minimize redundant data movement through critical paths. As illustrated in Fig. 9(a), the repeated access paths (S00, S01), (S00, S02), (S02, S08), and (S01, S04) are reduced from two accesses each to a single access. Similarly, a tree-based reduction is adopted for the merging phase, as depicted in Fig. 9(b). This phase conducts both local and global reduction. Local reduction pulls local replicas to the base die for merging, while global reduction

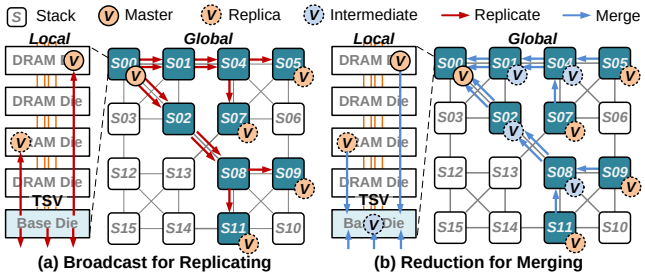


Fig. 9: Communication optimization mechanisms.

merges replicas from child nodes in the reduction tree.

Hardware support. Fig. 10 illustrates the architecture design of the base die logic. A combination and merging unit (MCU) is added for the combination and merging phase. An internal communication (IC), an external communication (EC) controller, and a synchronization unit are designed to support the proposed communication optimization. The IC controller generates commands and instructions to interact with PIM units, while the EC controller packets and delivers data to other stacks. To manage the delivery of vertices, a sliding window approach is adopted [28]. The synchronization unit distributes vertex IDs prepared for aggregation to local PIM units.

During the combination phase, the MCU uses a pull-based dataflow to perform matrix multiplication similar to the aggregation phase. After the combination phase, each updated vertex is delivered by the IC and EC controllers to different PIM units. Stationary vertices are directly returned to their allocated PIM units via the IC controller using the `Replic` instruction. Non-stationary vertices are broadcast by both the IC and EC controllers. Vertices that complete `Replic` instructions are sent to the synchronization unit, which then broadcasts `Sync` instructions to the PIM units for further aggregation. During the merging phase, the IC controller issues `Merge` instructions to PIM units, pulling aggregated replicas to the base die for local reduction. Once a vertex completes local reduction, its intermediate result (*Mid*) is delivered by the EC controller for global reduction. Finally, vertices that have completed merging are returned to their allocated PIM units.

IV. Evaluation

A. Experimental setup

Models and Datasets. To evaluate our scheme, we employ a set of representative GNN models, including GCN [14], GIN [23], and GraphSage [11]. Each GNN model consists of

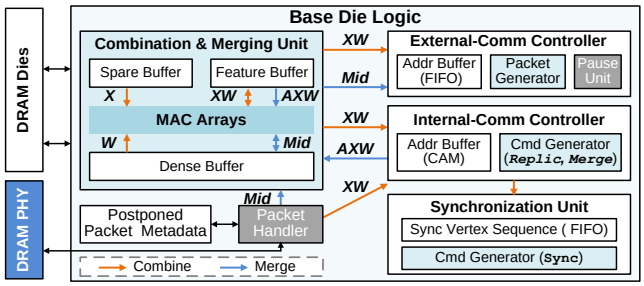


Fig. 10: Architecture design of the base die logic.

TABLE I: Graph datasets.

Dataset	# of vertices	# of edges	# of features	# of classes
Amazon (AM)	1,569,960	264,339,468	200	107
Cora (CR)	2,708	10,556	1,433	7
Flickr (FL)	89,250	989,006	500	7
Pubmed (PM)	19,717	88,648	500	3
Reddit (RD)	232,965	114,615,892	602	41
Yelp (YL)	716,847	13,954,819	300	100

TABLE II: System configurations and power-area models.

	Configuration
PIM System	16 stacks × 16 channels × (1 bank group + 3 PIM bank groups) × 4 banks, 8GB per stack, 48 PIM units per stack, 16GB/s intra-stack bandwidth per channel, 64GB/s inter-stack bandwidth per bidirectional link.
PIM unit	4×8 MACs @ 125MHz, 1KB conflict table, 2KB edge buffer, 2KB vertex buffer, Power=42.66mW, Area=0.18mm^2 (4.9 % of a bank group)
Base Die Logic	32×16 MACs @ 1MHz, 64KB sparse buffer, 32KB feature buffer, 1MB dense buffer, 8KB address buffer (FIFO), 8KB address table (CAM), Power=2719.68mW, Area=9.48mm^2(10.6% of the base die)

two layers with a hidden size of 64. As listed in Table I, we utilize six commonly used graph datasets [2] for evaluation.

Implementation and Configuration. We modify and extend DRAMSim3 [17], a cycle-accurate simulator, to support in-memory operations for GNNs. The area and power consumption of the processing units are obtained by implementing Anchor in Verilog and synthesizing the RTL using the Synopsys Design Compiler with a 32nm standard library. We employ CACTI [1] to estimate the SRAM buffer, interconnect components, and the DRAM stack. Detailed configurations, power models, and area models are listed in Table II.

Baselines. At the hardware level, we compare Anchor with state-of-the-art architectures, including HyGCN [24], GROW [12], GCIM [6], and Lift [7]. HyGCN and GROW are typical GNN accelerators, whereas GCIM and Lift are near-bank PIM architectures for GNNs. At the software level, we compare the proposed Mastav with graph partitioning methods, including METIS [13], DBH [21], and Lift [7].

B. Overall Performance

1) Speedup: Fig. 11 illustrates the performance speedup across various benchmarks, with Anchor outperforming all baselines. Specifically, Anchor achieves average speedups of 13.01×, 3.57×, 10.12×, and 3.15× over HyGCN, GROW, GCIM, and Lift, respectively. This performance gain is primarily driven by PIM architectures, which offer substantial processing bandwidth and minimize data movement. Additionally, Anchor further reduces data movement across PIM

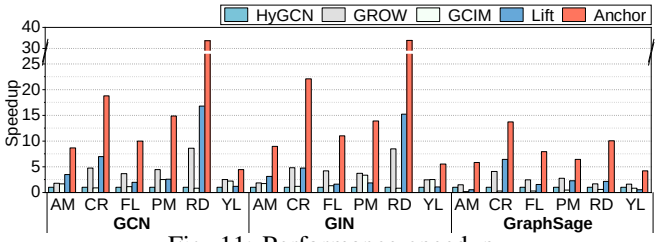


Fig. 11: Performance speedup.

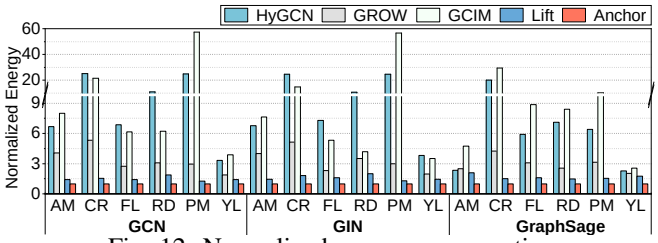


Fig. 12: Normalized energy consumption.

units by efficiently leveraging stationary data, enabling it to outperform other PIM architectures, including GCIM and Lift.

2) Energy Consumption: Fig. 12 presents the normalized energy consumption across various benchmarks, demonstrating that Anchor achieves the lowest average energy consumption. Anchor achieves energy savings of 11.10×, 3.19×, 14.40×, and 1.58× compared to HyGCN, GROW, GCIM, and Lift, respectively. This reduction is attributed to the hybrid dataflow, which minimizes unnecessary memory accesses, and optimized communication mechanisms that reduce redundant data movement.

C. Comparison of Graph Partitioning Algorithms

1) Performance and Scalability: Fig. 13 shows the speedups for different graph partitioning algorithms as the number of stacks increases. It can be observed that Mastav achieves the highest speedup among the baseline algorithms. As the stack number increases from 4 to 16 and 64, Mastav improves performance by an average of 3.22× and 7.16×.

2) Data Movement: Fig. 14(a) presents the data movement for different graph partitioning algorithms. Compared to Lift and DBH, Mastav reduces data movement by 40.55% and 38.44%, respectively. For the AM, CR, and PM graphs, METIS incurs less data movement than Mastav by prioritizing the minimization of cross-partition edges, often at the cost of workload balance. In contrast, Mastav effectively reduces data movement while maintaining balanced workloads.

3) Energy Breakdown: Fig. 14 (b) illustrates the normalized energy breakdown for various graph partitioning algorithms, where 'M', 'L', 'D', and 'A' denote METIS, Lift, DBH, and Mastav, respectively. Mastav achieves the lowest energy consumption, reducing energy usage by 28.15%, 26.30%, and 35.07% compared to METIS, Lift, and DBH, respectively. This reduction is attributed to Mastav's ability to minimize data movement, thereby lowering energy consumption.

4) Stationary Vertex Ratio: Fig. 15 shows the ratio of stationary vertices for different graph partitioning algorithms as the number of stacks increases. Both METIS [13] and Lift [7], which utilize source-cut edge partitioning, experience

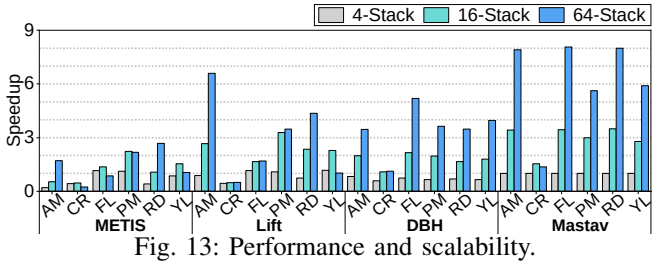


Fig. 13: Performance and scalability.

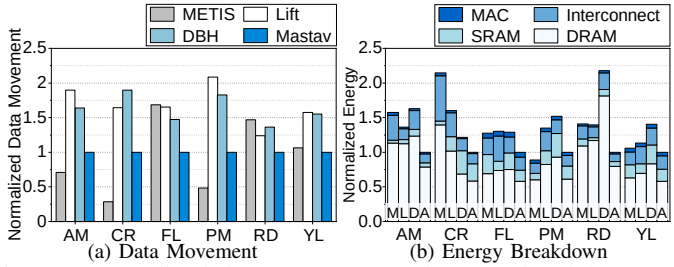


Fig. 14: Normalized data movement and energy breakdown.

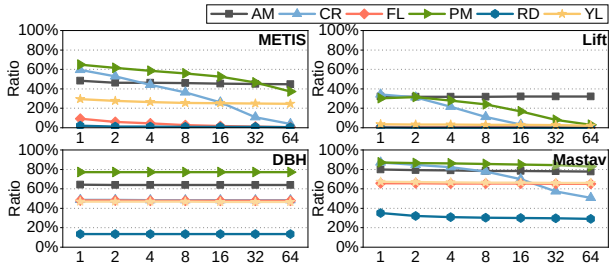


Fig. 15: Ratios of stationary vertices for different graph partitioning algorithms as the number of stacks increases.

a sharp decline in stationary vertex ratios as the stack count increases, dropping from 35.6% to 18.7% for METIS and from 16.9% to 6.3% for Lift. In contrast, DBH and Mastav, employing mixed-cut partitioning, maintain a stable stationary vertex ratio as the stack count increases. Compared to DBH, Mastav leverages clustering to optimize locality, achieving the highest stationary ratios, ranging from 62.0% to 70.5%.

V. Conclusion

This paper presents Anchor, a scalable PIM architecture for GNNs through a hardware-software co-design approach. Following the Max-Min principle, we focus on maximizing stationary data and minimizing the movement overhead of non-stationary data. A graph partitioning algorithm, Mastav, is proposed to maintain data in place. Furthermore, an optimized dataflow and a communication mechanism are introduced to reduce memory access and data movement. Experimental results demonstrated that Anchor significantly reduces latency and data movement compared to existing schemes. In the future, we plan to explore more architectural characteristics of PIM architecture to handle dynamic and more complex graph workloads.

Acknowledgements

The work described in this paper is supported in part by NSFC (U23B2040), in part by National Key Research and Development Program of China (2024YFB4505702), in part by Shenzhen Science and Technology Program (RCJC20221008092725019), in part by Guangdong Province Key Laboratory of Popular High Performance Computers (2017B030314073), in part by Guangdong Province Engineering Center of China-made High Performance Data Computing System and Shenzhen Key Laboratory of Service Computing and Applications. Yi Wang is the corresponding author.

REFERENCES

[1] "CACTI," https://www.hpl.hp.com/research/cacti, 2024.

[2] "Pytorch Geometric," https://github.com/pyg-team/pytorch_geometric, 2024.

[3] "UPMEM," https://www.upmem.com/, 2024.

[4] J. Arai, H. Shiokawa, T. Yamamuro, M. Onizuka, and S. Iwamura, "Rabbit Order: Just-in-Time Parallel Reordering for Fast Graph Analysis," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 22–31.

[5] C. Chen, K. Li, Y. Li, and X. Zou, "ReGNN: A Redundancy-Eliminated Graph Neural Networks Accelerator," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 429–443.

[6] J. Chen, Y. Lin, K. Sun, J. Chen, C. Ma, R. Mao, and Y. Wang, "GCIM: Toward Efficient Processing of Graph Convolutional Networks in 3D-Stacked Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 3579–3590, 2022.

[7] J. Chen, Z. Zhong, K. Sun, C. Ma, R. Mao, and Y. Wang, "Lift: Exploiting Hybrid Stacked Memory for Energy-Efficient Processing of Graph Convolutional Networks," in *Design Automation Conference (DAC)*, 2023, pp. 1–6.

[8] A. Fatima, S. Liu, K. Seemakhupt, R. Ausavarungnirun, and S. Khan, "vPIM: Efficient Virtual Address Translation for Scalable Processing-in-Memory Architectures," in *Design Automation Conference (DAC)*, 2023, pp. 1–6.

[9] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt, "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," in *International Symposium on Microarchitecture (MICRO)*, 2020, pp. 922–936.

[10] T. Geng, C. Wu, Y. Zhang, C. Tan, C. Xie, H. You, M. Herbordt, Y. Lin, and A. Li, "I-GCN: A Graph Convolutional Network Accelerator with Runtime Locality Enhancement through Islandization," in *International Symposium on Microarchitecture (MICRO)*, 2021, pp. 1051–1063.

[11] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 1024–1034.

[12] R. Hwang, M. Kang, J. Lee, D. Kam, Y. Lee, and M. Rhu, "GROW: A Row-Stationary Sparse-Dense GEMM Accelerator for Memory-Efficient Graph Convolutional Neural Networks," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 42–55.

[13] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, pp. 359–392, 1998.

[14] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations (ICLR)*, 2017, pp. 1–14.

[15] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product," in *International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43–56.

[16] J. Li, A. Louri, A. Karanth, and R. Bunescu, "GCNAX: A Flexible and Energy-Efficient Accelerator for Graph Convolutional Neural Networks," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 775–788.

[17] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[18] B. Tian, Q. Chen, and M. Gao, "ABNDP: Co-optimizing Data Access and Load Balance in Near-Data Processing," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 3–17.

[19] B. Tian, Y. Li, L. Jiang, S. Cai, and M. Gao, "NDPBridge: Enabling Cross-Bank Coordination in Near-DRAM-Bank Processing Architectures," in *International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1–16.

[20] H. Wang, S. Zhang, X. Fan, Z. Yang, and M. Zhang, "NDPGNN: A Near-Data Processing Architecture for GNN Training and Inference Acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 3997–4008, 2024.

[21] C. Xie, L. Yan, W.-J. Li, and Z. Zhang, "Distributed Power-law Graph Computing: Theoretical and Empirical Analysis," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 1–9.

[22] T. Xie, T. Zhao, Z. Zhu, X. Ning, B. Li, G. Dai, H. Yang, and Y. Wang, "DyPIM: Dynamic-Inference-Enabled Processing-In-Memory Accelerator," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[23] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *International Conference on Learning Representations (ICLR)*, 2019, pp. 1–17.

[24] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "HyGCN: A GCN Accelerator with Hybrid Architecture," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2020, pp. 15–29.

[25] H. Yu, Y. Zhang, L. He, D. He, Q. Li, J. Zhao, X. Liao, H. Jin, L. Gu, and H. Liu, "CDA-GNN: A Chain-driven Accelerator for Efficient Asynchronous Graph Neural Network," in *Design Automation Conference (DAC)*, 2024, pp. 1–6.

[26] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 544–557.

[27] C. Zhao, R. D. Chamberlain, and X. Zhang, "SuperCut: Communication-Aware Partitioning for Near-Memory Graph Processing," in *International Conference on Computing Frontiers (CF)*, 2023, pp. 42–51.

[28] Z. Zhou, C. Li, X. Wei, X. Wang, and G. Sun, "GNNear: Accelerating Full-Batch Training of Graph Neural Networks with Near-Memory Processing," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2023, pp. 54–68.

[29] Z. Zhou, B. Shi, Z. Zhang, Y. Guan, G. Sun, and G. Luo, "BlockGNN: Towards Efficient GNN Acceleration Using Block-Circulant Weight Matrices," in *Design Automation Conference (DAC)*, 2021, pp. 1009–1014.

[30] Z. Zhu, F. Li, G. Li, Z. Liu, Z. Mo, Q. Hu, X. Liang, and J. Cheng, "MEGA: A Memory-Efficient GNN Accelerator Exploiting Degree-Aware Mixed-Precision Quantization," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 124–138.

IntraFuzz: Coverage-Guided Intra-Enclave Fuzzing for Intel SGX Applications

Jinhua Cui, Qiao Peng, Yiwen Yao, Ke Ye, Jiliang Zhang*
College of Semiconductors (College of Integrated Circuits), Hunan University, Changsha 410082, China
{jhcui, pqlush, ywyao, keye, zhangjiliang}@hnu.edu.cn

***Abstract*—Intel SGX is susceptible to intra-enclave software vulnerabilities. Existing automated bug-finding methods primarily focus on fuzzing enclave boundaries for SGX applications in simulated, rather than *actual* hardware-protected enclaves. This limits the ability to identify potential security violations originating from within SGX application code. This paper presents INTRAFUZZ, the first system that enables efficient fuzzing of SGX applications inside *actual* hardware enclaves. We evaluated INTRAFUZZ with 21 real-world SGX applications, running on Intel Xeon scalable processors with up to 256 GB of enclave page cache. INTRAFUZZ successfully detected all vulnerabilities in SGX application code previously identified by the state-of-the-art tool EnclaveFuzz, as well as 6 previously undiscovered vulnerabilities. These results highlight the importance of hardware-based fuzzing in securing SGX applications.**

I. INTRODUCTION

Today's commercial processors natively support trusted execution environments (TEEs) to safeguard data in use from unauthorized access [1]–[3]. A prime example is Intel's Software Guard eXtensions (SGX) [4]–[7], which establishes a hardware-isolated environment, called *enclave*, specifically designed to be non-introspectable to prevent data leaks.

Despite hardware-enforced protections, SGX applications remain vulnerable to software flaws that can undermine system security [8]–[10]. Many are written in C/C++ languages, which are prone to memory vulnerabilities like buffer overflow and use-after-free. For instance, a heap or stack overflow in mbedTLS-SGX can be exploited to leak the entire enclave memory by manipulating untrusted inputs [10]. Even seemingly minor issues like null pointer dereference can become critical in SGX environments, as the untrusted OS controls the zero address [11]. Moreover, SGX-specific malware poses additional threats, such as stealing RSA keys from other enclaves [12] or encrypting sensitive data to restrict access [13].

The fundamental cause of these vulnerabilities often stems from developer-introduced flaws, either intentional or unintentional, and untrusted third-party libraries. For example, developers unfamiliar with SGX requirements may inadvertently introduce vulnerabilities into SGX applications [14]–[17]. On the other hand, legacy code is frequently ported into SGX enclaves [18]–[23] with limited validation, which inherits existing security flaws (e.g., OpenSSL library) or introduces new enclave-specific risks. These issues highlight the need for efficient detection mechanisms tailored to SGX applications.

Recent studies have introduced several tools to detect vulnerabilities in SGX applications [10], [11], [24]–[33]. Among them, fuzzing is more effective for vulnerability detection, as it automatically generates diverse inputs to rigorously test applications. Nevertheless, existing SGX fuzzing tools still exhibit notable limitations: 1) Lack of *actual* hardware fuzzing. Existing approaches like SGXFuzz [24] and EnclaveFuzz [11] perform fuzzing exclusively in simulated enclave environments, which lack essential SGX-specific protections and features, such as remote attestation and sealing/unsealing. This limits detection accuracy and scope. Further, SEnFuzzer [26] runs programs within actual SGX enclaves but collects only selective runtime data, which leads to insufficient analysis and potential false positives. 2) Incomplete application behavior analysis. Current methods focus primarily on enclave boundary interfaces (e.g., ECALLs or/and OCALLs) [10], [11], [24] and often overlook internal application bugs. For performance and security purposes [9], [34], SGX applications are often designed to use a single ECALL-OCALL pair to ensure that execution completes without intermediate exits or re-entries. As a result, the absence of runtime context information makes analyzing the application code challenging. 3) High overhead of crash detection on SGX. Exception handling in SGX enclaves is resource-intensive, which complicates efficient crash detection in hardware-based fuzzing. Current tools avoid this issue, as they typically run in simulated environments without SGX exception handling costs. This affects the detection speed and accuracy on actual hardware.

In this paper, we introduce INTRAFUZZ, an automated framework for full-process fuzzing of SGX applications directly within actual hardware enclaves. We adapt a coverage-guided fuzzing framework for SGX, which leverages a modified library OS as the runtime. This runtime transparently manages interactions between the target application and the fuzzing engine. INTRAFUZZ employs an intra-enclave tracer to monitor each fuzzing process and iteratively generates refined inputs through seed mutation. This process updates the corpus with high-quality seeds to enable efficient crash detection and vulnerability identification during program execution. Further, it incorporates an efficient process management mechanism based on in-enclave shared memory, which allows rapid runtime information exchange across processes. To handle hardware exceptions caused by program crashes, we set up a signal handling trampoline to promptly capture crashes and analyze them by parsing enclave stack memory traces.

We evaluate INTRAFUZZ on 21 real-world SGX applications running on Intel Xeon scalable processors, with up to 256 GB of enclave page cache. INTRAFUZZ successfully detected all vulnerabilities in SGX application code previously reported by the state-of-the-art tool EnclaveFuzz, along with 6 previously undiscovered vulnerabilities.

Detection Scope. This work focuses on fuzzing SGX application code within actual enclaves, rather than the host-enclave

*Corresponding author: Jiliang Zhang

boundary in simulated environments, which prior studies have extensively explored. Thus, it departs from existing methods and complements efforts to secure hardware enclaves.

Our contributions are summarized as follows:

- We conduct a systematic analysis of differences and challenges between simulated and *actual* hardware enclaves.
- We propose the first full-process intra-enclave fuzzing framework for SGX applications, called INTRAFUZZ, which enables internal tracers to monitor security violations in target application code.
- We evaluate INTRAFUZZ with 21 open-source projects configured to run with up to 256 GB of enclave page cache, and uncover 9 vulnerabilities in them, including 6 previously undiscovered ones.

II. BACKGROUND

A. Hardware-enforced Enclaves with Intel SGX

Intel Software Guard eXtensions (SGX) [35] is a set of secure instructions integrated into Intel x86 processors. It protects sensitive data and code within an isolated execution environment, known as an enclave, from the rest of the system.

Enclave Entry/Exits. SGX introduces specialized instructions to control the interaction between the enclave and non-enclave code. In particular, the `EENTER` instruction manages entry into the enclave (ECALLs), while `EEXIT` controls the exit (OCALLs). The processor validates the ECALL by checking that it points to an authorized entry located in the enclave's metadata. A predefined wrapper function is used to handle the OCALL request and securely return to the enclave. However, note that for performance and security consideration [9], [34], developers often utilize a single ECALL-OCALL pair for SGX applications to guarantee that execution completes without extra exits or re-entries.

Exception Handling in SGX Enclaves. When an exception occurs within an enclave, the hardware first triggers an Asynchronous Enclave Exit (AEX) to hand over control to the OS. After the OS receives the exception, it sends the corresponding signal to the host process. The host process re-enters the enclave via `EENTER`, and the exception handler within the enclave reads the CPU state in the Saved State Area (SSA) and executes the default or user-defined handling code. Once completed, the enclave exits via `EEXIT`, and the host process then resumes normal execution of the enclave via `ERESUME`. This mechanism ensures the secure saving and restoring of the enclave's state during exception handling.

B. Coverage-guided Fuzzing

Fuzzing is an automated technique for identifying software vulnerabilities by repeatedly executing the target program with diverse inputs [36]. Coverage-guided fuzzing refines this process by utilizing coverage feedback to select inputs that explore more new execution paths. Tools like Honggfuzz [37] compute coverage based on metrics such as executed instructions, branches, and unique basic blocks and retain test cases that uncover new code as good seeds. Prior SGX fuzzing methods [11], [24]–[26] primarily perform in non-SGX environments, where feedback is collected outside the enclave. This often misses critical runtime states within enclaves and hence causes overlooked vulnerabilities and false positives. Moreover, these methods mainly focus on testing enclave boundaries, such as ECALLs, OCALLs, and system calls. In contrast, our work emphasizes identifying vulnerabilities within the security-critical application code inside enclaves. This complements existing methods, collectively offering a more comprehensive strategy for securing SGX enclaves.

III. ANALYSIS OF SGX ENCLAVE FUZZING

A. Simulated vs. Actual Hardware Enclaves

Intel offers a software simulation environment that enables SGX functionality on machines without native SGX hardware support [38]. In simulation mode, SGX instructions are fully emulated through software. Since fuzzing depends on executing target programs and monitoring their runtime behavior to uncover vulnerabilities, the discrepancies between simulated and actual hardware environments can substantially diminish the reliability and effectiveness of fuzzing results.

D1. Security Gaps. Unlike actual hardware-based SGX, which provides strong security guarantees, simulated enclaves lack these protections that leaves them vulnerable to attacks. As a result, attackers can bypass software-based protections through various methods to steal sensitive data. This means fuzzing SGX applications in actual hardware enclaves can more accurately evaluate their security, as simulated environments fail to replicate real-world threats and risks.

D2. Library Discrepancies. Simulated and real hardware environments differ significantly in implementation. In hardware mode, SGX APIs interact with hardware features through dedicated libraries, such as `sgx_trts.lib`. In contrast, simulation mode relies on alternative libraries, such as `sgx_trts.lib_sim`, to emulate this functionality. Similarly, `sgx_tcrypto_opt.lib`, a hardware-optimized cryptographic library, provides enhanced performance on actual SGX hardware, while `sgx_tcrypto.lib` is a basic library intended for simulation environments.

D3. Functional Gaps. Certain operations, such as remote attestation, require hardware support and are unavailable in simulation mode. As a result, fuzzing conducted in a simulated environment produces less accurate results and may fail to fully reflect the effectiveness of tests on actual hardware. In contrast, hardware-based fuzzing provides more reliable and representative outcomes. While SEnFuzzer [26] introduces an on-device fuzzing framework, it still relies on extracting ECALL and OCALL data to construct a fuzzing harness, without running the application throughout SGX enclaves.

Consequently, we emphasize that fuzzing SGX applications on actual hardware is essential for uncovering vulnerabilities that may not manifest in simulated environments and for accurately reflecting real-world attack surfaces.

B. Challenges of Fuzzing on Actual Intel SGX CPUs

C1. Constraints on Intra-enclave Multi-process Fuzzing. Intel SGX imposes strict limits on the size of protected memory, typically 128 MB or 256 MB [4]. As such, the number of enclaves that can be instantiated simultaneously on a single platform is constrained by the memory footprint of each enclave. In addition, fuzzing, which typically employs a multi-process model for efficiency, demands real-time execution and monitoring across multiple processes. However, hardware enclaves lack native support for multitasking within a single

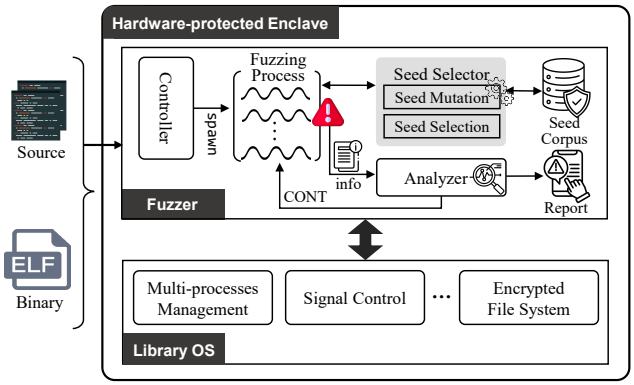


Fig. 1: INTRAFUZZ Overview

enclave. Running multiple enclaves for parallel fuzzing processes drastically reduces efficiency due to the high overhead of inter-enclave communication.

C2. Obstacles to Intra-enclave Monitoring. The security design of Intel SGX ensures that enclaves are isolated from one another and non-introspectable to the outside world. Consequently, conventional monitoring tools cannot directly access or inspect an enclave's internal operations. This inherent opacity poses significant challenges for detecting program crashes, tracking coverage information, and applying other fuzzing techniques within the SGX ecosystem.

C3. Costly Enclave I/O Operations. Communication across enclave boundaries is limited to the interfaces defined in an EDL file, namely ECALLs and OCALLs. This increases complexity and delays input mutation and output reporting during fuzzing. Further, since the enclave cannot directly access the file system, all file operations must be carried out through OCALLs. This affects the overall performance of the tests, particularly when frequent file reads and writes are required during fuzzing.

IV. INTRAFUZZ APPROACH

A. Overview

As illustrated in Figure 1, INTRAFUZZ enables intra-enclave fuzzing of SGX applications within actual SGX environments. The process starts with preparing the instrumented source code or binary file and initial seeds. At the beginning of fuzzing, the *seed selector* mutates the initial seed. The *controller* `clones` worker threads and `spawns` enclave processes to execute the target application with mutated inputs. When a crash occurs, the *analyzer* parses the crashed stack and generates a detailed report. During each iteration, the seed selector updates a seed corpus with high-quality inputs that either trigger vulnerabilities or explore new code paths. The *library OS* provides basic services, such as system call interfaces and an encrypted file system, to facilitate intra-enclave fuzzing. It employs in-enclave shared memory for process management, which enables precise control over the target program's execution state and efficient delivery of runtime information. INTRAFUZZ further integrates an exception-based monitoring mechanism to detect unexpected program behaviors and utilizes enclave stack traces for crash diagnostics.

To overcome the three challenges outlined in Section III-B, we 1) harness an SGX library OS to enable multi-process fuzzing inside a single enclave on SGX v2 with up to 256 GB of enclave private memory, which effectively supports an unlimited number of concurrent enclaves; 2) propose an intra-enclave multi-process management module to monitor program behaviors and gather runtime information; 3) incorporate an encrypted file system into SGX enclaves to enable secure and efficient seed file access during fuzzing.

B. Efficient Multi-process Management

Since the fuzzer process requires control over the target process and the collection of its runtime information, communication between the processes is essential. However, the fuzzer and target processes are isolated from each other due to the enforcement of multi-domain Software Fault Isolation (SFI) technology [39]. To address this limitation, we design a shared memory mechanism between the processes. Each block of shared memory is 2048 bytes in size and divided into three fields: `exec` (256 bytes), `info` (80 bytes), and `stack` (the left space). The `exec` field records execution status, the `info` captures details about abnormal signals and the `stack` stores information for stack backtracking.

As shown in Figure 2, the target process communicates its runtime state to the fuzzer process via shared memory. During initialization, the fuzzer creates N shared memory blocks, where N is the number of parallel processes specified by the user. To be specific, the main fuzzing process first clones N worker threads. Each worker thread then creates a child process using `spawn` to execute the target program. To organize this setup, the parent worker thread and its child process are grouped into a `workgroup`, with the parent thread's *TID* indexing the corresponding memory block to obtain runtime information. This design adheres to SGX's memory constraints while ensuring that each process operates securely within actual enclaves.

C. Enclave Crash Monitoring

When a crash occurs in the SGX application code, INTRAFUZZ must interpose the target process to collect runtime context information for detailed analysis. However, since each crash causes the current process to exit the SGX enclave—triggering an asynchronous enclave exit (AEX) exception—it is essential to restore control flow to the target process. To address this, we propose a signal handling based implementation, as shown in Figure 3. During the instrumentation of target program, a custom signal handler is registered (❶). When an exception occurs, this handler is invoked to collect crash details and store runtime context information into shared memory (❷). The analyzer subsequently reads this information, identifies the type of crash, and determines whether it represents a previously unseen issue (❸). If a new crash is detected, the analyzer generates a report and saves the seed (❹).

After completing the analysis, the analyzer writes `cont` signal to the `exec` field in shared memory to inform the target program to continue execution (❺). The target program then resumes from the point of the crash, which will trigger the same signal (e.g., SIGSEGV) and eventually exiting due to the unhandled exception. At the end of each iteration, the controller spawns a new child process to start the next iteration. This loop continues until a termination condition is reached, at which point the fuzzing process ends.

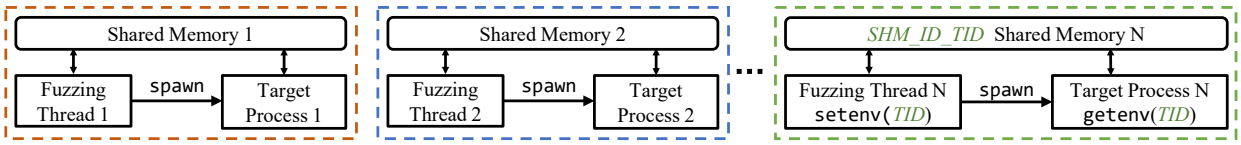


Fig. 2: Efficient Process Management via Shared Memory

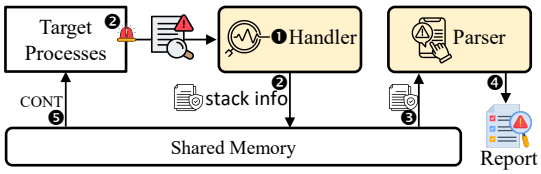


Fig. 3: Exception-based Monitoring Workflow

D. Enclave Crash Analyzing

When the target process crashes, gathering program details is necessary for analysis. However, pinpointing the root cause of a crash can be challenging. To address this, we introduce an adaptive enclave stack memory parsing mechanism within the analyzer. Upon receiving a signal, it dynamically analyzes runtime information using memory mappings, call stack traces, and instruction disassembly. Specifically, it extracts essential details from the signal structure (`siginfo_t`) and the program counter (`PC`) , which are stored in the `info` field of shared memory. It then leverages the `libunwind` library to trace the call stack and parses the `/proc/[pid]/maps` file to retrieve memory mapping details. Finally, we leverage Capstone engine to disassemble instructions in memory. Based on these above, IntraFuzz delivers real-time, detailed insights into the execution path and instruction-level information.

Spawn-based process creation introduces the challenge of identical instruction addresses mapping to different memory addresses, which makes the address alone insufficient to uniquely identify crashes. Despite each child process operating in a separate virtual space, the layout of code segments within the virtual address space is consistent across processes. Thus, crashes at the same instruction exhibit identical lower 12 bits (in-page offset) of the virtual address, which reflects the instruction's position in the code segment. Based on this, we define crash uniqueness using four key attributes: (i) the signal type, (ii) the lower 12 bits of the instruction address, (iii) the current instruction, and (iv) a hash value of the call stack.

V. Implementation

We implemented IntraFuzz prototype by extending honggfuzz [37], a lightweight, feedback-driven fuzzer, to work within advanced SGX runtime systems such as Occlum [18]. IntraFuzz retains the core components of honggfuzz, including the mutation algorithm, and test case scheduling. To enable intra-enclave fuzzing, we developed several key modules, such as multi-process management, signal control and enclave stack memory parsing. Due to SGX's isolation constraints, the `fork` system calls in honggfuzz are not supported within a single enclave. To resolve this, we modified the controller to use `spawn` for creating enclave processes. Moreover, the target program and initial seed are stored in the encrypted file system to facilitate the fuzzing.

VI. Evaluation

In this section, we evaluated IntraFuzz on real-world SGX applications to demonstrate its effectiveness in detecting vulnerabilities. We also compare IntraFuzz with existing state-of-the-art SGX vulnerability detection techniques and analyze their methodologies. Finally, we highlight the vulnerabilities newly uncovered in the target SGX applications.

A. Evaluation Settings

Experiment Setup. The evaluation was conducted on a server equipped with four Intel Xeon Platinum 8444H CPUs, 512 GB of RAM, and 256 GB of enclave page cache, running Ubuntu 20.04 with the 5.15.0-124-generic kernel. Each project was fuzzed for over 24 hours.

Tested SGX Applications. We selected 20 widely recognized open-source projects with notable star ratings on GitHub. Table I presents the fuzzing results for these SGX applications, including the size, coverage, and verified bugs. The specific versions of these applications were adapted to be compatible with IntraFuzz.

B. Vulnerability Detection

IntraFuzz detected hundreds of crashes across 7 applications during fuzzing. Since a single bug can lead to multiple crashes, deduplication is required. We applied an intra-enclave adaptive stack trace parsing mechanism to identify unique crashes, followed by manual analysis. This process revealed 9 distinct and verified bugs, as shown in Table I.

Detail of Reports. Table II shows a detailed summary of the bugs identified by IntraFuzz and other works. Two buffer overflow vulnerabilities were discovered in SGXCryptoFile and cave, both caused by input file sizes exceeding expected limits. These vulnerabilities were uniquely detected by our approach, as previous methods failed to preserve the original semantics of the applications, which is essential for effective fuzzing. In sgx-wallet, a double-free vulnerability was found. In webassemblyEnclave, a logic error was identified, where incorrect control flow leads to unintended execution branches. In sgx-kmeans, an out-of-bound access error was observed due to insufficient validation before data allocation. Detailed explanations of these vulnerabilities are provided in Section VI-D. For clarity, Table II excludes crash counts associated with boundary interface issues for SGXFuzz and EnclaveFuzz.

To validate the effectiveness of detecting vulnerabilities in the remote attestation process, which is not supported in simulated enclaves, we introduced a fault into a confidential function in the program. Experimental results demonstrated that IntraFuzz successfully identified the vulnerability.

Coverage. Table I shows that IntraFuzz achieves comparable coverage to EnclaveFuzz. While EnclaveFuzz analyzes the source code without considering contextual semantics, IntraFuzz, by fully preserving the context of SGX applications, results in improved coverage. Specifically, EnclaveFuzz achieves 14.8% coverage, whereas IntraFuzz reaches 17.7%.

TABLE I: Fuzzing Results of IntraFuzz and EnclaveFuzz

SGX Application	Version	Size	Coverage		Verified Bugs	
			EnclaveFuzz	IntraFuzz	EnclaveFuzz	IntraFuzz
SGXCryptoFile_Encrypt [40]	92f3cd6	520.7KB	18.5%	22.7%	1⋆	0
SGXCryptoFile_Decrypt [40]	92f3cd6	520.7KB		13.1%		1
sgx-gwas [41]	24e59f3	1.8MB	0.05%	0.08%	3◇	0
SGX-DFC [42]	d3f7e47	657.4KB	1.2%	13.7%	0	0
sgx-kmeans [43]	8ab6035	618.0KB	16.8%	24.3%	5◇	2
sgx-wallet [44]	8d15df1	59.0KB	36.9%	8.2%	1⋆+ 1◇	1
sgx-GuessTheNumber [45]	69bf5b3	577.6KB	26.5%	23.1%	1◇	0
SGX-AES-256 [46]	b0d87a8	707.6KB	–	2.9%	–	0
sgx-nbench [47]	799f0fc	486.8KB	22.3%	36.2%	1⋆+ 9◇	1
SwitchlessSGXSan [48]	d0778b9	474.0KB	–	22.9%	–	0
SampleSGXSan [49]	71c4227	497.0KB	20.7%	14.6%	3◇	0
sgx-dnet [50]	6e18300	1.8MB	2.4%	0.2%	1◇	0
webassemblyEnclave_Add [51]	2281ac9	1.3MB	15.3%	3.8%	8◇	1
webassemblyEnclave_Mul [51]	2281ac9	1.3MB		3.6%		1
sgxvpn [52]	7e61a63	469.0KB	26.0%	12.5%	5◇	0
ucmalloc-sgx [53]	3e2cdeb	492.5KB	6.2%	20.0%	–	0
mex-sgx [54]	429473e	424.1MB	5.3%	63.4%	–	0
sgxBankApp [55]	8637537	796.8KB	28.6%	2.8%	0	0
cave [56]	bb8ef3a	1.9MB	5.5%	3.6%	3◇	1
MatrixChainOrder [57]	1952bce	497.1KB	3.9%	64.1%	0	0
Attestation-test [58]	Crafted	463.5KB	–	16.0%	–	1
Total Bugs & Average Coverage			14.8%	17.7%	3⋆	9

⋆ indicates bugs found within security-critical application code. ◇ means bugs located at the host-enclave boundary. – denotes fuzzing failures under the original configurations.

TABLE II: Unique Crashes Identified across Different Works

SGX Application	Unique Crashes in SGX Application		
	SGXFuzz	EnclaveFuzz	IntraFuzz
SGXCryptoFile [40]	0	1	1
sgx-kmeans [43]	0	0	2
webassemblyEnclave [51]	0	0	2
cave [56]	0	0	1
sgx-wallet [44]	1	1	1
sgx-nbench [47]	1	1	1
Total crashes found	2	3	8

TABLE III: Comparison to Existing Methods

Name	Technique	Target	Is Testing Conducted within Actual Enclaves?
TEERex [28]	Symbolic Execution	Enclave Boundary	✗
COIN [10]	Symbolic Execution	Enclave Boundary	✗
SymGx [30]	Symbolic Execution	ECALLs & OCALLs	✗
Pandora [31]	Symbolic Execution	ECALLs & OCALLs	✗
SGXFuzz [24]	Fuzzing	ECALLs	✗
SEnFuzzer [26]	Fuzzing	ECALLs & OCALLs	✘
FuzzSGX [25]	Fuzzing	ECALLs & OCALLs	✗
EnclaveFuzz [11]	Fuzzing	ECALLs & OCALLs	✗
Emilia [27]	Fuzzing	System Calls	✗
IntraFuzz (this work)	Fuzzing	SGX Application Code	✓

✗ indicates fuzzing conducted entirely within simulated enclave environments. ✘ means that SGX applications run inside actual enclaves, while the fuzzing process is performed outside the enclave. ✓ refers to fuzzing conducted entirely within actual, hardware-protected enclaves.

Responsible Disclosure. When a bug is detected and manually verified, we adhere to a 90-day disclosure timeline and notify the affected vendor via email or repository issue.

C. Comparison with State-of-the-art Works

We compare advanced SGX vulnerability detection tools with IntraFuzz based on their detection targets and fuzzing environments, as summarized in Table III. Unlike existing tools that primarily focus on boundary interfaces such as ECALLs, OCALLs and syscalls, IntraFuzz targets intra-enclave application code. This focus enables it to uncover vulnerabilities that prior methods might overlook and expose critical code segments previously unexamined. IntraFuzz is the first to implement full-process fuzzing directly on SGX hardware. This offers comprehensive monitoring of the actual runtime behavior within hardware enclaves. In reproducing EnclaveFuzz, we observed that it requires manual source code modifications for instrumentation. In contrast, IntraFuzz automates this process.

D. Case Study

We showcase representative examples of each bug type identified by IntraFuzz.

```
1  int decryptFile(const char* input, const char* output)
2  {
3    ...
4    size_t decMessageLen = fsize - (SGX_AESGCM_MAC_SIZE +
5    SGX_AESGCM_IV_SIZE);
6    unsigned char *decMessage = (unsigned char *) malloc
7    ((decMessageLen+1)*sizeof(unsigned char));
8    sgxDecryptFile(message,fsize,decMessage,decMessageLen);
9    decMessage[decMessageLen] = '\0';
10   fwrite(decMessage, decMessageLen, 1, ofp);
11 }
```

Listing 1: Stack Overflow in CryptoFile

Stack Overflow. Several crashes caused by buffer overflow were found, with SGXCryptoFile as a notable example. As shown in Listing 1, if the input size (`fsize`) is smaller than `SGX_AESGCM_MAC_SIZE+SGX_AESGCM_IV_SIZE`, the calculation of `decMessageLen` results in an underflow (line 4) and produces an unexpected large value `decMessageLen`. This value is then passed to `malloc` (line 6), which causes excessive memory allocation. If the allocation succeeds, operations like writing to `decMessage` (line 10) will exceed the allocated buffer. This leads to memory corruption and hence may overwrite critical memory areas. More severely, this vulnerability may enable attackers to perform arbitrary writes, which compromise the confidentiality and integrity of the enclave.

```
1  cell eval(cell x, environment * env)
2  {
3    if (x.type == Symbol)
4      return env->find(x.val)[x.val];
5    ...
6  }
```

Listing 2: Logical Execution Vulnerability in CAVE

Logical Execution. Listing 2 illustrates a logical execution vulnerability in cave. The behavior of `eval`, which includes

multiple branches, can be directly influenced by the input data. This vulnerability allows attackers to manipulate the input and alter the program's execution flow within the enclave.

```
1  int* k_means(float** data, int n, int m, int k, float t,
       float** centroids)
2  {
3    ...
4    for (h = i = 0; i < k; h += n / k, i++) {
5      if (!centroids) {
6        c[i] = (float * ) calloc(m, sizeof(float));
7      }
8      /* pick k points as initial centroids */
9      for (j = m; j-- > 0; c[i][j] = data[h][j]);
10   }
11   ...
12   for (i = 0; i < k; i++) {
13     float distance = 0;
14     for (j = m; j-- > 0; distance += pow(data[h][j] -
15         c[i][j], 2));
16     ...
17   }
18 }
```

Listing 3: Out-of-bound Access in SGX-KMEANS

Out-of-bound Access. Out-of-bound access occurs when a program accesses data outside its intended boundaries, which may lead to memory corruption, crashes, or unpredictable behavior. In Listing 3, the issue arises when `data[h][j]` is inadequate for assignment to `c[i][j]`. The operation $\texttt{pow}(\texttt{data}[\texttt{h}][\texttt{j}] - \texttt{c}[\texttt{i}][\texttt{j}], 2)$ may fail due to the improper assignment. Specifically, if `data[h][j]` is not properly validated before the assignment, it may result in out-of-bound access when subtracting `c[i][j]`. This mis-assignment causes the program to access memory beyond the allocated bounds, which finally triggers undefined behavior or memory corruption.

VII. Discussion

IntraFuzz automatically detects memory corruption vulnerabilities in Intel SGX applications. However, some memory vulnerabilities may not cause crashes. For example, a missing bounds check in the handling of the TLS heartbeat [59] extension leads to an out-of-bound read across the enclave boundary, which ultimately reveals up to 64 KB of memory data to a connected client or server without triggering a crash. While sanitizers [60], [61] can address these issues, implementing them within the SGX environment requires much engineering effort, which we leave for future work. Moreover, several solutions for SGX compatibility have been proposed [18], [19], but we cannot perform vulnerability detection on these toolchains, as our work primarily focuses on discovering vulnerabilities in SGX application code itself. For such toolchains, fuzzing would likely require the generation of specialized fuzzing harnesses.

VIII. Related Work

A. Fuzzing for SGX Applications

SGXFuzz [24] simulates key SGX execution conditions and uses the kAFL fuzzer along with the Nyx snapshot-based fuzzing engine to recreate critical enclave contexts outside the SGX environment. FuzzSGX [25] is a fuzzer based on input and program mutations. It applies static analysis and symbolic execution to construct the host application and modifies both the application and seed inputs iteratively to evaluate the enclave program. SEnFuzzer [26] extracts ECALL and OCALL information from the EDL files to create specialized fuzzing drivers. Despite executing the enclave program within an actual SGX environment, the fuzzer runs outside the enclave and only focuses on the ECALL and OCALL interfaces. EnclaveFuzz [11] presents a multi-dimensional structure-aware fuzzing framework that extracts input structures from enclave source code and analyzes their inter-dependencies to generate fuzz harnesses. Emilia [27] intercepts system call return values to reproduce potential Iago attacks [34] through fuzzing. However, it operates outside actual SGX environments and is limited to the return values, which results in reduced coverage. In summary, existing SGX fuzzing methods analyze the enclave boundaries. However, none of them perform full-process fuzzing within actual SGX enclaves, leaving real vulnerabilities undetected, as demonstrated by IntraFuzz.

B. Symbolic Execution for SGX Applications

TEERex [28] takes a binary file as input, identifies ECALLs through static analysis, and then uses symbolic exploration to detect vulnerabilities during execution. COIN [10] identifies security vulnerabilities in enclave interfaces (ECALLs/OCALLs) and performs symbolic execution on binary files by parsing interface information from EDL files. SymGX [30] introduces a global transformation graph and context-sensitive analysis model, GSTG-CAP, for simulating SGX applications. It records execution behaviors and drives symbolic execution for vulnerability detection. Pandora [31] uses the angr framework with SGX-specific semantic augmentation. It extracts the enclave's internal state to guide symbolic execution, with plugins designed to detect different vulnerabilities.

C. Others Techniques for Shielding SGX Enclaves

SgxMonitor [32] introduces a security tracking mechanism akin to Intel PT, which implements provenance analysis on SGX. It monitors the target enclave's operating status to ensure it aligns with expectations and identifies any discrepancies. SGXRacer [33] detects controlled data races in enclave code by analyzing shared variables and thread interleavings to ensure proper synchronization. It employs a two-phase approach: identifying shared and lock variables, followed by reentrancy-aware, lockset-based data race detection.

IX. Conclusion

We introduce IntraFuzz, a new system for efficiently fuzzing SGX applications within actual hardware-protected enclaves. Using an internal tracer, IntraFuzz monitors intra-enclave processes to detect security violations with mutated inputs. Our results show IntraFuzz identified 9 vulnerabilities across 21 real-world SGX applications on actual Intel SGX processors, including 6 vulnerabilities missed by simulation-based tools like EnclaveFuzz. These findings highlight the importance of actual hardware-based fuzzing for comprehensive security analysis of SGX applications.

X. Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. U24A20289, 62402169), the Key R&D Projects in Hunan Province (Grant No. 2024JK2011), the Natural Science Fundation of Hunan Province (Grant No. 2023JJ40160), the Excellent Young Scholar Project of the Hunan Provincial Department of Education (Grant No. 23B0036), the CCF-Huawei Populus Grove Fund (CCF-HuaweiTC202402).

REFERENCES

[1] Intel, "Microsoft azure confidential computing with intel software guard extensions," https://www.intel.com/content/www/cn/zh/developer/articles/technical/microsoft-azure-confidential-computing-with-intel-sgx.html, 2018.

[2] J. Associates, "The critical need for confidential computing," https://www.intel.cn/content/dam/www/central-libraries/cn/zh/documents/the-critical-need-for-confidential-computing.pdf, 2020.

[3] "Alibaba cloud ecs bare metal instance supports intel sgx," https://www.alibabacloud.com/zh/product/ebm?_p_lc=1&spm=a2c5t.1069566.

[4] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *HASP@ISCA*, 2013.

[5] B. C. Xing, M. Shanahan, and R. Leslie-Hurd, "Intel® software guard extensions (intel® sgx) software support for dynamic memory allocation inside an enclave," *HASP*, pp. 1–9, 2016.

[6] Intel, "Intel software guard extensions sdk-documentation," https://software.intel.com/en-us/sgx-sdk/documentation.

[7] V. Costan, "Intel sgx explained," *IACR Cryptol, EPrint Arch*, 2016.

[8] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *ACM SIGSAC CCS*, 2019, pp. 1741–1758.

[9] J. Cui, J. Z. Yu, S. Shinde, P. Saxena, and Z. Cai, "SmashEx: Smashing sgx enclaves using exceptions," in *ACM SIGSAC CCS*, 2021, pp. 779–793.

[10] M. R. Khandaker, Y. Cheng, Z. Wang, and T. Wei, "COIN attacks: On insecurity of enclave untrusted interfaces in sgx," in *ASPLOS*, 2020, pp. 971–985.

[11] L. Chen, Z. Li, Z. Ma, Y. Li, B. Chen, and C. Zhang, "EnclaveFuzz: Finding vulnerabilities in sgx applications," in *NDSS*, 2024.

[12] M. Schwarz and M. Lipp, "When good turns evil: Using intel sgx to stealthily steal bitcoins," *Black Hat Asia*, 2018.

[13] M. Marschalek, "The wolf in sgx clothing," *Bluehat IL*, 2018.

[14] M. Schwarz, S. Weiser, and D. Gruss, "Practical enclave malware with intel sgx," in *DIMVA*, 2019, pp. 177–196.

[15] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, "Hacking in darkness: Return-oriented programming against secure enclaves," in *USENIX Security*, 2017, pp. 523–539.

[16] S. Van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "Sgaxe: Dfc string pattern matching library for sgx application," https://github.com/sparkly9399/SGX-DFC, 2020.

[17] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves," in *ESORICS*, 2016, pp. 440–457.

[18] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and efficient multitasking inside a single enclave of intel sgx," in *ASPLOS*, 2020, pp. 955–970.

[19] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *USENIX ATC*, 2017, pp. 645–658.

[20] J. Cui, S. Shinde, S. Sen, P. Saxena, and P. Yuan, "Dynamic binary translation for sgx enclaves," *ACM Transactions on Privacy and Security*, vol. 25, no. 4, pp. 1–40, 2022.

[21] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves." in *NDSS*, 2017.

[22] MICROSOFT, "Open enclave sdk," https://openenclave.io/sdk/.

[23] ASYLO, "Asylo: An open and flexible framework for enclave applications," https://asylo.dev/.

[24] T. Cloosters, J. Willbold, T. Holz, and L. Davi, "SGXFuzz: Efficiently synthesizing nested structures for sgx enclave fuzzing," in *USENIX Security*, 2022, pp. 3147–3164.

[25] A. Khan, M. Zou, K. Kim, D. Xu, A. Bianchi, and D. J. Tian, "Fuzzing sgx enclaves via host program mutations," in *IEEE EuroS&P*, 2023, pp. 472–488.

[26] D. Yu, J. Wang, H. Fang, Y. Fang, and Y. Zhang, "SEnFuzzer: Detecting sgx memory corruption via information feedback and tailored interface analysis," in *RAID*, 2023, pp. 485–498.

[27] R. Cui, L. Zhao, and D. Lie, "Emilia: Catching iago in legacy code." in *NDSS*, 2021.

[28] T. Cloosters, M. Rodler, and L. Davi, "TeeRex: Discovery and exploitation of memory corruption vulnerabilities in sgx enclaves," in *USENIX Security*, 2020, pp. 841–858.

[29] P. Antonino, W. A. Woloszyn, and A. Roscoe, "Guardian: Symbolic validation of orderliness in sgx enclaves," in *CCSW*, 2021, pp. 111–123.

[30] Y. Wang, Z. Zhang, N. He, Z. Zhong, S. Guo, Q. Bao, D. Li, Y. Guo, and X. Chen, "Symgx: Detecting cross-boundary pointer vulnerabilities of sgx applications via static symbolic execution," in *ACM SIGSAC CCS*, 2023, pp. 2710–2724.

[31] F. Alder, L.-A. Daniel, D. Oswald, F. Piessens, and J. Van Bulck, "Pandora: Principled symbolic validation of intel sgx enclave runtimes," in *IEEE (S&P)*, 2024.

[32] F. Toffalini, M. Payer, J. Zhou, and L. Cavallaro, "Designing a provenance analysis for sgx enclaves," in *ACSAC*, 2022, pp. 102–116.

[33] S. Chen, Z. Lin, and Y. Zhang, "Controlled data races in enclaves: Attacks and detection," in *USENIX Security*, 2023, pp. 4069–4086.

[34] S. Checkoway and H. Shacham, "Iago attacks: Why the system call api is a bad untrusted rpc interface," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 253–264, 2013.

[35] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual," *Volume 3B: System programming Guide, Part*, vol. 2, no. 11, pp. 0–40, 2011.

[36] "LibFuzzer," https://llvm.org/docs/LibFuzzer.html.

[37] R. Swiecki, "Honggfuzz," http://code.google.com/p/honggfuzz.

[38] Intel, "How to run intel® software guard extensions' simulation mode," https://www.intel.com/content/www/us/en/developer/articles/training/usage-of-simulation-mode-in-sgx-enhanced-application.html.

[39] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in *SOSP*, 1993, pp. 203–216.

[40] rscosta, "Sgxcryptofile - app for encrypting and decrypting files using intel sgx," https://github.com/rscosta/SGXCryptoFile.

[41] Nazmus-Sadat, "Trusted hardware assisted genome-wide association studies," https://github.com/Nazmus-Sadat/sgx-gwas.

[42] sparkly9399, "Dfc string pattern matching library for sgx application," https://github.com/sparkly9399/SGX-DFC.

[43] Dsc-sgx, "sgx-kmeans," https://github.com/dsc-sgx/sgx-kmeans.

[44] Asonnino, "Simple password-wallet application based on intel sgx for linux," https://github.com/asonnino/sgx-wallet.

[45] jeflora, "guess-the-number." https://github.com/jeflora/sgx-demo-app/tree/master/guess-the-number.

[46] Maxul, "Aes-ni crypto for intel sgx," https://github.com/Maxul/SGX-AES-256.

[47] utds3lab, "The nbench benchmark ported to sgx," https://github.com/utds3lab/sgx-nbench.

[48] LeoneChen, "Switchlesssgxsan," https://github.com/LeoneChen/SwitchlessSGXSan.

[49] occlum, "Sampleencalve," https://github.com/occlum/linux-sgx/tree/cd4e96fd2854536b8ad36d58b4a5554203cebf33/SampleCode/SampleEnclave.

[50] lemonviv, "Sgx-darknet: Sgx compatible ml library," https://github.com/lemonviv/sgx-dnet.

[51] SabaEskandarian, "Port of webassembly interpreter to run inside an sgx enclave," https://github.com/SabaEskandarian/webassemblyEnclave.

[52] tyurek, "sgxvpn," https://github.com/tyurek/sgxvpn.

[53] AmbitionXiang, "Sample for allocating and managing untrusted memory inside an enclave," https://github.com/AmbitionXiang/ucmalloc-sgx.

[54] donayam1, "Mex sgx," https://github.com/donayam1/mex-sgx.

[55] R-Niagra, "Bank app built usig linux sgx," https://github.com/R-Niagra/sgxBankApp.

[56] benschreiber, "A lisp interpreter in sgx for use as a secure autograder," https://github.com/benschreiber/cave.

[57] S. Sen, "Intel-sgx-programs," https://github.com/sensatyaki/Intel-SGX-Programs.

[58] intel, "Remoteattestation," https://github.com/intel/linux-sgx/tree/main/SampleCode/RemoteAttestation.

[59] "The heartb donayam1leed bug," https://heartbleed.com/.

[60] K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "Address-sanitizer: A fast address sanity checker," in *USENIX ATC*, 2012, pp. 309–318.

[61] W. Han, B. Joe, B. Lee, C. Song, and I. Shin, "Enhancing memory error detection for large-scale applications and fuzz testing," in *NDSS Symposium*, 2018.

MHDiff: Memory- and Hardware-Efficient Diffusion Acceleration via Focal Pixel Aware Quantization

Chunyu Qi[1], Xuhang Wang[1], Ruiyang Chen[1], Yuanzheng Yao[1], Naifeng Jing[1], Chen Zhang[1], Jun Wang[2]*, Zhihui Fu[2], Xiaoyao Liang[1], and Zhuoran Song[1]*
[1]Shanghai Jiao Tong University, Shanghai, China
[2]OPPO Research Institute
[1]{qichunyu, songzhuoran}@sjtu.edu.cn

***Abstract*—Diffusion models have demonstrated superior performance in image generation tasks, thus becoming the mainstream model for generative visual tasks. Diffusion models need to execute multiple timesteps sequentially, resulting in a dramatic increase in workload. Existing accelerators leverage the data similarity between adjacent timesteps and perform mixed-precision differential quantization to accelerate diffusion models. However, merging differential values with raw inputs in each layer of each timestep to ensure computational correctness requires significant memory access for loading raw inputs, which creates a heavy memory burden. Moreover, mixed-precision computations may lead to low hardware utilization if not well designed. Unlike these works, we propose MHDiff, a tailored framework that identifies the focal pixels at the first layer and finetunes them to fit all layers, then represents focal pixels with high-precision while using low-precision for others, thereby accelerating diffusion models while minimizing memory burden. To improve hardware utilization, MHDiff employs a packing module that merges low-precision values into high-precision values to create full high-precision matrices and designs a processing element (PE) array to efficiently process the packed matrices. Extensive experiment results demonstrate that MHDiff can achieve satisfactory performance with negligible quality loss.**

I. Introduction

Currently, diffusion models have demonstrated superior performance in generative vision tasks. In the field of image generation, diffusion models such as Stable Diffusion [1], DALL-E 2 [2], and Midjourney [3] have achieved remarkable results in various tasks (e.g., image generation [4]–[7], image super-resolution [8]–[10], and image inpainting [11], [12]). Due to their powerful generative capabilities, diffusion models are increasingly becoming the mainstream model for generative visual tasks. The fundamental characteristic of diffusion models is the iterative generation process, which requires the entire model to execute multiple timesteps sequentially. While this characteristic helps produce higher-quality images, it comes at the cost of a dramatic increase in workload, hindering their further application.

To eliminate these problems, researchers have designed various accelerators to accelerate the generation process of diffusion models, such as Cambricon-D [13] and HCDiff [14]. Cambricon-D leverages the data similarity between adjacent timesteps in diffusion models by applying outlier-aware differential quantization to accelerate the generation process. As illustrated in Fig. 1(a), at each layer of every timestep, Cambricon-D retrieves the input data of the previous timestep $T-1$ from DRAM (indicated by the purple arrow) and computes the differentials by subtracting it from the current input data. As a result, the differential data can be computed using mixed-precision. Afterwards, the current result is added by the previous result of timestep $T-1$ (indicated by the purple arrow) to obtain the final result of the current timestep T. Next, it writes the resultant data back to DRAM (indicated by the orange arrow), awaiting for the differential data generation during the next timestep $T+1$. In summary, Cambricon-D faces two limitations: 1) It requires two read operations and one write operation to DRAM at each layer of each timestep, which **creates a heavy memory burden**; 2) It uses different processing elements (PEs) to process high-precision and low-precision differentials separately, which **results in low hardware utilization** since the distribution of high- and low-precision differentials is unpredictable and irregular.

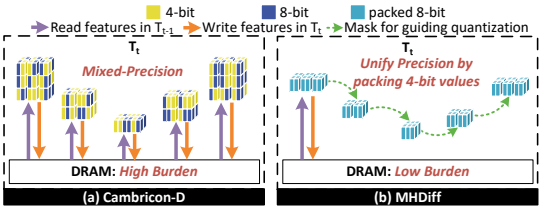


Fig. 1. Comparison of MHDiff and traditional method.

Different from Cambricon-D, our goal is to accelerate diffusion models in a memory-efficient and hardware-efficient manner. In this paper, we observe that the focal pixels to be generated at each timestep play a critical role in image quality, where the focal pixels are defined as the features that change significantly within a timestep. To excavate the acceleration opportunity brought by focal pixels, our study addresses two key challenges: 1) How to identify focal pixels of each timestep in a memory-efficient manner? Given that the focal pixels are consistent across all layers, just like an artist focuses on painting a special area on the canvas for a period of time, we propose a first-predict-then-finetune method. It predicts focal pixels at the beginning of each timestep and then refines them according to the specific operation of each

* Zhuoran Song and Jun Wang are the corresponding authors.
This work is partly supported by the National Natural Science Foundation of China (Grand No. 62202288).

layer, enabling the identified focal pixels to be effectively utilized across all subsequent layers. As shown in Fig. 1(b), compared to Cambricon-D, since we only fetch previous data for prediction at the first layer (indicated by the purple arrow), our approach avoids $n-1$ reads and writes for each timestep given n layers, significantly alleviating the burden on memory. 2) How to accelerate diffusion models based on focal pixels in a hardware-efficient manner? We propose to use high- and low-precision quantization for focal pixels and others respectively, thereby achieving acceleration. Building upon this, we design a specialized packing module that reorganizes the quantized data to pack them into full high-precision matrices. Specifically, based on the principle of avoiding conflicts between high-precision values, the packing module selectively packs adjacent low-precision values into high-precision values as much as possible, thus unifying precision. Therefore, our PE array can process the condensed high-precision matrices at the same speed, resulting in higher hardware utilization.

The main contributions of this paper are listed as follows:

1) On the algorithm level, we propose a MHDiff algorithm, which can identify the focal pixels for each timestep, finetune them according to the specific operation of each layer, and apply high- and low-precision quantization for focal pixels others respectively, thereby achieving acceleration.
2) On the hardware level, we design a MHDiff architecture to support the proposed algorithm. Specifically, MHDiff architecture employs a packing module to pack mixed-precision matrices into full high-precision matrices and use a PE array to process the condensed high-precision data through minor modifications to the PE units.
3) Extensive experiments and in-depth research across various diffusion models consistently validate the advantages of our proposed MHDiff framework, leading to 3.1x, 6.2x, and 38.4x speedup over state-of-the-art diffusion accelerator Cambricon-D, GPU, and CPU while maintaining image quality.

II. Background

A. Diffusion Models

Diffusion models are a new class of machine learning algorithms that have excellent performance in generative vision tasks such as image generation, super-resolution, and image inpainting. Essentially, a diffusion model functions as a denoising model, taking an image composed of Gaussian noise as input and producing a visual image through successive denoising steps. Each denoising step, called a timestep, uses the output from the previous step as input, applying a U-Net model to generate an image with reduced noise. This progressive generation process enables diffusion models to achieve superior image quality compared to previous models. However, this advantage comes at the cost of significantly increased computation and latency, as the model must execute many steps sequentially to produce the final image.

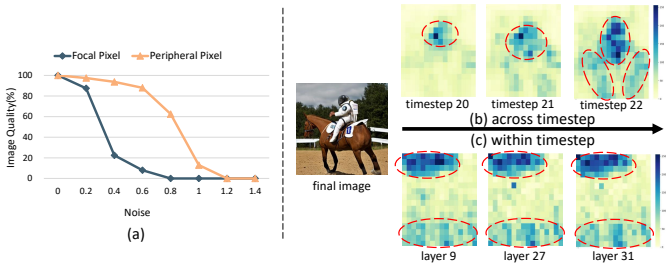


Fig. 2. Explore focal pixels: image quality bearing different noise patterns (a) and the visual focal pixels (marked with red dashed boxes) of different timesteps (b) and different layers (c).

B. Related Work

To address the limitations of diffusion models, researchers have done some work to optimize them from different perspectives, but *they all lack exploration of the relationships between timesteps and the characteristics within each timestep*. On the algorithmic side, DPM-Solver++ [15] introduces a new high-order sampling solver that reduces the number of timesteps required to generate images. Q-Diffusion [16] focuses on model quantization, proposing a timestep-aware post-training quantization method that reduces the activation of diffusion models to 8 bits, thereby accelerating the generation process.

On the hardware side, both Cambricon-D [13] and HCDiff [14] exploit the data similarity between adjacent timesteps in diffusion models by computing the differentials and using outlier-aware quantization to quantize them to a low bitwidth, thus accelerating the inference of diffusion models. Cambricon-D features an outlier-aware PE array to support mixed-precision computations, while HCDiff separates the quantized results into dense normal value matrices and sparse outlier value matrices for separate computation. However, both methods ignore the characteristics within each timestep, thus requiring computing differentials for each layer, which severely increases memory burden. Moreover, they all use different PEs to process high-precision and low-precision differentials separately, which results in low hardware utilization.

III. Motivation

While existing works have achieved notable acceleration, they overlook the existence of focal pixels in diffusion models. To study the impact of focal pixels on image quality, we add different levels of noise to focal pixels and other pixels respectively and measure image quality. As shown in Fig. 2(a), the curve "Focal Pixel" and the curve "Peripheral Pixel" indicate that noise is added to the focus pixels and the same number of other pixels, respectively. We can find that focal pixels are more sensitive to noise. Therefore, we propose a pixel-adaptive quantization method to apply high-precision representation for focal pixels and low-precision representation for others, thereby achieving acceleration while ensuring image quality.

Furthermore, we find that the focal pixels in diffusion models differ across timesteps. To validate this, we compare the change levels of pixels in different timesteps of Stable Diffusion and visualize them using heatmaps. As shown in Fig. 2(b), darker colors represent greater changes, while lighter colors represent smaller changes. We can find that

the number of focal pixels and their corresponding locations across different timesteps vary greatly. Therefore, we propose a prediction method to identify focal pixels for each timestep before applying pixel-adaptive quantization.

Lastly, we discover focal pixels across different layers within the same timestep exhibit consistency. As shown in Fig. 2(c), the focal regions across different layers within the same timestep align closely, indicating the consistency of these regions. Therefore, we propose a prediction finetuning method that performs prediction only once at the first layer and then finetunes the prediction results to efficiently predict focal pixels in subsequent layers, thus minimizing memory burden.

IV. MHDiff Algorithm

A. Focal Pixel Prediction

To identify the focal pixels at each timestep, we propose the focal pixel prediction operation. Given a $H \times W \times C$ input feature map, it identifies whether a $1 \times C$ vector (pixel P) is a focal pixel based on its change level. Specifically, we compute the L_1 distance between the pixel P_{ij}^t at the current timestep t and the corresponding pixel P_{ij}^{t-1} from $t-1$. As shown in Eqn. 1, where i denotes the row coordinate of the pixel, and j denotes the column coordinate. After calculating the L_1 distance, we compare it with a predefined threshold $Bias$. If the L_1 distance exceeds $Bias$, we think this pixel changes significantly and is a focal pixel; otherwise, we think it is a peripheral pixel. In this way, we divide these pixels into two categories: focal pixels and peripheral pixels, and generate a Bitmask B to record their locations.

$$L_1[i][j] = \sum_{k=1}^{C} abs(P_{ij}^t[k] - P_{ij}^{t-1}[k]) \quad (1)$$

B. Pixel-adaptive Quantization

To accelerate diffusion models based on focal pixels, we propose the pixel-adaptive quantization method, which applies different precisions to different pixels based on the Bitmask from the focal pixel prediction operation. Specifically, we use high-precision quantization for focal pixels as they require detailed generation in current timestep, thus necessitating a large bitwidth to ensure accuracy. In contrast, peripheral pixels exhibit minimal change, allowing us to apply low-precision quantization. Fig. 3 illustrates the process of the pixel-adaptive quantization. We first extract focal and peripheral pixels from input feature map based on the Bitmask, followed by applying uniform high-precision ($int8$) quantization to focal pixels and outlier-aware mixed-precision ($int4 + int8$) quantization to peripheral pixels. Subsequently, we merge the quantized results back into a complete feature map. The input feature map becomes a mixed-precision feature map after quantization, which results in low hardware utilization. Therefore, we design a packing module in Section 5.2 to address this problem.

C. Prediction Finetuning

To identify focal pixels for each layer within a timestep in a memory-efficient manner, we propose the prediction finetuning operation that performs the same calculation process on Bitmasks as on each layer so that Bitmasks can match the focal

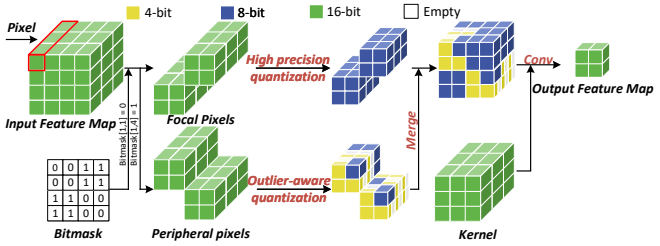


Fig. 3. The process of the pixel-adaptive quantization method.

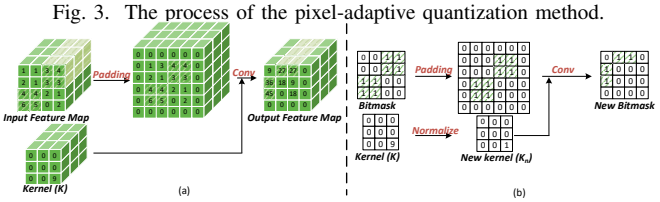


Fig. 4. An example of the prediction finetuning operation.

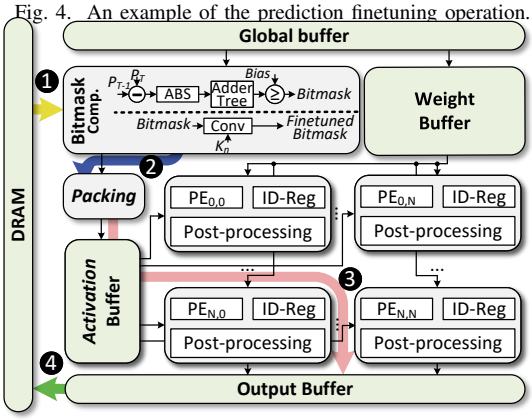


Fig. 5. Overview of the MHDiff architecture.

pixels of each layer. The necessity of the prediction finetuning operation is that the window-based sliding processing of each layer will cause a slight shift in the positions of focal pixels, which makes the Bitmask obtained in the first layer no longer valid in subsequent layers.

Specifically, as depicted in Fig. 4(a), we use a $3 \times 3 \times 3$ convolution kernel K and a $4 \times 4 \times 3$ input feature map as an example. For simplicity, we only show the calculation on one channel. We can find that since the weights are concentrated in the bottom right corner, when K is used to perform convolution on the input feature map, the focal pixels (the shaded area in Fig. 4(a)) will be shifted to the upper left corner. Therefore, current Bitmask is no longer valid for the next layer. To solve this problem, as shown in Fig. 4(b), we finetune this Bitmask according to the calculation process of convolution operation. We first normalize K to generate a new kernel K_n. Then we use K_n to perform convolution on the original Bitmask to obtain a new one, where the distribution of "1" perfectly matches the distribution of focal pixels in the output feature map. Through this method, we achieve finetuning Bitmasks, allowing Bitmasks to effectively guide the quantization for subsequent layers with minimal overhead.

V. MHDiff Architecture

A. Overview

Fig. 5 presents an overview of the MHDiff architecture, which consists of a Bitmask computation module, a packing

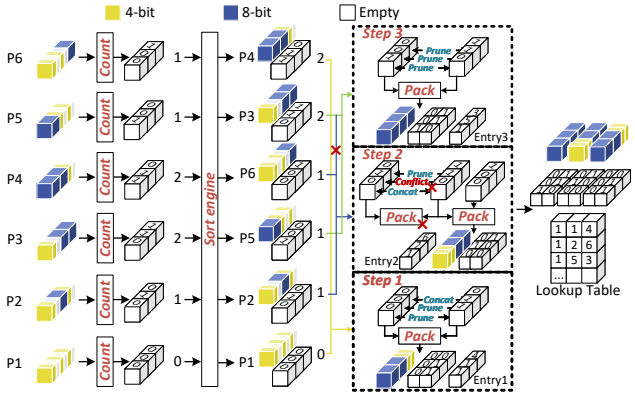


Fig. 6. The process of packing.

module, a PE array, and four buffers. The Bitmask computation module performs focal pixel prediction and prediction finetuning operations. The packing module packs mixed-precision feature maps into high-precision feature maps to facilitate efficient processing of the PE array. The PE array executes tensor multiplication operations. The on-chip buffers store activations, weights, outputs, and intermediate data generated by these modules, respectively.

The dataflow of the MHDiff architecture is also depicted in Fig. 5. In step 1, the Bitmask computation module first determines whether it is currently at the first layer of a timestep. If so, it reads the feature map of the previous timestep from DRAM and performs the focal pixel prediction operation to generate a Bitmask for the current timestep. Otherwise, it performs the prediction finetuning operation on the existing Bitmask to generate a new one for current layer. In step 2, the packing module extracts peripheral pixels from feature maps based on the Bitmask. Then they are packed into high-precision pixels. In step 3, the PE array reads the packed feature maps composed of focal pixels and packed peripheral pixels from activation buffer and the weight vectors from weight buffer for computation and writes the results to the output buffer. Finally, in step 4, the results in the output buffer is requantized and written back to DRAM.

B. Packing Module

To support mixed-precision ($int4 + int8$) computation, the PE units in the PE array need to be able to perform int4 or int8 calculations. However, PE units that perform $int4$ computation will complete faster and need to wait for that perform $int8$ computation, which results in low hardware utilization due to frequent synchronization. Therefore, we propose to pack two $int4$ values to form a $int8$ value, resulting in PE units running in the same speed. As a result, we are able to avoid synchronization, enabling higher hardware utilization.

Since only the peripheral pixels are mixed-precision, we need to pack these pixels to condense into full high-precision pixels. As the positions of $int8$ values in the quantized peripheral pixels are unpredictable, there are three possible cases during packing: (1) packing two $int4$ values, (2) packing a $int4$ value with a $int8$ value, and (3) packing two $int8$ values. For case 1, we can directly pack them into a $int8$ value that is called $int4+int4$ format. For case 2, considering

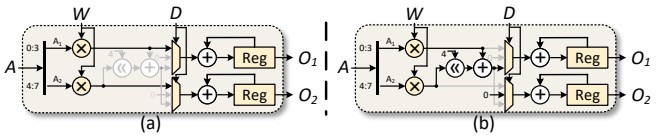


Fig. 7. The structure of PE unit.

that $int4$ values have little impact on model accuracy, we can directly discard the $int4$ value and only save the $int8$ value, which is called $int8+0$ format. For case 3, since $int8$ values are critical to model accuracy, we can't discard any of them. Therefore, we need to avoid this case. To this end, we design a packing module based on the greedy algorithm that attempts to avoid two $int8$ values conflicts.

The packing module consists of multiple count units, a sort engine, and a merge unit, which are responsible for recording the number of $int8$ values and their positions, sorting peripheral pixels, and merging them, respectively. The dataflow of the packing module is shown in Fig. 6. Upon receiving peripheral pixels, each count unit will count the number of $int8$ values and generate an identifier vector to record their positions. Then the sort engine will sort these pixels based on their number of $int8$ values. Afterwards, the merge unit will pack the pixel with the most $int8$ values and that with the least $int8$ values. In the example of Fig. 6, the packing module needs to pack 6 pixels ($P1-P6$). In step 1, it tries to pack $P4$ with $P1$. To avoid packing two $int8$ values, it first performs a logical AND operation on their identifier vectors and finds the result contains no "1", which means that these pixels don't have $int8$ values at the same position, so they can be packed together. It then packs these two pixels and their identifier vectors, and creates an entry to record the positions of these two pixels before packing. In step 2, the packing module attempts to pack $P3$ with $P2$, but finds they have two $int8$ values at position 2. Therefore, they cannot be packed together. It then attempts to pack $P2$ with $P6$. Finally, these two pixels are successfully packed together. In step 3, the packing module packs the remaining two pixels ($P5$ and $P3$), whose packing process is the same as step 1. In this way, we successfully pack the mixed-precision peripheral pixels into high-precision pixels and use the packed identifier vectors to record the type of each packed high-precision value. At the same time, we form a lookup table with the generated entries to facilitate the PE array to write results back to the correct buffer addresses.

C. PE Unit

In this section, we design a new PE unit to process the packed high-precision ($int8$) values. The specific structure of this PE unit is shown in Fig. 7. To support the $int4 + int4$ format, as shown in Fig. 7(a), the PE unit reads an activation ($int8$) and a weight ($int16$) from ports A and W, respectively. It then splits the high and low parts of the activation into two $int4$ values, A_1 and A_2. Then A_1 and A_2 are multiplied with the weight, and the results are accumulated into two registers. To support the $int8+0$ format, as shown in Fig. 7(b), after A_1 and A_2 are multiplied with the weight, the result of A_2 is left-shifted by 4 bits and then added to the result of A_1 to complete high-precision calculation. This result is then accumulated into

one of the registers. In this way, the PE unit efficiently supports the packed high-precision values.

Since this PE unit needs to support two different data formats, a key issue is how it can identify the format of the activation. To solve this problem, we add an additional port D, which reads a 2-bit identifier generated by the packing module. If the identifier is "00", it indicates the activation is formed by packing two $int4$ values. In this case, the PE unit will be configured as shown in Fig. 7(a). If the identifier is "10" or "01", it indicates the activation is originally a $int8$ value. In this case, the PE unit will be configured as shown in Fig. 7(b) and the identifier will control the multiplexer to write the result into the upper or lower register.

D. PE array

Building upon such PE units, we design a PE array to perform computations on the packed high-precision feature maps. Since the packing module disrupts the locations of pixels in the original feature map, it introduces two challenges. The first challenge is how to find the corresponding weights for each feature. The second challenge is how to write results back to the correct buffer addresses after computation.

For the first challenge, although the packing module disrupts the locations of pixels along the H (height) and W (width) dimensions, the locations along the C (input channel) dimension remain unaffected. Therefore, we propose to prioritize convolution calculations in the C dimension rather than in the H and W dimensions. Based on this, we design a PE array where PE units share a broadcast weight vector ($1 \times C$) but receive different activation vectors ($1 \times C$) for dot-product computations.

To address the second challenge, we equip each PE unit with a post-processing module that uses the lookup table generated by packing module to correctly write the results back to the output buffer. The lookup table we established is shown in Fig. 6. The i_{th} row of the table stores the information about the i_{th} pixel in the packed feature map, with three values indicating whether the vector is a packed vector and the positions of the two packed vectors in the original feature map. For example, in Fig. 6, the first row of the lookup table stores the information about the first pixel in the packed feature map. After a PE unit completes the calculation of this pixel, its post-processing module will read the first value of this row and find the value is "1", indicating that this pixel is a packed pixel. Then, according to the last two values, the post-processing module will read the two results from the O_1 and O_2 ports of the PE unit and write them to address "1" and "4" in the output buffer, respectively.

E. Bitmask Computation Module

The Bitmask computation module consists of multiple L_1 distance computation units and a convolution unit, which are responsible for performing the focal pixel prediction operation and the prediction finetuning operation. The L_1 distance computation unit reads a pixel at the current timestep and the corresponding pixel at the previous timestep. The two pixels undergo element-wise subtraction, followed by the absolute function, resulting in an absolute value vector. This vector is then passed into an adder tree for accumulation. Finally, the output from the adder tree is compared with a preset threshold to generate an element of the Bitmask. When the prediction finetuning operation is to be executed, the Bitmask computation module will perform convolution on the Bitmask, and generate the new Bitmask to guide the next layer.

TABLE I
AREA AND POWER BREAKDOWN OF MHDIFF.

Modules	Parameters	Area (mm^2)	Power(mW)
PE Array	32×32 PEs	0.474	164.5
Packing Module	128 comparators 128 adders	0.045	16.9
Bitmask Comp.	1×9 MACs 128 adders	0.031	10.4
On-chip Buffers	360KB Global buffer 132KB Activation buffer 48KB Weight buffer 132KB Output buffer	0.778	50.7
Total	UMC 28nm: Area=1.328 mm^2, Power=242.5 mW		

TABLE II
PRECISIONS BEFORE AND AFTER MHDIFF.

Model	FP16	Int8	MHDiff
Stable Diffusion 512	60%	59%	59%
Guided Diffusion 128	65%	64%	64%
Guided Diffusion 512	67%	66%	66%
Latent Diffusion-bed	67%	67%	66%
Latent Diffusion-church	63%	62%	62%

VI. EVALUATION

A. Evaluation Methodology

Software implementation To verify the effectiveness of the MHDiff algorithm, we select Stable Diffusion [1], Latent Diffusion [4], and Guided Diffusion [17] as benchmarks. Stable Diffusion is assessed on the Conceptual Captions dataset (at 512×512 resolution) [18]. Latent Diffusion is tested on LSUN datasets (bedroom and church-outdoor, both at 256x256 resolution) [19]. Guided Diffusion is tested on ImageNet datasets (at resolutions of 128x128 and 512x512) [20].

Hardware implementation We implement the proposed MHDiff architecture in Verilog and synthesize it by Synopsys Design Compiler to get the area and power consumption under 28nm technology, with a design frequency of 500MHz. For SRAM-based on-chip buffers, we use CACTI 7 [21] to model their area and power consumption. We then rescale the area and power data to 28nm technology using DeepScaleTool [22]. Table I provides the detailed design parameters, area, and power breakdown for the MHDiff.

Platforms for comparison We compare MHDiff with widely used hardware platforms, including a server CPU (Intel Xeon Gold 6226R) and a GPU (Nvidia A100). We also compare a state-of-the-art Diffusion accelerator Cambricon-D [13] on the same area budget.

B. Experiment Result

1) Precision: We measure the image quality of each workload using precision, which is a widely used evaluation metric [13], [17]. As illustrated in Table II, compared to the

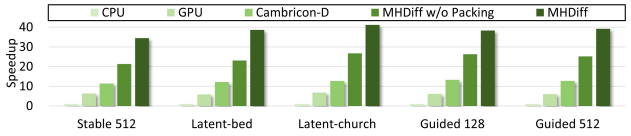


Fig. 8. Speedup of MHDiff.

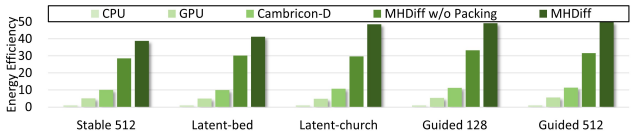


Fig. 9. Energy efficiency of MHDiff.

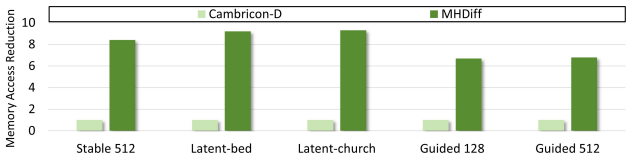


Fig. 10. Memory traffic of MHDiff compared with Cambricon-D.

baseline and $int8$ quantization, the precision of the MHDiff algorithm drops by less than 1% on average, which indicates that the MHDiff algorithm can maintain image quality well. Moreover, the MHDiff algorithm creates 83% $int4$ values and 17% $int8$ values on average, which greatly reduces computation compared to $int8$ quantization.

2) Speedup: Fig. 8 shows the speedup of the MHDiff over CPU, GPU, and Cambricon-D. MHDiff outperforms CPU, GPU, and Cambricon-D by 38.4×, 6.2×, and 3.1×, respectively. Compared to CPU and GPU, the speedup mainly comes from the proposed pixel-adaptive quantization algorithm that discards redundancy in diffusion models. Compared to Cambricon-D, the speedup originates from two aspects. First, Cambricon-D's high memory burden increases memory access latency, which drags down its acceleration gains. As we can see in Fig. 10, compared to Cambricon-D, we reduce memory traffic by 8.1x, which proves that MHDiff reduces the memory burden very well. Second, Cambricon-D has low hardware utilization. For example, as shown in Fig. 11, the PE array in Cambricon-D is idle nearly 50% of the time, while the idle time of the PE array in MHDiff is close to 0%. Therefore, MHDiff can make full use of its computing resources and achieve better acceleration.

In order to discuss the acceleration gained by the packing module, we implement a MHDiff accelerator without the packing module (labeled as MHDiff w/o Packing). As we can see in Fig. 8, MHDiff outperforms MHDiff w/o Packing by 1.6x. This is mainly because the packing module unifies precision, thus eliminating synchronization overhead and improving hardware utilization.

3) Energy Efficiency: As shown in Fig. 9, the energy efficiency of MHDiff is 45.2×, 8.3×, and 4.4× over CPU, GPU, and Cambricon-D. Compared to CPU and GPU, MHDiff uses fewer bits to represent data and perform calculations, which reduces the energy consumed by the computing units and memory. Compared to Cambricon-D, energy savings can be attributed to the following factors: 1) since MHDiff reduces memory accesses, the energy consumption caused by memory

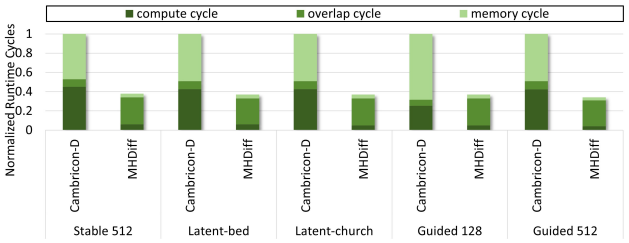


Fig. 11. Runtime breakdown of MHDiff compared with Cambricon-D.

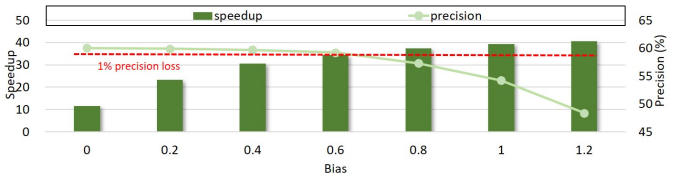


Fig. 12. Exploration on predefined threshold Bias.

is reduced; 2) the packing module improves hardware utilization, allowing MHDiff to save more computation energy.

4) Runtime Breakdown: Fig. 11 shows the runtime breakdown of MHDiff compared to the Cambricon-D, where the compute cycle, memory cycle, and overlap cycle refer to the computation time, memory access time, and the time they can run in parallel. We observe that MHDiff has less computation time than Cambricon-D because MHDiff has higher hardware utilization. Moreover, we find that the overlap cycle of MHDiff accounts for a large proportion, while that of Cambricon-D accounts for a small proportion. This is because Cambricon-D's heavy memory burden makes its PE array often idle due to lack of input data, resulting in serial execution of computation and memory access. In contrast, our memory traffic is relatively light, allowing computation and memory access to be performed in parallel. This is the main reason why the runtime of MHDiff can be significantly reduced.

5) Design Space Exploration: The goal of the MHDiff algorithm is to achieve a balance between optimizing efficiency and maintaining image quality by identifying focal pixels, in which the threshold $Bias$ plays an important role. Specifically, a larger $Bias$ means that fewer pixels are identified as focus pixels, which results in higher speedup but lower image quality. To explore the impact of $Bias$, we vary $Bias$ from 0 to 1.2 and see the precision and speedup of Stable Diffusion model on Conceptual Captions dataset. As illustrated in Fig. 12, when $Bias$ increases from 0 to 0.6, fewer pixels are identified as focus pixels, leading to higher speedup. But when we keep increasing $Bias$, the precision drops severely. As a result, we set $Bias$ as 0.6 to ensure both precision and performance.

VII. Conclusion

This paper proposes MHDiff, a framework for accelerating diffusion models. On the algorithm side, MHDiff identifies focal pixels for each timestep and applies different precision representations for different pixels, thereby achieving acceleration. On the architecture side, MHDiff transforms mixed-precision matrices into high-precision matrices and designs a specialized PE array through minor modifications to PE units. Experiments show that MHDiff can achieve satisfactory performance while maintaining image quality.

REFERENCES

[1] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis, march 2024," *URL http://arxiv.org/abs/2403.03206*.

[2] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, vol. 1, no. 2, p. 3, 2022.

[3] W. A., "Midjourney," https://www.midjourney.com/home, accessed:11.08.2024.

[4] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[5] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, "Cascaded diffusion models for high fidelity image generation," *Journal of Machine Learning Research*, vol. 23, no. 47, pp. 1–33, 2022.

[6] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," *arXiv preprint arXiv:2112.10741*, 2021.

[7] D. Epstein, A. Jabri, B. Poole, A. Efros, and A. Holynski, "Diffusion self-guidance for controllable image generation," *Advances in Neural Information Processing Systems*, vol. 36, pp. 16 222–16 239, 2023.

[8] Z. Yue, J. Wang, and C. C. Loy, "Resshift: Efficient diffusion model for image super-resolution by residual shifting," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] H. Li, Y. Yang, M. Chang, S. Chen, H. Feng, Z. Xu, Q. Li, and Y. Chen, "Srdiff: Single image super-resolution with diffusion probabilistic models," *Neurocomputing*, vol. 479, pp. 47–59, 2022.

[10] J. Wang, Z. Yue, S. Zhou, K. C. Chan, and C. C. Loy, "Exploiting diffusion prior for real-world image super-resolution," *International Journal of Computer Vision*, pp. 1–21, 2024.

[11] C. Corneanu, R. Gadde, and A. M. Martinez, "Latentpaint: Image inpainting in latent space with diffusion models," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 4334–4343.

[12] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 461–11 471.

[13] W. Kong, Y. Hao, Q. Guo, Y. Zhao, X. Song, X. Li, M. Zou, Z. Du, R. Zhang, C. Liu *et al.*, "Cambricon-d: Full-network differential acceleration for diffusion models," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 903–914.

[14] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei *et al.*, "20.2 a 28nm 74.34 tflops/w bf16 heterogenous cim-based accelerator exploiting denoising-similarity for diffusion models," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 362–364.

[15] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5775–5787, 2022.

[16] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang, and K. Keutzer, "Q-diffusion: Quantizing diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 535–17 545.

[17] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[18] P. Sharma, N. Ding, S. Goodman, and R. Soricut, "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 2556–2565.

[19] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[21] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[22] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

A Data-Centric Hardware Accelerator for Efficient Adaptive Radix Tree

Jin Zhao[†], Yu Zhang[†], Jun Huang[†], Weihang Yin[†], Hui Yu[†], Hao Qi[†], Zixiao Wang[†], Longlong Lin[§], Xiaofei Liao[†], Hai Jin[†]

[†]*National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, China*

[§]*College of Computer and Information Science, Southwest University, Chongqing, China*

{zjin, zhyu, jun_huang, hannyin, huiy, theqihao, zwang62}@hust.edu.cn longlonglin@swu.edu.cn {hjin, xfliao}@hust.edu.cn

***Abstract*—*Adaptive Radix Tree* (ART) is a widely used tree index structure prevalent in various domains such as databases and key-value stores. Despite many solutions have been proposed to improve the performance of ART, they still suffer from significant redundant tree traversals and serious synchronization cost when concurrently performing the operations (e.g., read/write) over ART. In this work, we observe that most operations of real-world workloads tend to target a small subset of ART nodes frequently, exhibiting strong temporal and spatial similarities among the operations. Based on this observation, we propose a data-centric hardware accelerator, called *DCART*, to efficiently support the operations over ART. Specifically, DCART proposes a novel data-centric processing model into the accelerator design to coalesce the operations associated with the same ART nodes and adaptively cache the frequently traversed ART nodes and their search results, thereby fully exploiting the similarities among the operations for lower tree traversal and synchronization overhead. We implemented DCART on the Xilinx Alveo U280 FPGA card and compared it with the cutting-edge solutions, DCART achieves 21.1×-44.2× speedups and 71.1×-148.9× energy savings.**

I. Introduction

Adaptive Radix Tree (ART) [6], [8], [9], [17], [19], [27] is an efficient tree index structure, which plays a crucial role in numerous applications like large-scale database systems [7], [11] and key-value stores [12], [31]. With the wide application of ART, many optimizations (such as DART [28], Heart [17], and SMART [11]) are designed to improve the performance of the operation (e.g., read or write a key-value item) over ART. However, when concurrently performing the operations over ART, these solutions still suffer from significant redundant tree traversals and serious synchronization cost on general-purpose processors due to the following two problems.

First, different operations usually traverse the same nodes of ART individually, which incurs significant *redundant tree traversal overhead*. Moreover, the irregular and unpredictable tree traversal cause massive random data accesses, resulting in poor data locality. It eventually causes most proportion (up to 90.1%) of the fetched nodes in the on-chip memory to be redundant, underutilizing the memory bandwidth significantly. Second, the same node of ART may be handled by different operations concurrently, which brings serious *synchronization cost*. This is because existing ART design applies a lock-based algorithm [8], [9] for concurrency control and the real-world workloads typically perform the operations frequently over a few common nodes of ART. As a result, the synchronization cost wastes much execution time (up to 53.6%).

Through analyzing the operation distribution of the real-world workloads, we have two findings. First, the same nodes of ART may be frequently handled by different operations within a short time interval, which shows temporal similarity. This motivates us to coalesce the processing of the operations targeting the same nodes to achieve lower synchronization and tree traversal cost. Second, many operations get involved with only a small subset of ART nodes, which exhibits spatial similarity. This suggests that these frequently traversed nodes and their search results can be cached in the on-chip memory to further reduce off-chip communications.

Based on the observations, this paper proposes an efficient **data-centric** hardware accelerator *DCART*, which leverages a novel data-centric *Combine-Traverse-Trigger* processing model to effectively handle the operations over ART by fully exploiting the similarities among them. Specifically, DCART features specialized hardware pipelines to dynamically *combine* the operations targeting the same nodes of ART and then *traverse* the nodes of ART associated with these combined operations so as to *trigger* their processing together. Moreover, DCART dynamically maintains the search results (i.e., which are essentially the shortcuts) for the frequently traversed nodes of ART, enabling multiple operations to reuse these shortcuts to quickly search their target nodes. In this way, the redundant tree traversals and synchronization overhead can be effectively alleviated, ensuring higher utilization of the processing unit and fewer off-chip communications. To achieve better data locality, DCART further specializes the memory subsystem to preferentially resident the frequently traversed nodes of ART in the on-chip memory for multiple operations.

We have implemented DCART on a Xilinx Alveo U280 FPGA card. Experimental results show DCART outperforms the cutting-edge ART implementations (i.e., SMART [11] and

This paper is supported by National Key Research and Development Program of China (No. 2023YFB4503400), NSFC (No. 62402457), and Key Research and Development Program of Hubei Province (No. 2023BAB078). This work is also supported by Ant Group through CCF-Ant Research Fund (No. CCF-AFSG RF20240204). Yu Zhang (zhyu@hust.edu.cn) is the corresponding author of this paper.

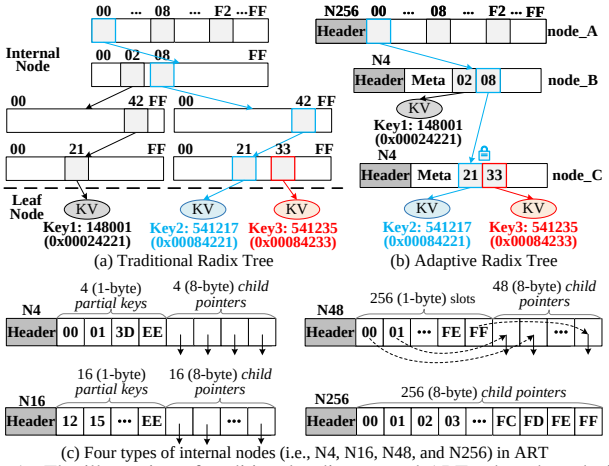


Fig. 1. The illustration of traditional radix tree and ART, where hexadecimal partial keys are shown to enhance clarity

CuART [6]) running on Intel Xeon CPU and NVIDIA A100 GPU by 35.9×-44.2× and 21.1×-31.2× with 92.7×-148.9× and 71.1×-126.2× energy savings, respectively.

II. Background and Motivation

A. Adaptive Radix Tree

Adaptive Radix Tree (ART) [7]–[9], [11], [17], [19], [27], [28] standing out as a widely used tree index structure crafted to enhance the memory efficiency of traditional radix tree. As depicted in Figure 1(a) and (b), both the traditional radix tree and ART organize the segmented keys within the top-down search path of the tree. Specifically, in traditional radix tree, each internal node contains an array of child pointers, where each pointer corresponds to a *partial key* (i.e., a segment of bits of the whole key). Note that an internal node in the traditional radix tree typically reserves space for all 256 potential partial keys. However, due to the sparse distribution of actual keys [8], [11], [17], many of these pointers remain empty, resulting in inefficient memory usage. To resolve this issue, as shown in Figure 1(b), ART [8], [9] optimizes the traditional radix tree by reducing tree height with path compression and introducing four well-designed internal node structures (i.e., `N4`, `N16`, `N48`, and `N256` as shown in Figure 1(c)), where each accommodating a distinct number of pointers (4, 16, 48, and 256, respectively). For better memory efficiency, ART dynamically selects the most suitable internal node structure according to the actual data distribution.

For each operation (e.g., read or write a key-value item) over an ART, it needs to traverse the internal nodes of this ART by performing top-down partial key matching so as to locate the target node of this operation. When concurrently performing the operations over ART, a lock-based algorithm (i.e., the *Read-Optimized Write EXclusionn* (ROWEX) protocol [9]) is used to achieve concurrency control of ART. In detail, it adopts the node-level write locks (e.g., `node_C` must be locked when an operation related to `Key2` needs to modify it), and any read or write is executed atomically. Note that if the operation causes a change in the type of node (e.g., `N4` split into `N16` when `N4` is full), its parent node also needs to be locked.

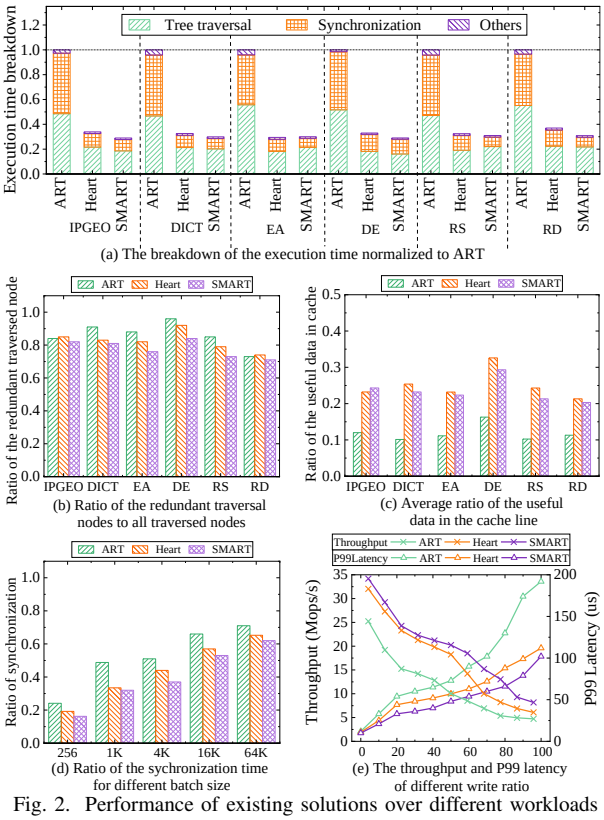


Fig. 2. Performance of existing solutions over different workloads

B. Challenges of Existing Solutions

Although many solutions [6]–[9], [11], [17], [19], [27], [28] have been developed to enhance the performance of the operations (e.g., read or write a key-value item) over ART, they still suffer from significant redundant tree traversal and serious synchronization overhead when concurrently processing the operations over ART. These unique challenges lead to inefficient utilization of the cores and memory bandwidth of the general-purpose processors. To demonstrate these problems, we evaluated three state-of-the-art ART implementations (i.e., ART [9], Heart [17], and SMART [11]) running on Intel Xeon CPU. Note that the details of the platform and benchmarks used in this evaluation are described in Section IV-A. Figure 2(a) shows that, although SMART outperforms other under all circumstances, the majority of execution time for the processing of operations on SMART (more than 95.82%) is still consumed by tree traversal and synchronization. We illustrate the reasons for the above inefficiency as follows.

Challenge 1: Massive redundant tree traversal cost incurs the underutilization of memory bandwidth. Different operations may need to individually traverse the same nodes of ART, incurring significant redundant tree traversal cost. As depicted in Figure 1(b), when processing two operations associated with `Key2` and `Key3`, respectively, each operation needs to individually conduct partial key matching along the nodes `node_A`→`node_B`→`node_C`. Such duplicate tree traversal incurs many redundant off-chip communications (e.g., the accesses to `node_A`, `node_B`, and `node_C`). As shown in Figure 2(b), more than 77.8% of traversed nodes are redundant when handling the operations over SMART, while this ratio

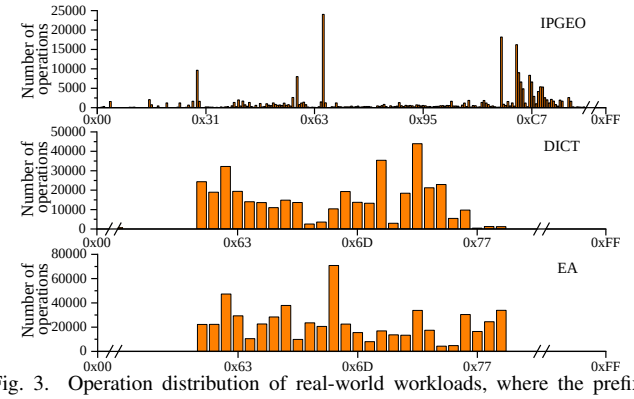


Fig. 3. Operation distribution of real-world workloads, where the prefixes (ranging from 0x00 to 0xFF) of the keys are listed along the horizontal axis and the number of operations led by different prefixes are shown vertically

can be even as high as 86.1% and 82.5% for ART and Heart, respectively. Besides, massive irregular and unpredictable data accesses during the tree traversal cause significant random data accesses, incurring poor data locality. Note that the partial key and child pointer are typically sized of 1 and 8 bytes, respectively, significantly smaller than the 64-byte cachelines of general-purpose processors. Thus, only a small portion of the data loaded into the cache line (20.2% on average in Figure 2(c)) is useful. This eventually results in many superfluous fragmented off-chip communications, underutilizing the on-chip memory and memory bandwidth significantly.

Challenge 2: The concurrency control of ART suffers from serious synchronization overhead. Existing ART implementations typically use lock-based algorithms [6]–[9], [19], [27], [28] to achieve concurrency control, which incurs serious synchronization overhead among different operations over ART. As depicted in Figure 1(b), when the operation associated with Key2 has acquired a lock for the node (e.g., node_C) of ART, the operation associated with Key3 can be conducted only when the operation associated with Key2 has been completed and released the lock. Although several works (e.g., Heart [17] and SMART [11]) endeavor to alleviate the node-level lock contention using the atomic primitive *Compare-And-Switch* (CAS), atomic operations are still very inefficient. This is because the operations over ART suffer from poor data locality, which leads to frequent cache misses, whereas a CAS operation on an Intel processor experiences a slowdown of more than 15 times when data resides in RAM compared to when it resides in L1 cache [21]. As shown in Figure 2(d), when the number of operations increases in the real-world workload *IPGEO*, the ratio of the execution time associated with concurrency control to the total execution time increases from 16.2% to 62.1% over Heart and SMART, while this ratio increases from 24.1% to 71.3% over ART. In addition, from Figure 2(e), we can observe that the performance deteriorates rapidly when the ratio of write operations increases in *IPGEO* (i.e., more lock acquirement and release are required).

C. Similarities among the Operations over ART

Figure 3 describes the statistical studies on the operation distribution of three real-world workloads. We observe that there are strong spatial and temporal similarities among the operations over ART. It provides opportunities to address the limitations of existing architectures and enhance the performance of the operations over ART.

Observation 1: The same nodes of ART may be frequently handed by different operations within a short period of time, which exhibits the temporal similarity. Figure 3 shows that the nodes corresponding to the keys with the prefix 0x67 are traversed and handled by over 24,000 operations for *IPGEO*. This motivates us to propose a novel ***data-centric*** *Combine-Traverse-Trigger* (CTT) processing model to efficiently coalesce the operations over ART by fully exploiting the temporal similarity. This model first *combines* the operations targeting the same nodes of ART according to the prefixes of their keys. It then *traverses* the nodes of ART to search the target nodes of these combined operations and finally *triggers* the processing of these operations together. Thus, each node of ART needs to be traversed only once to drive the processing of multiple operations, and the concurrent operations associated with this node can be serialized to naturally alleviate synchronization overhead. This is fundamentally different from the ***operation-centric*** processing model used in existing solutions [6]–[9], [11], [17], [19], [27], [28] that launch each operation to individually perform the tree traversal over ART to retrieve and atomically handle its target nodes. Take Figure 1(b) as an example, our proposed CTT processing model first combines the operations associated with Key2 and Key3 because their keys have same prefix, and then traverses the nodes of ART through conducting the partial key matching along node_A→node_B→ node_C. After that, the combined operations will be triggered to perform together without the synchronization of node_C when these operations need to modify node_C. Besides, the duplicate partial key matching from node_A to node_C can be eliminated by coalescing the same tree traversals, achieving lower tree traversal overhead.

Observation 2: Many operations involve only a small fraction of nodes within ART, which exhibits the spatial similarity. Figure 3 shows that more than 96.65% of tree traversals is responsible for accessing only 5% of the nodes in ART. The results depict that the node traversal distribution is usually skewed. This prompts us to enhance the performance of our proposed data-centric CTT processing model via the following aspects. First, the frequently traversed nodes (denoted as the high-value nodes) can be preferentially resident in the on-chip memory, efficiently exploiting the spatial similarity among the operations for fewer off-chip communications. Second, the partial key matching results associated with these nodes (e.g., the prefix of the key and the address of the corresponding node such as <0x000842, address of node_C> in Figure 1(b), which is essentially a *shortcut*) can be cached to be directly reused by other operations that need to handle them (i.e., the operations that need to handle node_C), further reducing the tree traversal overhead.

Challenges: Although our proposed data-centric CTT processing model ensures much lower synchronization cost and fewer tree traversals, its software-only implementation suffers from high runtime overhead (see Section IV-B) due to the

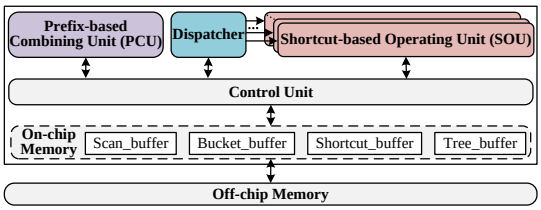


Fig. 4. Architecture of DCART

following reasons. First, it requires expensive runtime cost to dynamically coalesce the operations and maintain the shortcuts on the fly, which may impede the performance improvement brought by exploiting the similarities among the operations over ART. Second, the tree traversal involves data-dependent branches, which induces low instruction-level parallelism. To overcome these challenges, we present a data-centric hardware accelerator *DCART*, which customizes specialized hardware designs to efficiently handle the operations over ART.

III. DCART Architecture

A. DCART Overview

Figure 4 depicts the architecture of DCART, which contains three key hardware units (i..e, *Prefix-based Combining Unit* (PCU), *Dispatcher*, and *Shortcut-based Operating Unit* (SOU)) and on-chip buffers. Specifically, the PCU is employed to combine the operations targeting the same ART nodes by assigning the operations into disjoint buckets based on the prefixes of their keys. The *Dispatcher* is used to dispatch these buckets to different SOUs for parallel processing, in which the operations targeting the same nodes are only handled by a single SOU, alleviating synchronization overhead. Each SOU is employed to efficiently handle the combined operations in each bucket and also maintain the shortcuts to eliminate the duplicate tree traversal cost. On-chip memory is composed of several buffers to cache different types of data and also isolate the accesses to these different data (e.g., tree structure and operations), efficiently reducing off-chip communications. To fully exploit the spatial similarity of the operations over the ART, it also adaptively caches the frequently handled nodes for multiple operations, avoiding the data thrashing of them.

B. Prefix-based Operation Combining

To alleviate the lock contention among the concurrent operations over ART, the PCU sequentially scans the keys of the operations and dynamically assigns them into different disjoint buckets according to the prefixes of these keys. As shown in Figure 5, the PCU performs three stages as a pipeline to efficiently achieve the operation combining. Note that sixteen tables are created in the off-chip memory to store the combined operations for different buckets, where each table (e.g., *Bucket_Table*$_i$) is specified to a particular bucket (i.e., the i^{th} bucket) using a label (defined by the prefix). When the concurrent operations are arrived, at the *Scan_Operation* stage, it sequentially obtains the key of each operation. Then, the *Get_Prefix* stage calculates the specified prefix (e.g., *Prefix*$_i$) of this key, where the first 8 bits of the key are used as the specified prefix by default. After that, at the *Combine_Operation* stage, it assigns this operation into the corresponding *Bucket_Table* by matching *Prefix*$_i$ with

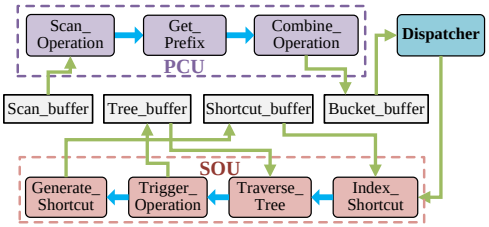


Fig. 5. Microarchitecture of DCART, where blue solid lines and green solid lines denote data flow and on-chip memory transfers, respectively

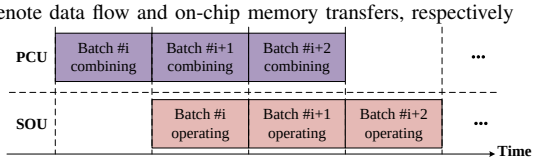


Fig. 6. The overlap between PCU and SOU

the labels of *Bucket_Tables* and then sets this operation as combined. In this way, the operations that target the same nodes are combined into the same bucket. It enables the *Dispatcher* to dispatch the operations that target the same node to be handled by only a single SOU, avoiding the acquirement of locks to perform synchronization for concurrent control.

C. Shortcut-based Operating

When a SOU is dispatched with a bucket of operations, this SOU sequentially retrieves an operation from this bucket and searches the target node of this operation by traversing the tree or using the shortcuts (to be introduced later), efficiently conducting the operations that target the same node together with lower tree search overhead. Note that a hash table, i.e., *Shortcut_Table*, is created in the off-chip memory to store the partial key matching results to be reused by multiple operations, avoiding significant redundant tree traversal overhead. Each entry of *Shortcut_Table* is the form of <`Key_ID`, `Address_Target_Node`, `Address_Parent_Node`>.

Specifically, as shown in Figure 5, each SOU contains four stages, which are conducted as a pipeline. At the *Index_Shortcut* stage, the SOU pops an operation from the assigned bucket and obtains the key (e.g., `Key`$_x$) of this operation. Then, it uses the `Key`$_x$ to obtain the addresses of the target node (e.g., `Node_X`) and its parent node (e.g., `Node_Y`) according to the *Shortcut_Table*. If the corresponding addresses can be obtained from the *Shortcut_Table* (e.g., `Node_X` has been handled by another operation previously), the *Traverse_Tree* stage will directly use these addresses to fetch `Node_X` and `Node_Y` from the tree structure (i.e., the ART), avoiding the tree traversal. Otherwise, the *Traverse_Tree* stage will use `Key`$_x$ to obtain `Node_X` and `Node_Y` through performing top-down partial key matching over the tree. At the *Trigger_Operation* stage, the SOU triggers all retrieved operation (e.g., read or write) on the target node to be performed together. If `Node_X` and `Node_Y` are obtained by traversing the tree, the *Generate_Shortcut* stage will generate a shortcut associated with `Key`$_x$, i.e., creating an entry (i.e., <`Key`$_x$, `Address_Node_X`, `Address_Node_Y`>) in *Shortcut_Table*. Note that the corresponding entry in *Shortcut_Table* needs to be updated when this operation causes a change in the type of `Node_X`. As the operations that target the same node being coalesced, the SOU typically obtains the

target nodes of the operations directly based on the maintained shortcuts and also achieves better data locality.

D. Overlap of Combining and Operating

The combining of the operations needs to assign them into different disjoint buckets on the fly, which introduces extra runtime cost. Fortunately, the interleaving between combining and operating for each operation batch provides the potential opportunity to hide the operation combining overhead. Thus, we can sequentially divide the arrived operations into a series of batches. As shown in Figure 6, when the i^{th} batch of operations has been handled by PCU, the Dispatcher will assign the combined operations associated with the i^{th} batch to be handled by SOUs. In the meantime, the PCU starts combining the operations of the $(i+1)^{th}$ batch. By such means, the overheads of the operation combining can be usually hidden for better performance.

E. Value-aware Memory Hierarchy

To efficiently access the ART nodes, arrived operations, *Bucket_Tables*, and *Shortcut_Table*, as shown in Figure 4, four on-chip buffers, i.e., *Tree_buffer*, *Scan_buffer*, *Bucket_buffer*, and *Shortcut_buffer*, are employed to cache these data, respectively. However, the high-value nodes (introduced in Section II-C) may be evicted from the *Shortcut_buffer* due to the irregular tree traversal over ART. The access to the tree structure typically accounts for the most portion of all data accesses. Therefore, we design a value-aware data management strategy to efficiently manage the *Tree_buffer*, while DCART uses LRU [4] to manage the other buffers by default. In detail, for the management of the *Tree_buffer*, DCART uses the values of the nodes to determine the victims. Note that a node (e.g., `Node_X`) will be handed more times when the corresponding bucket owns more operations, thus we use the number of the operations in the corresponding bucket to approximate the value of this node (i.e., $Value_x$), which can be obtained after the operation coalescing. When a node `Node_X` is requested, if the *Tree_buffer* is not full, `Node_X` will be directly cached in the *Tree_buffer*. Otherwise, DCART will evict the node with the lowest value (i.e., $Value_{low}$) from the *Tree_buffer* and cache `Node_X` in the *Tree_buffer* when $Value_x$ is greater than $Value_{low}$. In this way, DCART can effectively prevent cache thrashing for high-value nodes (i.e., frequently traversed nodes), ensuring better data locality.

IV. Evaluation

A. Experiment Setup

DCART Settings. We implement DCART on the Xilinx Alveo U280 FPGA card, which is equipped with the XCU280 FPGA chip. The FPGA provides 1.3 M LUTs, 2.6 M Registers, 9 M BRAM resources, and 8 GB HBM. DCART has 16 SOUs, and we implement the on-chip memory of DCART using the BRAM resources. More parameter details of DCART are depicted in Table I. We use Xilinx Vivado 2019.1 to obtain the clock rate for DCART and conservatively use 230MHz in our experiments. Besides, we also implement our data-centric CTT processing model on a machine equipped with two 48-core Intel Xeon Platinum 8468 processors and 1024 GB DRAM.

TABLE I
Parameter details of DCART

Compute	1×PCU, 1×Dispatcher, 16×SOUs
On-chip memory	*Scan_buffer* (512 KB), *Bucket_buffer* (2 MB), *Shortcut_buffer* (128 KB), *Tree_buffer* (4 MB)

The CPU version (i.e., the software-only implementation of our processing model) is called *DCART-C*.

Workloads. We use three real-world workloads (i.e., *IP address record by GeoLite2-Country* (IPGEO)[1], DICT[2], and *E-mail Addresses* (EA)[3]) and three synthesized workloads [9] with 50M dense 8-byte integer keys (DE), 50M random sparse 8-byte integers keys (RS), and 50M random dense 8-byte integers keys (RD), respectively. For each workload, the operations consist of 50% read and 50% write by default.

Baselines. We compare DCART with two cutting-edge CPU-based tree indexes (i.e., ART [9] and SMART [11]) and a state-of-the-art GPU-based tree index (i.e., CuART [6]), where ART and SMART are used as the CPU baselines and CuART is used as the GPU baseline. Note that SMART is designed for disaggregated memory, thus we port them to shared-memory by re-implementing them from scratch. All experiments run on a machine configured with two 48-core Intel Xeon Platinum 8468 processors, 1024 GB DRAM, an NVIDIA A100 GPU, and a Xilinx Alveo U280 FPGA. Like existing solutions [1], [3], [22], [26], we employ CPU Energy Meter [15], nvidia-smi [18], and xbutil [25] to measure the energy consumption of the CPU-based solutions (i.e., ART [9], SMART [11], and DCART-C), the GPU-based solution (i.e., CuART [6]), and the FPGA-based solution (i.e., DCART), respectively.

B. Overall Performance

Lock Contentions. Figure 7 evaluates the number of lock contentions induced by different solutions. It shows that the lock contentions induced by DCART-C and DCART are only 3.2%-19.7% of that by other solutions. This is because our data-centric CTT processing model only needs to acquire a single lock for multiple operations that need to handle the same node by efficiently coalescing these operations, alleviating massive lock contentions for lower synchronization overhead.

Partial Key Matches. Figure 8 shows the number of partial key matches performed by different solutions. It depicts that the number of partial key matches of DCART-C and DCART are only 3.2%-5.7%, 6.5%-14.3%, and 8.8%-15.9%, of ART, SMART, and CuART, respectively. The reasons are twofold. First, different operations can share the common tree traversals through combining these operations according to their prefixes. Second, the shortcuts of the frequently traversed nodes can be reused by multiple operations to directly search their target nodes, further reducing the tree traversal overhead.

Performance. Figure 9 evaluates the execution time of different solutions. We can find that DCART-C only slightly outperforms ART, SMART, and CuART, although our processing model can efficiently reduce redundant tree traversals

[1]https://github.com/analogic/ipgeo#dbsql
[2]https://github.com/dwyl/english-words
[3]https://archive.org/details/300MillionEmailDatabase

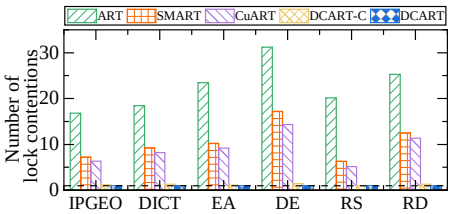


Fig. 7. Number of lock contentions

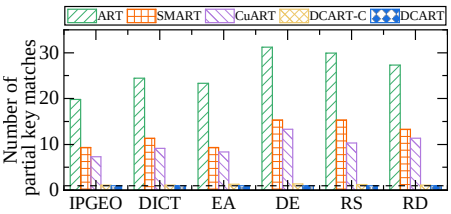


Fig. 8. Number of partial key matches

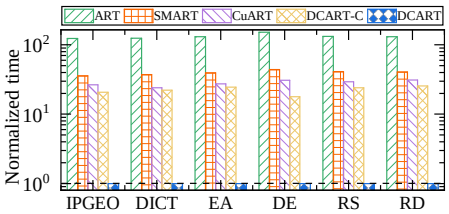


Fig. 9. Normalized execution time

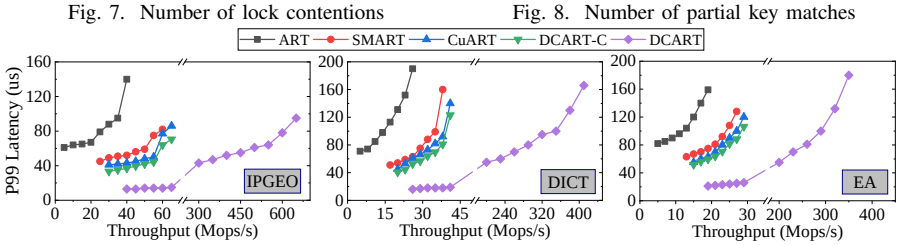


Fig. 10. The throughput-latency curves under different real-world workloads

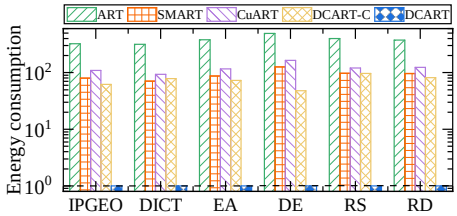


Fig. 11. Normalized energy consumption

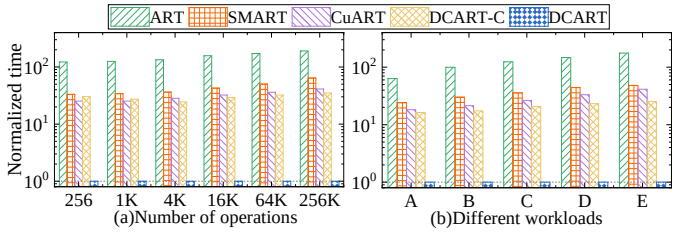


Fig. 12. Sensitivity studies

and synchronization overhead when handling the operations over ART. This is because DCART-C requires high runtime overhead to dynamically combine the operations, maintain the shortcuts, and irregularly traverse the tree structure on the fly. This impedes the overall efficiency of our processing model. Different from DCART-C, DCART can achieve 123.8×-151.7×, 35.9×-44.2×, and 21.1×-31.2× performance improvements compared with ART, SMART, and CuART, respectively. The reasons are manifold. First, DCART can reduce the runtime cost arising from our processing model because of its sophisticated hardware designs. Second, DCART can preferentially resident the frequently traversed ART nodes on the on-chip memory, achieving better data locality. Figure 10 evaluates the throughput-latency curves of different solutions with three real-world workloads, using various numbers of the operations. It shows that DCART achieves lower P99 latency and higher throughput in comparison with other solutions.

Energy Savings. Figure 11 shows that DCART reduces the energy consumption of ART, SMART, CuART, and DCART-C by 315.1×-493.5×, 92.7×-148.9×, 71.1×-126.2×, and 48.1×-97.6×, respectively.

C. Sensitivity Studies

Figure 12(a) evaluates the performance of different solutions when handling *IPGEO* with different number of concurrent operations. It shows that DCART achieves better performance as the number of operations increases. This is because DCART can efficiently coalesce the operations to alleviate more lock contention and tree traversal cost. Figure 12(b) evaluates the performance of different solutions with various workloads of *IPGEO*, where the workloads contain A (100% read), B (75% read, 25% write), C (50% read, 50% write), D (25% read, 75% write), and E (100% write). The results show that better performance improvement can be obtained as the ratio of write increases (which means more lock contention).

V. Related Work

Two categories of index structures (i.e., hash indexes and tree-based indexes) are most widely used today for databases.

Hash Indexes. Hash indexes [2], [14], [16], [32] are flat data structures that are able to support fast point access within constant lookup time complexity, i.e., O(1). However, because hash tables scatter the keys randomly, they are unable to support range queries efficiently.

Tree-based Indexes. Tree-based indexes [10], [20], [29], [30] are critical for many applications requiring range queries, and most previous databases typically apply the variants of B+tree [5], [13], [23], [24] to build range indexes. However, B+tree suffers from write amplification. In comparison with it, ART [8], [9] has smaller write amplification because it does not hold the entire keys in its internal nodes. Recently, many optimizations [7], [12], [19], [27], [31] have been proposed for ART. DART [28] introduces a triplet-based search mechanism to guarantee its scalability. Heart [17] proposes a PM-friendly node structure and a CAS-based concurrency control method to reduce the high PM overhead and improve scalability. SMART [11] is proposed to achieve efficient ART for disaggregated memory. To further improve the performance of ART, CuART [6] offloads the read and write operations onto GPUs. However, these solutions still suffer from serious redundant tree traversals and synchronization cost when concurrently performing the operations over ART. In contrast, DCART can effectively overcome these challenges by fully exploiting the temporal and spatial similarities among these operations.

VI. Conclusion

This paper proposes a novel data-centric hardware accelerator DCART to enhance the efficiency of operations over ART. By coalescing the operations targeting the same ART nodes and adaptively caching the frequently traversed ART nodes and their search results, DCART significantly reduces the synchronization cost and off-chip communications, achieving high energy efficiency. Compared to the state-of-the-art solutions, i.e., SMART [11] and CuART [6], DCART gains 21.1×-44.2× speedups with 71.1×-148.9× energy savings.

REFERENCES

[1] X. Chen, Y. Chen, F. Cheng, H. Tan, B. He, and W. Wong, "Regraph: Scaling graph processing on hbm-enabled fpgas with heterogeneous pipelines," in *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture*, 2022, pp. 1342–1358.

[2] D. Hu, Z. Chen, J. Wu, J. Sun, and H. Chen, "Persistent memory hash indexes: an experimental evaluation," *Proceedings of the VLDB Endowment*, vol. 14, no. 5, 2021.

[3] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: focus on CPU and GPU," in *Proceedings of the 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, 2023, pp. 106–118.

[4] D. A. Jiménez, "Insertion and promotion for tree-based pseudolru last-level caches," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 284–296.

[5] W.-H. Kim, M. K. Ramanathan, X. Fu, S. Kashyap, and C. Min, "PACTree: a high performance persistent range index using pac guidelines," in *Proceedings of the 28th ACM SIGOPS Symposium on Operating Systems Principles*, 2021, pp. 424–439.

[6] M. Koppehel, T. Groth, S. Groppe, and T. Pionteck, "Cuart - a cuda-based, scalable radix-tree lookup and update engine," in *Proceedings of the 50th International Conference on Parallel Processing*, 2021, pp. 1–10.

[7] S. K. Lee, K. H. Lim, H. Song, B. Nam, and S. H. Noh, "WORT: write optimal radix tree for persistent memory storage systems," in *Proceedings of the 15th USENIX Conference on File and Storage Technologies*, 2017, pp. 257–270.

[8] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: Artful indexing for main-memory databases," in *Proceedings of the 29th IEEE International Conference on Data Engineering*, 2013, pp. 38–49.

[9] V. Leis, F. Scheibner, A. Kemper, and T. Neumann, "The ART of practical synchronization," in *Proceedings of the 12th International Workshop on Data Management on New Hardware*, 2016, pp. 3:1–3:8.

[10] Y. Li, B. He, J. Yang, Q. Luo, and K. Yi, "Tree indexing on solid state drives," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 1195–1206, 2010.

[11] X. Luo, P. Zuo, J. Shen, J. Gu, X. Wang, M. R. Lyu, and Y. Zhou, "SMART: a high-performance adaptive radix tree for disaggregated memory," in *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation*, 2023, pp. 553–571.

[12] S. Ma, K. Chen, S. Chen, M. Liu, J. Zhu, H. Kang, and Y. Wu, "Roart: Range-query optimized persistent art," in *Proceedings of the 19th USENIX Conference on File and Storage Technologies*, 2021, pp. 1–16.

[13] Y. Mao, E. Kohler, and R. T. Morris, "Cache craftiness for fast multicore key-value storage," in *Proceedings of the 2012 European Conference on Computer Systems*, 2012, pp. 183–196.

[14] A. Mathew and C. Min, "Hydralist: A scalable in-memory index using asynchronous updates and partial replication," in *Proceedings of the VLDB Endowment*, 2020, pp. 1332–1345.

[15] L. Munich, "CPU energy meter," 2021. [Online]. Available: https://github.com/sosy-lab/cpu-energy-meter

[16] M. Nam, H. Cha, Y. ri Choi, S. H. Noh, and B. Nam, "Write-optimized dynamic hashing for persistent memory," in *Proceedings of the 17th USENIX Conference on File and Storage Technologies*, 2019, pp. 31–44.

[17] L. Nie, S. Zheng, B. Zhang, J. Xu, and L. Huang, "Heart: a scalable, high-performance art for persistent memory," in *Proceedings of the 41st IEEE International Conference on Computer Design*, 2023, pp. 487–490.

[18] NVIDIA, "Nvidia system management interface," 2023. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[19] W. Pan, T. Xie, and X. Song, "HART: A concurrent hash-assisted radix tree for DRAM-PM hybrid memory systems," in *Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium*, 2019, pp. 921–931.

[20] K. Ren, Y. Guo, J. Li, X. Jia, C. Wang, Y. Zhou, S. Wang, N. Cao, and F. Li, "Hybridx: New hybrid index for volume-hiding range queries in data outsourcing services," in *Proceedings of the 40th IEEE International Conference on Distributed Computing Systems*, 2020, pp. 23–33.

[21] H. Schweizer, M. Besta, and T. Hoefler, "Evaluating the cost of atomic operations on modern architectures," in *Proceedings of the 2015 International Conference on Parallel Architectures and Compilation*, 2015, pp. 445–456.

[22] L. Song, Y. Chi, L. Guo, and J. Cong, "Serpens: a high bandwidth memory based accelerator for general-purpose sparse matrix-vector multiplication," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 211–216.

[23] Q. Wang, Y. Lu, and J. Shu, "Sherman: a write-optimized distributed b+tree index on disaggregated memory," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 1033–1048.

[24] Z. Wang, A. Pavlo, H. Lim, V. Leis, H. Zhang, M. Kaminsky, and D. G. Andersen, "Building a bw-tree takes more than just buzz words," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 473–488.

[25] Xilinx, "Vitis unified software development platform 2020.2 documentation," 2020. [Online]. Available: https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/index.html

[26] Z. Yang, K. Adamek, and W. Armour, "Accurate and convenient energy measurements for gpus: A detailed study of nvidia gpu's built-in power sensor," in *Proceedings of the 2024 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 307–323.

[27] J. Zhang, Y. Luo, P. Jin, and S. Wan, "Optimizing adaptive radix trees for nvm-based hybrid memory architecture," in *Proceedings of the 2020 IEEE International Conference on Big Data*, 2020, pp. 5867–5869.

[28] W. Zhang, H. Tang, S. Byna, and Y. Chen, "DART: distributed adaptive radix tree for efficient affix-based keyword search on hpc systems," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 24:1–24:12.

[29] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Towards practical and privacy-preserving multi-dimensional range query over cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3478–3493, 2022.

[30] W. Zhong, C. Chen, X. Wu, and S. Jiang, "REMIX: efficient range query for lsm-trees," in *Proceedings of the 19th USENIX Conference on File and Storage Technologies*, 2021, pp. 51–64.

[31] T. Ziegler, S. T. Vani, C. Binnig, R. Fonseca, and T. Kraska, "Designing distributed tree-based index structures for fast rdma capable networks," *Proceedings of the 2019 International Conference on Management of Data*, pp. 56–67, 2020.

[32] P. Zuo, Y. Hua, and J. Wu, "Write-optimized and high-performance hashing index scheme for persistent memory," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, pp. 461–476.

AutoClock: Automated Clock Management for Power-Efficient HLS Designs on FPGAs

Jiawei Liang[1,2], Linfeng Du[1], Xiaofeng Zhou[1], Zhe Lin[3], Jiang Xu[4], Wei Zhang[1,2,*]
[1]*The Hong Kong University of Science and Technology*, [2]*Guangzhou HKUST Fox Ying Tung Research Institute*
[3]*Sun Yat-sen University*, [4]*The Hong Kong University of Science and Technology (GZ)*
{jliangbr, linfeng.du, xzhoubu}@connect.ust.hk, linzh235@mail.sysu.edu.cn, jiang.xu@hkust-gz.edu.cn, wei.zhang@ust.hk

***Abstract*—High-level synthesis (HLS) tools streamline FPGA design by enabling engineers to implement hardware using C/C++ languages. However, while clock management serves as a critical stage in the FPGA EDA flow that affects system-level performance, area, and especially power consumption, existing commercial HLS tools lack comprehensive solutions for clock management. Specifically, the diversity of clock resources creates a vast design space for finding the optimal configuration, and the insufficient analysis of multiple clock domain scenarios hinders effective clock-oriented optimizations in HLS. This work introduces AutoClock, an open-source integrated clock management framework that complements AMD Vitis HLS. AutoClock allocates resources for clock generation, assigns modules to appropriate clock domains, addresses metastability and time division multiplexing (TDM) malfunctioning introduced by multi-clock domain architectures, and hierarchically gates the clock of modules in a design. Experimental results demonstrate that AutoClock can fully utilize clock resources on FPGAs and help reduce dynamic power consumption by up to 74.38%.**

***Index Terms*—High-level synthesis, clock management, clock domain crossing, clock gating, clock multiplexer.**

I. Introduction

Field-programmable gate array (FPGA) is widely used in various fields, such as autonomous driving [1]–[3], data center [4]–[6], and deep learning algorithm acceleration [7]–[9], owing to its reconfigurable nature with low latency. As applications become increasingly complex, the requisite time for their implementation on FPGA has increased exponentially. To address this problem, *high-level synthesis (HLS)* has gained significant traction as a methodology that enables engineers with limited hardware design expertise to use high-level programming languages for application implementation, largely bypassing the demand for learning specific low-level hardware minutiae.

Despite the popularity of HLS for FPGA, few commercial or academic tools have placed a focus on clock resource management, although the clock is one of the most essential components of an FPGA design. On the one hand, the use of clock resources is highly related to *performance, power, and area (PPA)* metrics of FPGA designs, particularly the power consumption [10], [11]. On the other hand, when serving as an *application-specific integrated circuit (ASIC)* prototyping platform, FPGA should maintain the same clock behaviors as on ASIC. The inherent disparities in clock resources between FPGAs and ASICs have historically necessitated manual reconstruction of clock topology for FPGA-based prototypes, which involves a time-consuming process. Consequently, empowering HLS to manage clock resources automatically would not only mitigate the power dissipation of FPGA designs but also enable HLS to support more complex applications while accelerating the overall development cycle for engineers.

Clock resources available in AMD FPGAs include clock buffers such as BUFGMUX and BUFGCE, as well as clock management tiles containing *phase-locked loop (PLL)* and *mixed-mode clock manager (MMCM)* [12]. Though previous works have explored various approaches for managing clock resources in FPGA designs, their efforts addressed only a limited spectrum of obstacles. Some challenges remain when seeking to fully leverage these resources from the HLS level:

- Existing studies have largely focused on PLLs for clock generation [13] or conceptualized digital clock management modules without delving into the practicalities of resource selection [10]. However, modern FPGAs, such as those from AMD, offer a diverse array of clock resources with similar functionalities but distinct power profiles. This diversity, coupled with the finite availability of these resources, presents a considerable challenge in optimizing power consumption while meeting design specifications.
- Besides, prior works have predominantly address specific *clock domain crossing (CDC)* scenarios, such as memory sharing across two clock domains and handshake mechanisms [13], or double buffering [14]. Various types of multi-clock domain designs present a challenge in automatically handling all types of CDC cases appropriately.
- Additionally, the limited number of clock gates in FPGAs demands a strategic allocation methodology. Nevertheless, existing HLS-based clock gating insertion work [15] has primarily dealt with small benchmarks and has not addressed the limitation of resources.
- Finally, while previous research [15] has explored clock gating in streaming modules using buffer status for back-pressure, real-world FPGA applications often encompass a broader range of modules, necessitating more versatile clock gating methods that can accommodate the diverse module interactions.

To address these challenges, this paper proposes the first integrated, automatic clock resource management framework, named AutoClock, aimed at presenting power optimization of HLS designs on the AMD Vitis flow. AutoClock leverages

* Corresponding author.

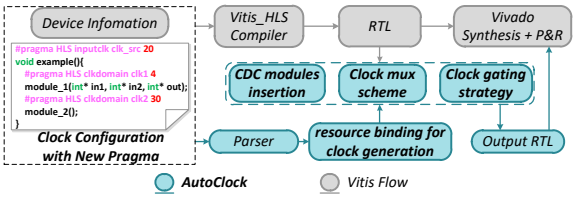


Fig. 1. Overview architecture of AutoClock.

clock buffers and clock management tiles while considering a broader range of CDC and clock gating scenarios. Additionally, the quantity and power consumption of these resources are considered to minimize overall power cost. Our contributions are highlighted as follows:

- We develop an open-source clock resource management framework that addresses a significant gap in current HLS tools. The open-source repository is available at: https://github.com/MasLiang/AutoClock.
- By formulating the resource binding of clock generation as an *integer linear programming (ILP)* problem featuring iterative constraint refinement, and coupling it with a greedy hierarchical clock gating strategy, AutoClock minimizes the overall power consumption of FPGA designs.
- AutoClock provides adaptive automatic CDC modules insertion for various interfaces and signals, as well as a clock multiplexer-based scheme to handle diverse multiple clock domain scenarios.

II. AutoClock Framework Design

The overall flow of the proposed clock management framework is illustrated in Fig. 1. Firstly, a pair of new pragmas are introduced that allow users to configure clock domains within an HLS design, accompanied by a dedicated parser to process these pragmas. Based on the parsing result, AutoClock generates a *clock reset generator (CRG)* for clock generation and reset synchronization while binding modules and related signals to different clock domains. Subsequently, AutoClock inserts clock gates into various types of modules in each clock domain using a greedy, hierarchical strategy to significantly minimize dynamic power consumption. Afterward, issues introduced by multiple clock domains are solved by inserting appropriate CDC modules automatically for various types of signals, including async-FIFO, async-BRAM, level- and pulse-synchronizer, to ensure reliable data transfer between clock domains, and applying clock multiplexer scheme to handle TDM modules invoked by multiple clock domains. Finally, the modified RTL code with clock management is fed into Vivado for further implementation.

A. Clock Configuration with New Pragmas

For compatible with Vitis HLS with supporting multiple clock domains configuration, a pair of new custom pragmas is proposed as listed in Table I. With these pragmas, users can easily configure the clock domain and target frequency of each HLS function. The parser processes these pragmas, using naming conventions to associate configurations with corresponding RTL modules. The pragmas are then removed from the project to ensure compatibility with Vitis HLS, while the extracted clock domain information is used in subsequent steps of the AutoClock framework.

TABLE I
AutoClock-specific Pragmas: Templates and Descriptions.

Coding Template	Description
`#pragma HLS inputclk <name> <frequency>`	Used before the top module to define clock frequency of the input clock.
`#pragma HLS clkdomain <name> <frequency>`	Used before a module to define the clock domain of this module.

B. Resource Binding for Clock generation

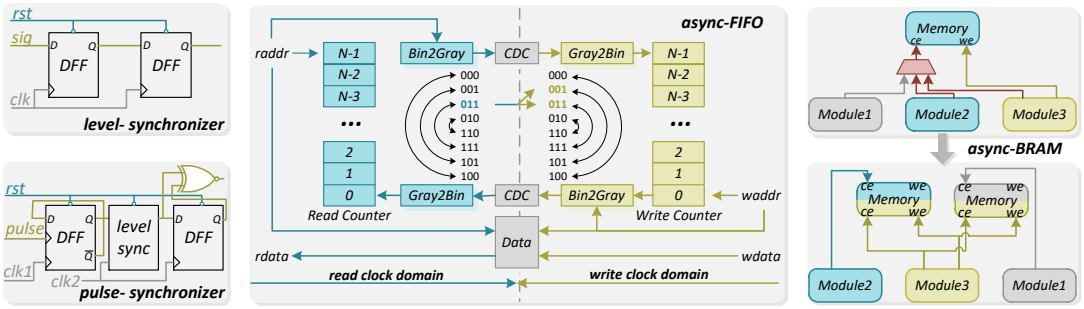
Given the clock frequencies specified in the user configuration, multiple types of clock resources can be leveraged for generating them, namely BUFGCE_DIV and clock management tiles, including MMCM and PLL, in which MMCM can support the most complex functionality while BUFGCE_DIV costs the least power. Our purpose is to find the optimal binding of these resources that can satisfy the design requirements and consume minimum power. The problem is formulated as:

Problem. *Given the input system clock frequency f_{in}, the list of clock frequencies needing to generate $\boldsymbol{F} = \{f_i\} \in \mathbb{R}$, and the set of available clock resources $\boldsymbol{S}$. The goal of resource binding for clock generation is to identify a set of clock resources S_f that fulfill $\boldsymbol{F} = g(f_{in}; \boldsymbol{S}_f)$ while minimizing the power consumption under the constraints of resource quantity and design performance.*

Given the large number of clock resources and broad exploration space, the problem is formulated as an ILP problem featuring iterative constraint refinement for speedup. To simplify the problem, we first traverse all the frequency values to find the ones that BUFGCE_DIV can generate under the constraint of resource number. Subsequently, the remaining part of the problem can be solved as an ILP problem:

$$\begin{aligned} & min(n_{PLL} \times p_{PLL} + n_{MMCM} \times p_{MMCM}) \\ & s.t.\ n_{PLL} \le N_{PLL} \\ & \quad n_{MMCM} \le N_{MMCM} \\ & \quad Nf_{PLL}^{max} \times n_{PLL} + Nf_{MMCM}^{max} \times n_{MMCM} \ge n_{freq} \\ & \quad n_{PLL} \in \mathbb{Z} \\ & \quad n_{MMCM} \in \mathbb{Z} \end{aligned} \tag{1}$$

where n_s and p_s mean the number and power consumption of clock resource of type s, respectively, N_s is the upper limit of each type of resource, Nf_s^{max} represents the maximum number of frequency that a type of resource can generate. After obtaining the solution, the feasibility of it is validated by considering additional physical constraints. Frequencies in $\mathbf{F}$ are grouped into subsets based on the solution above, namely $\{\mathbf{F}_i^{MMCM}, \mathbf{F}_j^{PLL}\}$ representing clock frequencies bound to an MMCM or a PLL. Each subset must satisfy the requirement of its corresponding resource. Take a subset of clock frequency



(a) (b) (c)

Fig. 2. CDC modules. (a) single-bit synchronizers for level- and pulse- signals. (b) async-FIFO. (c) async-BRAM.

bound to an MMCM as an example. MMCM contains a global multiplier, a global divider, and some specific dividers for each output clock. Input frequency is processed by the global multiplier and divider to produce an intermediate value, which is then divided by specific dividers to generate output frequencies. Therefore, there are certain constraints associated with the MMCM:

$$\begin{aligned} &\exists \alpha, \beta \in \mathbb{R} \\ s.t. &\frac{\alpha}{f_{in}} \in \textbf{factor}_{mult_g} \\ &\frac{\alpha}{\beta} \in \textbf{factor}_{div_g} \\ &\frac{\beta}{f_k} \in \textbf{factor}_{div_k} \end{aligned} \tag{2}$$

in which α and β represent results of the global multiplier and divider, $\textbf{factor}_{mult_g}$, $\textbf{factor}_{div_g}$, and $\textbf{factor}_{div_k}$ denotes the valid factors of the global multiplier, the global divider, and divider for $k-$th output clock in an MMCM, respectively. PLL has a similar constraint. It should be noted that there are many combinations when grouping these clock frequencies, and a solution is feasible as long as there is a combination that meets the requirements. If a solution is infeasible, a new constraint is introduced to exclude it from the solution space:

$$s.t.\ n_{PLL} \times p_{PLL} + n_{MMCM} \times p_{MMCM} > P_{solution} \tag{3}$$

where the $P_{solution}$ is the power consumption of the failed solution. Then the ILP solver will be exerted again with the new constraint. This process is iterated until a feasible solution is found.

C. CDC Modules Insertion

Since different clock domains have varying features, such as frequency and phase, signals crossing clock domains must be synchronized to prevent metastable states from being sampled and propagating, which leads to incorrect circuit behavior. To satisfy this requirement, a series of CDC modules, including single-bit synchronizers, namely synchronizers for pulse- and level-signals, as well as multi-bit synchronizers, i.e., *asynchronous first-in first-out (async-FIFO)* and *asynchronous block random access memory (async-BRAM)*, are tailored to deal with a different type of signals.

1) Single-bit synchronizer: There are two types of single-bit signals: level signals and pulse signals, between which level signals are the most common in digital circuits. The top part of Fig. 2a shows a two-flip-flop synchronizer for single-bit level signals. Since it is used as a part of the handshake mechanism in [13], there is no need to care about the relationship between the frequency of two clock domains. However, this synchronizer has limitations when being applied independently. As illustrated in Fig. 3a, the width of the signal in the faster clock domain might be narrower than the period of the slower clock domain, leading to the risk of missing sampling. To mitigate this risk, AutoClock expands those level signals crossing the clock domain as Fig. 3b so that they can always be properly sampled in the slower clock domain.

Pulse signals are commonly used in control circuits. When this type of signal crosses clock domains, the two-flip-flop synchronizer can not fulfill the requirement. For instance, a pulse from the slower clock domain that sustains longer than twice the period of the faster clock domain will be sampled twice, generating a level signal after synchronizing. Therefore, a dedicated pulse-synchronizer is employed, shown in the bottom part of Fig. 2a. In this synchronizer, the input pulse serves as an enable signal to control the conversion of a level signal, which will cross the clock domain boundary relying on the level-synchronizer. In the destination clock domain, the edge of the synchronized level signal triggers the generation of a pulse signal. Fig. 3d gives the timing diagram for this process. Due to the constraint of level-synchronizer, the width of the level signal, i.e., the interval between two pulses T_{pulse}, must exceed the period of the target clock.

In addition to regular signals, the reset signal is another significantly important type of signal. Similar to regular signals needing to satisfy setup and hold time constraints, reset signals must satisfy the requirement of the recovery and removal time constraint as Fig. 3c, or metastable state might propagate. As a result, reset signals must also be synchronized for each clock domain when de-assertion. The reset-synchronizer shares the same module with the level-synchronizer depicted in the top part of Fig. 2a with the input signal set as constant 0.

2) multi-bit synchronizer: CDC for multi-bit signals introduces more complexity. Although the reliability of two-flip-flop based level-signal synchronizer has been proven in [13]

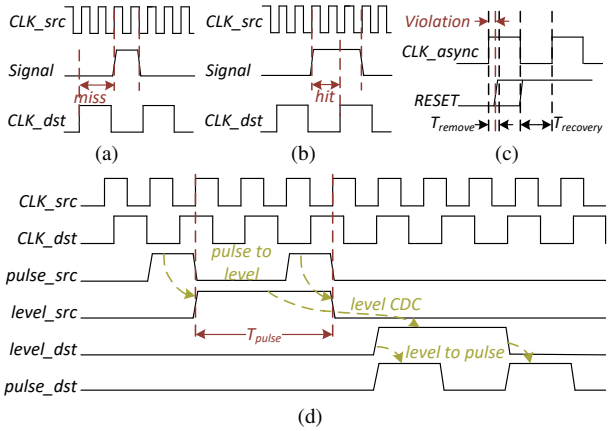


Fig. 3. Timing analysis of synchronizers. (a) Miss of signal sampling. (b) After expanding, the signal is sampled by the slower clock. (c) A violation example of removal time. (d) The timing of the pulse synchronizer.

to have an enormous *mean time before failure (MTBF)*, the MTBF of n parallel level-synchronizers for multi-bit signals will decrease by n times [16]. To address this problem, async-FIFO and async-BRAM are applied. While there are some other types of CDC modules, such as synchronizers based on handshake mechanism [16], they can be viewed as variations of previous basic synchronizers and will not be analyzed in detail here.

Async-FIFO is used when sequential multi-bit signal transfers. As illustrated in Fig. 2b, an async-FIFO includes shared data storage space and two address pointers binding to two clock domains. Pointers are converted to Gray Code before being synchronized because only a single bit changes in Gray Code when the address value increases. In this way, the MTBF is equal to a single-bit synchronizer. Moreover, there are only two conditions when Gray Code crosses clock domains: 1) successfully sampling and 2) failing to sample the changed bit that the value remains unchanged. For instance, in Fig. 2b, when the original *r-address* changes from 001 to 011, the synchronized *r-address* will either change to 011 successfully or remain at 001. If it remains, the state of the *r-pointer* in the write clock domain does not change. Thus, async-FIFO can guarantee the correctness of reading and writing.

BRAM is widely used as a shared data storage space among modules for transferring data. When a BRAM is shared by two clock domains, two BRAM ports are bound to different clock domains [13]. However, BRAM might be scheduled with time division multiplexing to be accessed by multiple modules for minimizing resource cost, e.g., as shown in the top part of Fig. 2c, $module3$ writes data to the BRAM while $module1, 2$ from different clock domains read data via a multiplexer. In this case, the BRAM will be copied for each read clock domain and memory accessing via multiplexer is separated to multiple independent accessing.

D. Clock multiplexer scheme

To limit resource allocation in the implemented design and reduce resource utilization, Vitit-HLS supports TDM for

Pseudo-Algorithm 1. Hierarchical greedy clock gating strategy

Input: the number of BUFGCEs g_n
the number of level L
the number of modules at l-th level N_l
the number of gated module $N_g = 0$
input design with a set of modules $M = \{m_0^0, ...m_{n-1}^{L-1}\}$

Output: Set of gated module M_g

```
for l = 0 to L do
    for n = 0 to N_l do
        calculate power that can be saved using Equation. 6
    sort modules by saved power
    for n = 0 to N_l do
        if father module is dataflow module do
            skip this module
        else do
            insert BUFGCE for clock gating
            N_g += 1
            if N_g == g_n do
                break
     if N_g == g_n do
        break
```

modules. However, multi-clock domains strategy makes it malfunctioning because a module is confined to a single clock domain. Thus, invoking the same module in multiple clock domains requires creating copies for each clock domain. It will introduce additional resource utilization as well as static power cost. To address this issue, a clock multiplexer scheme is proposed. For modules utilized across multiple clock domains, a new clock domain is created that incorporates multiple clocks with different frequencies corresponding to clock domains initiating these modules. In the protocol from Xilinx official document [17], an *finite state machine (FSM)* is implemented to control status of each kernel. A selection signal is then generated based on the FSM states to indicate which clock domain is invoking the module. With this signal, a BUFGMUX is instantiated to select the specific clock frequency. By implementing this scheme, the module can still function as a TDM module. Additionally, to mitigate potential signal instability because of frequency conversion, redundant idle states are inserted to the FSM. This remains sufficient time for the clock multiplexing process to ensure stable operation.

E. Clock Gating Strategy

Dynamic power dissipation, as described by Eq. 4, is closely related to active ratio. When signals remain, clock-related transistors still guzzle energy since their active ratio is 100%, including logic elements like LUT and DFF, as well as buffers and programmable switches on the route of the clock tree. To save such unnecessary power consumption, BUFGCEs are inserted into specific modules as clock gates in the clock tree to disable clock signals when their associated functions are idle. Due to the limited number of BUFGCEs on a given FPGA, a hierarchical greedy clock gating strategy, as illustrated in Pseudo-Algorithm 1, is advised for efficiently binding these BUFGCEs to maximize power saving.

$$\begin{aligned} P_{total} &= P_{static} + P_{dynamic} \\ &= Vdd \times I_{leakage} + \sum_{w \in W} \alpha_w C_w Vdd^2 f \end{aligned} \tag{4}$$

Firstly, there are two kinds of modules, i.e., dataflow modules whose sub-modules run continuously in the pipeline and non-dataflow modules where sub-modules exert in sequence. Due to the continuous working of sub-modules in a well-optimized dataflow module throughout the entire module execution, independent clock gating for them is unnecessary. For non-dataflow modules, some of their sub-modules might remain inactive when they are exerting. Thus, clock signals can be gated during the idle of these sub-modules to reduce power consumption. Previous works [11], [15] only care about dataflow modules and formulate them as an actor controlled through FIFO interface for inserting gates. When expanding to non-dataflow modules, the FIFO interface is not enough to cover all cases. Thus, the state of FSM in each module is used. Namely, when the FSM is in idle state, the clock is disabled.

Based on Xilinx documents [17], there are three block-level control protocols: *ap_ctrl_chain*, *ap_ctrl_hs*, and *ap_ctrl_none*. To generate an FSM for a block, *ap_ctrl_none* protocol is prohibited from use. In other protocols, the two signals, *ap_start* and *ap_done*, indicate the states of the FSM, thus are used to assert and de-assert the enable signals of clock gates. It is worth noting that the clock driving the FSM should always be on. Otherwise, deadlock will occur: if the clock is disabled, the state of the FSM will never change, and the clock will not be enabled. Additionally, the synchronous reset strategy is widely used in FPGA design, so the clock should be enabled when synchronous reset signals are asserted.

Besides, when a BUFGCE is inserted into a module, it influences circuits within this module and its sub-modules. Consequently, gating higher-level modules can save power more effectively than gating lower-level modules. Based on this, clock gates are inserted into modules from higher to lower levels. At the same level, all modules are sorted by power saving, which is closely related to two factors. First, the amount of saved power is linearly related to gated time. Second, modules with more transistors will result in more power saving in the same number of gated cycles. In a conclusion, the power that can be saved in a module is represented by:

$$P_{saved} = \frac{C_{idle}}{C_{total}} \times \sum_{t \in T} P_t \tag{5}$$

Where C is the number of cycles extracted from Vitis HLS reports, T is the set of basic elements in FPGA related to clock signals, including buffers in clock tree, LUTs, registers, DSP, and BRAM, and P_t is the power of an electronic element t in a cycle, obtained from AMD Xilinx Power Estimator. Since the idle time of a module overlaps with its higher-level modules, the clock gating time will also be calculated repeatedly. As a result, those time slots that have already been gated by clock gates in higher-level modules should be excluded when comparing with other modules. Thus, the Eq. 5 becomes:

$$P_{saved} = \frac{C^l_{idle} - \sum_{k=0}^{l-1} C^k_{idle}}{C_{total}} \times \sum_{t \in T} P_t \tag{6}$$

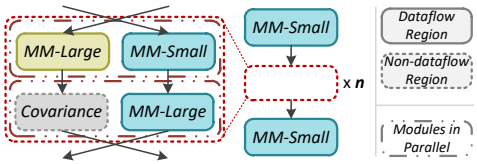


Fig. 4. Topology of of benchmarks PnMC. Different **background** color represent different clock domains.

in which C^l_{idle} and C^k_{idle} are the number of idle cycles of module at level l and its higher level modules, respectively. Subsequently, modules are sorted by the amount of saved power, and clock gates are allocated to those modules that have greater power savings.

III. Experiment

A. Experimental Setup and Benchmarks

The proposed AutoClock is implemented in Python with the Pyverilog tool [18] to parse and generate RTL code. All experiments are conducted on the AMD Alveo U280 FPGA based on Vitis 2023.2. *Xilinx Run Time (XRT)* is applied to measure power consumption, with power counter readings taken at *1ms* intervals. Each benchmark is executed 10,000 times successively to obtain a reasonable average power cost. Each experiment is repeated 20 times to eliminate random errors. For normalization, the system clock frequency of all benchmarks is set to *100MHz*.

To effectively evaluate AutoClock, a series of benchmarks are designed. Since the Vitis flow lacks clock management features, conventional HLS design benchmark suites such as Vitis Libraries [19] and Rosetta [20] are developed with a single clock domain, thus not clearly presenting the advantages of AutoClock. Similar to [21], we generate two dataflow kernels using AutoSA [22], namely *MM-Large* and *MM-Small* with the same matrix multiplication function but different latency, and a non-dataflow kernel *Covariance* from PolyBench [23]. We combine these to create composite large designs for generality in non-dataflow protocol. As these kernels have different latencies, they can be bound to different clock domains when working in parallel. Fig. 4 illustrates the structure of benchmarks used in this section, where PnMC represents cascading n parallel blocks. The background colors represent the clock domains to which these kernels belong. In the parallel block, two pairs of kernels work in parallel for multi-clock domain strategy analysis. They are also designed to demonstrate the effectiveness of the clock multiplexer scheme, as *MM-Large* works across different clock domains.

B. Clock Gating Evaluation

To thoroughly evaluate the impact of clock gating, we limited the number of BUFGCEs to 10 to address corner cases. Table II presents the comparison results. As benchmarks expand from P1MC to P3MC, the power reduction ratio consistently remains high, demonstrating the effectiveness of the clock gating method regardless of design scale. Additionally, since static power consumption is clock-independent, we focus on dynamic power. The results indicate that dynamic

TABLE II
POWER EVALUATION WITH MULTI-CLOCK DOMAINS STRATEGY AND CLOCK GATING ON BENCHMARKS WITH PARALLEL BLOCKS.

Benchmark	Total power (W)	Dynamic power (W)	Resource		
			LUT	FF	BRAM
P1MC	1.58	0.55	256235 (19.65%)	335103 (12.85%)	906.5 (44.97%)
P2MC	1.53	0.88	341516 (26.20%)	445657 (17.09%)	1326 (65.77%)
P3MC	1.95	1.31	426115 (32.69%)	556248 (21.33%)	1745.5 (86.58%)
P1MC_cg	0.87 (-23.92%)	0.27 (-50.20%)	257203 (+0.08%)	339255 (+0.16%)	906.5 (+0.00%)
P2MC_cg	1.01 (-34.12%)	0.36 (-59.57%)	341867 (+0.02%)	450863 (+0.20%)	1326 (+0.00%)
P3MC_cg	1.13 (-41.90%)	0.49 (-62.35%)	426601 (+0.03%)	563456 (+0.28%)	1745.5 (+0.00%)
P3MC_cg_rand1	2.04 (+4.77%)	1.40 (+7.11%)	426457 (+0.03%)	556226 (+0.00%)	1745.5 (+0.00%)
P3MC_cg_rand2	1.93 (-1.09%)	1.29 (-1.63%)	426248 (+0.01%)	556261 (+0.00%)	1745.5 (+0.00%)
P3MC_cg_rand3	1.95 (+0.28%)	1.31 (+0.41%)	426230 (+0.00%)	556265 (+0.00%)	1745.5 (+0.00%)
P3MC_cg_dfs	1.70 (-12.84%)	1.06 (-19.10%)	426546 (+0.03%)	561953 (+0.22%)	1745.5 (+0.00%)
P3MC_cg_bfs	1.14 (-41.29%)	0.50 (-61.44%)	426409 (+0.02%)	561411 (+0.20%)	1745.5 (+0.00%)
P1MC_cdc	1.09 (-4.93%)	0.54 (-9.48%)	255976 (-0.02%)	340237 (+0.20%)	922 (+0.76%)
P2MC_cdc	1.29 (-15.45%)	0.73 (-24.45%)	341534 (+0.00%)	452899 (+0.26%)	1357 (+1.54%)
P3MC_cdc	1.58 (-19.14%)	1.03 (-26.64%)	426159 (+0.00%)	565563 (+0.36%)	1792 (+2.31%)
P1MC_cdc_cg	0.76 (-33.84%)	0.21 (-65.12%)	257297 (+0.09%)	340268 (+0.20%)	922 (+0.76%)
P2MC_cdc_cg	0.81 (-47.01%)	0.25 (-74.38%)	341640 (+0.01%)	452915 (+0.28%)	1357 (+1.54%)
P3MC_cdc_cg	0.97 (-50.41%)	0.42 (-70.16%)	426533 (+0.03%)	565599 (+0.36%)	1792 (+2.31%)

power consumption can be reduced by up to 62.35%, similar to results from manually inserting clock gates at the RTL level [24], and greater than HLS-based work [15], which achieved a 53.7% reduction and focused solely on small dataflow designs.

Furthermore, to demonstrate the advantage of the proposed BUFGCE allocation strategy, random selection, *breadth-first search (BFS)*, and *depth-first search (DFS)* strategies were applied to P3MC to allocate a limited number of BUFGCEs. For random selection, conducted three times with different seeds, power cost increases in corner cases because the additional logic from clock gating consumes more power than it saves. The results demonstrate that our strategy achieves more significant power cost reduction. Resource utilization is also reported; although enabling signals introduces extra logic, the increases in LUT and FF usage are minimal.

C. Multi-Clock Domains Strategy Evaluation

The advantages of the multi-clock domains strategy are shown in Table II. In the baseline design, all modules within a parallel block run at the same clock frequency but exhibit varying latencies, causing faster modules to idle while waiting for slower ones. In such cases, modules with lower latency can be assigned a lower clock frequency. For example, the latency of *MM-Large* is 1/4 of *MM-Small*, corresponding to a clock frequency of *100MHz* $\times$ 1/4. As recorded in Table II, total and dynamic power can be reduced by up to 19.14% and 26.64%, respectively. Additionally, since BRAMs are used between kernels, extra BRAMs are required for CDC. However, only 2.31% of BRAMs for CDC are used in the corner case P3MC, which is acceptable. Given that BRAM is more likely to be the bottleneck than LUT, our benchmarks consist of numerical kernels, so async-FIFO and async-BRAM are widely used for CDC, helping explore the upper limit of BRAM usage. In designs with less data transfer and more control signals, the additional BRAM cost will be lower, while LUT consumption will be higher.

TABLE III
POWER AND AREA EVALUATION FOR CLOCK MULTIPLEXER SCHEME.

Flow	Total power(W)	Dynamic power(W)	LUT	DFF	BRAM
Vitis	0.75	0.29	230326	305926	762.5
AutoClock	0.67	0.21	230451	306992	799.5

We report the power consumption for these benchmarks using both methods. Clock gating proves to be more effective in saving power by disabling clock signals, while the multi-clock domain strategy only lowers frequency, even when modules are idle. These two methods are relatively independent and can be used simultaneously for greater power savings. In our benchmarks, dynamic power savings of up to 74.38% are achieved. Notably, although P3MC shows higher absolute dynamic power reduction, P2MC outperforms P3MC in power reduction ratio due to having fewer kernels and lower dynamic power. Subsequently, the TDM optimization strategy is applied to P1MC, limiting the *MM-Large* kernel in the RTL design to one instance. As shown in Table III, AutoClock still achieves a 25.58% reduction in power dissipation without compromising the effectiveness of the TDM optimization strategy and maintaining resource utilization.

IV. CONCLUSION

This paper introduces AutoClock, an open-source automatic clock management framework for HLS designs using the Vitis flow. AutoClock assists engineers in efficiently utilizing FPGA clock resources and reducing the power consumption of HLS designs. Our experiments demonstrate the effectiveness of AutoClock, achieving dynamic power savings of up to 74.38%.

V. ACKNOWLEDGMENT

This research is supported partly by National Key R&D Program of China (2023YFB4405100, 2023YFB4405101), General Research Fund (16214123), the National Natural Science Foundation of China (62404257), and Guangdong Basic and Applied Basic Research Foundation (2024A1515013155).

REFERENCES

[1] Y. Li, S. E. Li, X. Jia, S. Zeng, and Y. Wang, "Fpga accelerated model predictive control for autonomous driving," *Journal of intelligent and connected vehicles*, vol. 5, no. 2, pp. 63–71, 2022.

[2] A. Kojima, "Autonomous driving system implemented on robot car using soc fpga," in *2021 International Conference on Field-Programmable Technology (ICFPT)*, pp. 1–4, IEEE, 2021.

[3] Y. Liu, S. Hao, K. Huang, M. Yang, Z. Huang, X. Qi, Y. Li, L. Zhou, Y. Long, and J. Zhou, "An fpga-based ultra-high performance and scalable optical flow hardware accelerator for autonomous driving," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2024.

[4] C. Wang, K. He, R. Fan, X. Wang, W. Wang, and Q. Hao, "Cxl over ethernet: A novel fpga-based memory disaggregation design in data centers," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 75–82, IEEE, 2023.

[5] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, *et al.*, "The future of fpga acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 3, pp. 1–42, 2022.

[6] Z. Wang, H. Huang, J. Zhang, F. Wu, and G. Alonso, "FpgaNIC: An FPGA-based versatile 100gb SmartNIC for GPUs," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, (Carlsbad, CA), pp. 967–986, USENIX Association, July 2022.

[7] Y. Wong, Z. Dong, and W. Zhang, "Low bitwidth cnn accelerator on fpga using winograd and block floating point arithmetic," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 218–223, 2021.

[8] S. Abi-Karam, R. Sarkar, D. Xu, Z. Fan, Z. Wang, and C. Hao, "Inr-arch: A dataflow architecture and compiler for arbitrary-order gradient computations in implicit neural representation processing," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2023.

[9] X. Jia, Y. Zhang, G. Liu, X. Yang, T. Zhang, J. Zheng, D. Xu, Z. Liu, M. Liu, X. Yan, *et al.*, "Xvdpu: A high-performance cnn accelerator on the versal platform powered by the ai engine," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 2, pp. 1–24, 2024.

[10] G. Lhairech-Lebreton, P. Coussy, and E. Martin, "Hierarchical and multiple-clock domain high-level synthesis for low-power design on fpga," in *2010 International Conference on Field Programmable Logic and Applications*, pp. 464–468, 2010.

[11] F. Ratto, T. Fanni, L. Raffo, and C. Sau, "Mutual impact between clock gating and high level synthesis in reconfigurable hardware accelerators," *Electronics*, vol. 10, no. 1, p. 73, 2021.

[12] I. Advanced Micro Devices, "Ultrascale architecture clocking resources user guide." [Online], 2023.

[13] O. Ragheb and J. H. Anderson, "High-level synthesis of fpga circuits with multiple clock domains," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 109–116, IEEE, 2018.

[14] T. Liang, J. Zhao, L. Feng, S. Sinha, and W. Zhang, "Hi-clockflow: Multi-clock dataflow automation and throughput optimization in high-level synthesis," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, IEEE, 2019.

[15] E. Bezati, S. Casale-Brunet, M. Mattavelli, and J. W. Janneck, "Clock-gating of streaming applications for energy efficient implementations on fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 4, pp. 699–703, 2017.

[16] S.-J. Kim, J.-G. Lee, and K. Kim, "A parallel flop synchronizer for bridging asynchronous clock domains," in *Proceedings of 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits*, pp. 184–187, 2004.

[17] "Vitis high-level synthesis user guide (ug1399)." [Online]. https://docs.amd.com/r/en-US/ug1399-vitis-hls/pragma-HLS-allocation.

[18] "Pyverilog." [Online]. https://github.com/PyHDI/Pyverilog/.

[19] "Xilinx. 2022. vitis accelerated libraries." [Online]. https://github.com/Xilinx/Vitis_Libraries/.

[20] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, Feb 2018.

[21] L. Du, T. Liang, S. Sinha, Z. Xie, and W. Zhang, "Fado: F loorplan-a ware d irective o ptimization for high-level synthesis designs on multi-die fpgas," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 15–25, 2023.

[22] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021.

[23] T. Y. Louis-Noel Pouchet, "Polybench/c." [Online], 2016.

[24] M. H. Vo, "The merged clock gating architecture for low power digital clock application on fpga," in *2018 International Conference on Advanced Technologies for Communications (ATC)*, pp. 282–286, 2018.

NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph

Wenji Fang[1], Wenkai Li[1], Shang Liu[1], Yao Lu[1], Hongce Zhang[2], Zhiyao Xie[1*]
[1]Hong Kong University of Science and Technology
[2]Hong Kong University of Science and Technology (Guangzhou)
[*]Corresponding Author

***Abstract*—Circuit representation learning has shown promise in advancing Electronic Design Automation (EDA) by capturing structural and functional circuit properties for various tasks. Existing pre-trained solutions rely on graph learning with complex functional supervision, such as truth table simulation. However, they only handle simple and-inverter graphs (AIGs), struggling to fully encode other complex gate functionalities. While large language models (LLMs) excel at functional understanding, they lack the structural awareness for flattened netlists. To advance netlist representation learning, we present NetTAG, a netlist foundation model that fuses gate semantics with graph structure, handling diverse gate types and supporting a variety of functional and physical tasks. Moving beyond existing graph-only methods, NetTAG formulates netlists as text-attributed graphs, with gates annotated by symbolic logic expressions and physical characteristics as text attributes. Its multimodal architecture combines an LLM-based text encoder for gate semantics and a graph transformer for global structure. Pre-trained with gate and graph self-supervised objectives and aligned with RTL and layout stages, NetTAG captures comprehensive circuit intrinsics. Experimental results show that NetTAG consistently outperforms each task-specific method on four largely different functional and physical tasks and surpasses state-of-the-art AIG encoders, demonstrating its versatility.**

I. Introduction

Machine learning (ML) techniques have demonstrated remarkable achievements in electronic design automation (EDA). Existing ML methods are mostly tailored to specific tasks such as predicting physical metrics (e.g, timing [1], [2], [3], area [4], [5], [6], power [7], [8], [9], and routability [10], [11], [12]) or the reasoning of circuit functionalities [13], [14], [15], [16]. These methods are typically developed by supervised training, requiring extensive label collection and model customization for each task. Despite obvious effectiveness, they are time-consuming to develop and lack generalizability, often capturing only task-specific patterns rather than a generalizable understanding of circuits.

Recent circuit representation learning methods (i.e., encoders) generate informative embeddings for circuits and are able to support a range of downstream tasks [17], [18]. Existing methods mainly focus on VLSI circuit netlists, and some works also explore RTL circuits [19], analog circuits [20], and FPGAs [21]. We provide a detailed summary of the netlist encoders in TABLE I. All these netlist encoders capture circuit functionality through graph structure leveraging graph learning models, such as Graph Neural Networks (GNNs) or Graph Transformers (GTs). They either directly infer functionality with graph topology [22], [23], [24], or employ functional information to pre-train the models [25], [26], [27], [28], [29], [30], [31].

However, we argue that existing graph-only netlist encoders lack sufficient expressiveness to capture complex circuits. **These methods rely on graph models that inherently prioritize *structural* information over *semantics* (i.e., functionality).** Their limitations include: *(1) Limited to and-inverter graph (AIG).* Most methods target only simple AIG-based netlists, representing a narrow subset of standard cell libraries, and cannot handle post-mapping netlists with diverse gate types. *(2) Dependence on complex functional supervision.* Representative methods like the DeepGate family [25], [26], [27], [30] and FGNN [28], [29] incorporate truth table simulation to capture AIG functionality. However, truth tables suffer from exponential state expansion when applied to post-synthesis netlists with multi-input complex gates (e.g., full adders, multiplexers), limiting their applicability. *(3) Lack of physical information.* None of these methods consider physical characteristics, focusing solely on logic functionality. Consequently, these encoders are limited to AIG-based tasks (e.g., equivalence checking, SAT solving) and struggle with tasks for post-mapping netlists [2], [3], [7], [10], [11], [12], [14], [15], especially those evaluating physical design quality.

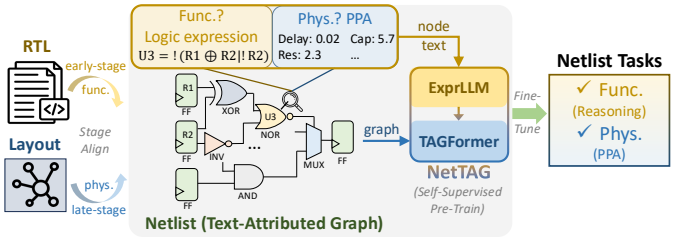


Fig. 1: Overvew of NetTAG. Netlists are formulated as text-attributed graphs, with functional and physical text attributes extracted for each gate. Within NetTAG, gate attributes are initially encoded by ExprLLM, then refined with global netlist graph structures using TAGFormer. NetTAG is pre-trained with self-supervised objectives and aligned with RTL and layout embeddings, enabling versatile support for both functional and physical tasks after fine-tuning.

Recently, large language models (LLMs) have demonstrated remarkable expressiveness in circuit-related generative tasks [32], [33], [34]. **However, LLMs inherently capture circuit textual *semantics* rather than *structure*.** Limited work has explored LLMs for general-purpose circuit representation learning, with key limitations summarized as follows: *(1) Struggle with netlists.* Although LLMs can interpret circuit functionality from RTL code, the gate-level netlists are more flattened and lack informative context, making functional understanding more challenging. *(2) Lack of structural encoding.* LLMs struggle to capture the circuit structures, limiting their utility for netlist representation learning.

In this work, we present **`NetTAG`**, a foundation model[1] for netlists that captures functional and physical properties across diverse gate types. Unlike previous circuit encoders and LLM solutions focusing on single circuit modality, NetTAG fuses gate text *semantics* with global graph *structures* to achieve *functional* and *physical* understanding. Serving as a foundation model, the pre-trained NetTAG generates versatile embeddings for logic gates, register cones, and full circuits. The embeddings can be easily fine-tuned for various downstream tasks, supporting largely different netlist-stage functional and physical tasks across these circuit granularities. As shown in Fig. 1, we propose the following innovative strategies in NetTAG:

- **Preprocess: formulating netlists as text-attributed graphs.** This paper is the first to represent netlists in text-attributed graphs (TAGs) format. In the netlist graph, we annotate each gate with functional symbolic logic expressions and physical characteristics as node-level text attributes. Our TAG format combines gate textual attributes with graph topology, moving beyond existing graph-only circuit representation learning.

[1]The code and pre-trained NetTAG model are available at https://github.com/hkust-zhiyao/NetTAG. The pre-trained model enables users to easily generate and fine-tune embeddings for their own netlist tasks.

TABLE I: Comparision of state-of-the-art netlist representation learning methods.

Method	Target Circuit		Encoding Methodology				Downstream Tasks	
	Cell Type	Circuit Type	Modality	ML Model	Pre-Train Target	Cross-Stage Align	Target	Type
DeepGate1/2 [25], [26]	AIG	Comb.	Graph	GNN	Gate	N/A	Gate	Func.
DeepGate3 [27]	AIG	Comb.	Graph	GT	Gate&Circuit	N/A	Gate	Func.
FGNN [28], [29]	AIG	Comb.	Graph	GNN	Circuit	N/A	Gate&Circuit	Func.
HOGA [23]	AIG	Comb.&Seq.	Graph	GNN	N/A	N/A	Gate&Circuit	Func.&Phys.
NetTAG (Ours)	**Any Gate**	**Comb.& Seq.**	**Text& Graph**	**LLM& GT**	**Gate& Reg. Cone**	**RTL& Layout**	**Gate& Reg. Cone& Circuit**	**Func.& Phys.**

- **Multimodal architecture: fusing semantic and structure.** Leveraging TAGs, NetTAG introduces an innovative multimodal architecture that combines an LLM-based text encoder (i.e., ExprLLM) with a graph transformer (i.e., TAGFormer). ExprLLM first encodes *fine-grained* gate text attributes into semantic-rich node embeddings, then TAGFormer refines them with *overall* graph structure, generating final individual gate and overall graph embeddings.
- **Aligned pre-training: self-supervised and cross-stage-aware.** We introduce four circuit-specific self-supervised objectives to pre-train NetTAG across circuit granularities, enhancing Boolean logic understanding and fusing gate semantics with the global graph structure. Moreover, we align netlist embeddings with those from RTL and layout stages to strengthen cross-stage functional and physical awareness.

We evaluate NetTAG on four largely different EDA tasks, covering both functional reasoning and physical design quality predictions at different circuit granularities. NetTAG consistently outperforms all state-of-the-art (SOTA) task-specific models, achieving 14% and 13% higher accuracy in logic gate and register function identification, respectively, and reducing errors by 2% for register slack, and 7% for both circuit area and power predictions. NetTAG demonstrates strong generalization across these representative tasks. Finally, we evaluate the scalability of NetTAG by scaling up model size and dataset volume. The results indicate a huge potential for improvement by scaling up either model or circuit datasets in future work.

II. Methodology

A. Overview

Fig. 2 outlines the NetTAG workflow. First, netlist data is converted into TAG format, detailed in Section II-B. NetTAG's multimodal architecture, which combines fine-grained gate text and overall circuit graph refinement, is illustrated in Section II-C. Step 1 enhances Boolean logic comprehension for ExprLLM (Section II-D), and Step 2 fuses gate semantics with circuit structure in TAGFormer, incorporating cross-stage alignment to strengthen both functional and physical understanding (Section II-E). Finally, Section II-F demonstrates the fine-tuning strategy that leverages the pre-trained NetTAG for various downstream tasks.

B. Circuit Data Preprocessing

Conventional graph methods like GNNs and GTs mainly capture structural information but fall short when the semantic context is also needed. TAG representation learning bridges this gap by encoding nodes as text and leveraging graph connections [35], [36], effectively improving node classification performance. *Intuitively, netlists, with their graph structure and detailed gate logic functions, are ideally suited for formulation as TAGs.*

Formulating netlist as TAG. Given a gate-level netlist $\mathcal{N}$, we represent it as a text-attributed graph $\mathcal{G}_{\mathcal{N}} = \{\mathcal{T}, \mathcal{E}\}$, where each gate in the netlist is a node in the graph. Here, $\mathcal{T} = \{t_1, \ldots, t_m\}$ is a set of gate text attributes, where m representing the total gate count. Each attribute t_i contains functional and physical properties of the i^{th} gate, as shown in Fig. 3(b), and consists of a sequence of tokens t_i. $\mathcal{E}$ is the set of edges connecting the gates.

Specifically, for functionality, we extract the symbolic logic expression for each gate based on its k-hop fan-in cone. These expressions are constructed using a formal verification toolkit [37], providing formal definitions for gate functionality. An example is demonstrated

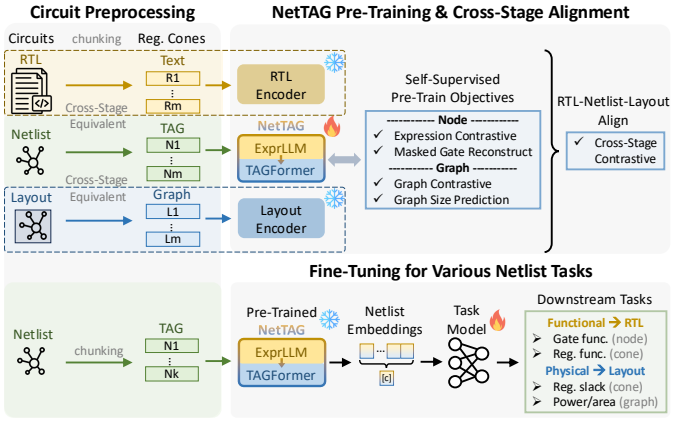


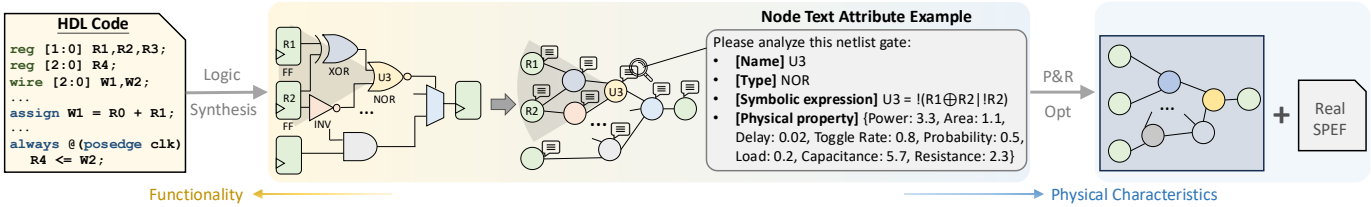
Fig. 2: NetTAG Workflow. Sequential netlists are chunked into combinational register cones and converted into TAGs. During pre-training, NetTAG is trained with node-level and graph-level self-supervised objectives, and it is aligned with RTL and layout embeddings. The pre-trained NetTAG then generates netlist embeddings, which are fine-tuned with netlist-stage task labels.

in Fig. 3(b), the 2-hop expression for this NOR gate with the symbolic name "U3" is derived from the Boolean functions of nodes within its 2-hop fan-in cone. The expression for "U3" is represented as: "U3 = !((R1 ⊕ R2)|!R2)". Each expression includes symbolic names for input gates and the Boolean operations. We summarize the key advantages of exploiting the symbolic expression for functional encoding:

1) **Versatile gate support:** Our symbolic expressions use Boolean formulas to represent any gate type, including complex gates like AOI and full adders. This enables NetTAG to support diverse gate types in netlists, beyond the limitations of AIG formats.
2) **Direct functional encoding:** In contrast to pre-trained circuit encoders that rely on graph-based learning with truth table supervision [29], [27], symbolic expressions enable straightforward static analysis, covering all input conditions without exponential growth problems by exhaustive truth table simulation.
3) **Structure independence:** Unlike structure-based methods that infer functionality with graph topology [23], [24], our approach derives gate expressions from Boolean formulas. It easily recognizes the same functionality across varying structures and distinguishes different functions in similar topologies.
4) **Compatible with language models:** Expressions can be seamlessly processed by language models, unleashing LLMs' reasoning capabilities to enhance circuit functional understanding. This functional expression also enables self-supervised pre-training for LLMs, such as expression contrastive learning, to improve LLMs' logic understanding from expression data itself.

For the physical characteristics, we annotate each gate with information extracted from the standard cell library, including characteristics such as power, area, delay, toggle rate, probability, load, capacitance, and resistance. Both functional and physical attributes are combined to form the text attribute for each gate.

Processing RTL and layout data. In addition to the target netlists, we utilize corresponding RTL and layout data to enhance NetTAG's cross-stage awareness. As shown in Fig. 3(a) and (c), RTL code is processed directly as text containing functional semantics, while layout data is represented as connectivity graphs annotated with physical characteristics. Specifically, nodes in the layout graphs are



(a) RTL Code (Text) (b) Netlist (Text-Attributed Graph) (c) Layout (Graph)

Fig. 3: Circuit data design stages and modalities. RTL code is processed directly as text. Netlists are represented as TAGs, with node attributes including gate name, type, symbolic expression, and physical property. Symbolic expressions are derived from each gate's k-hop input cone. Layout data is converted into graphs and annotated with physical information extracted from the SPEF file.

annotated with capacitance, resistance, and delay values extracted from the Standard Parasitic Extraction Format (SPEF) file.

Chunking sequential circuit into register cones. To handle large-scale sequential circuits, we propose chunking the original circuit into register cones. For each register, we backtrace through all driving logic up to other registers, creating a subcircuit cone that captures the complete state transition of the register, including update relationships and timing paths. This chunking process significantly reduces circuit size for our NetTAG, enhancing scalability. We apply this method consistently across netlist, RTL, and layout data, ensuring that cross-stage cones remain functionally equivalent[2] for subsequent alignment.

C. NetTAG Model Architecture

Fig. 4 illustrates the architecture of NetTAG, which comprises two main components: ExprLLM for gate-level text encoding and TAGFormer for further circuit-level graph encoding. Additionally, two auxiliary pre-trained encoders are employed to generate embeddings for the RTL code and layout graph, facilitating cross-stage alignment. We detail our NetTAG architecture below.

ExprLLM of NetTAG is initialized using an LLM-based text embedding model to encode gate text attributes as initial node embeddings. Following [38], we adapt a decoder-only LLM into a general-purpose text encoder by converting causal attention to bidirectional attention, which has shown superior encoding performance over encoder-only models (e.g., BERT) [39]. ExprLLM takes the gate text attributes t_i as input, converting them into the text embeddings T_i:

$$T_i = \text{ExprLLM}(t_i). \tag{1}$$

TAGFormer of NetTAG refines node embeddings from ExprLLM using a graph transformer model, which captures netlist graph structure via global attention. We include a `[CLS]` node connected to all other nodes in the graph, serving as the graph-level embedding. The initial embedding for each gate n_i is created by concatenating its text embedding T_i from ExprLLM with its physical characteristics vector $x_{\text{phys}i}$. TAGFormer then processes node embeddings $\{n_1, \ldots, n_m\}$ with their connectivity $\mathcal{E}$ into a sequence of netlist gate embeddings $\{N_1, \ldots, N_m, N_{\text{cls}}\}$, where N_{cls} represents the entire graph embedding:

$$\begin{aligned} n_i &= (T_i, x_{\text{phys}_i}), \\ \{N_1, \ldots, N_m, N_{\text{cls}}\} &= \text{TAGFormer}(\{n_1, \ldots, n_m\}, \mathcal{E}). \end{aligned} \tag{2}$$

Auxiliary RTL and Layout Encoders support cross-stage alignment for NetTAG by providing functional and physical insights from RTL and layout stages. The pre-trained RTL encoder, an LLM-based text encoder, generates RTL embeddings R_{cls}. Similarly, the layout encoder, a pre-trained graph transformer, produces layout embeddings L_{cls}. Please note that these auxiliary cross-stage encoders are used only during pre-training.

D. Pre-Training Step 1: Enhacing logic understanding in ExprLLM

We design pre-training objectives tailored for circuit netlists to capture both functional and structural information using our multimodal NetTAG. As shown in Fig. 4, the pre-training process is divided into two main steps: (1) Pre-training ExprLLM to strengthen its understanding of Boolean logic, and (2) Pre-training TAGFormer to fuse gate text semantics from frozen ExprLLM with circuit graph structure. Additionally, two pre-trained RTL and layout encoders are used for cross-stage alignment. We detail our pre-training objectives below.

Objective #1 Symbolic expression contrastive learning. In step1, we apply logic expression contrastive learning to enhance ExprLLM's understanding of Boolean expressions. As shown in Fig. 3, we begin by building a gate expression dataset from 2-hop symbolic expressions[3]. Then each expression is transformed using randomly applied Boolean equivalence rules[4] to generate a new positive sample. The InfoNCE loss [40] is then applied to differentiate each positive expression pair from other negatives, formulated as:

$$\mathcal{L}^{\#1}_{\text{expr}} = \mathcal{F}_{\text{CL}}(\boldsymbol{T}, T^+) = -\log \frac{\exp(T_{\text{ori}} \cdot T^+/\tau)}{\sum_{i=0}^{k} \exp(T_{\text{ori}} \cdot T_i/\tau)}, \tag{3}$$

where τ is the temperature scaling factor, T^+ is the positive embeddings, and $\boldsymbol{T}$ represents all k samples in the batch, including the original sample embeddings T_{ori} along with all other $(k-1)$ negative sample embeddings $T_i (i \neq \text{ori})$.

E. Pre-Training Step 2: Fusion in TAGFormer & Cross-Stage Align

1) Pre-training within TAGFormer for semantic and structure fusion: We propose gate-level and graph-level self-supervised objectives to jointly pre-train TAGFormer, fusing semantic-rich gate embeddings from pre-trained and frozen ExprLLM with the global circuit structure. The pre-training objectives are illustrated below:

Objective #2.1 Masked gate reconstruction. This objective aims to capture the structural roles of various gates by leveraging their connectivity within the netlist graph. Specifically, we propose to randomly mask a subset of gates in the netlist using a special `[MASK]` node. The model is then tasked with predicting the gate type (e.g., NOR, MUX, AND, etc.) of the masked gates based on the remaining unmasked nodes.

This objective is formulated as a multi-class classification problem, where the predicted gate types are treated as discrete labels. NetTAG first processes the masked graph to generate inputs and generates masked node embeddings N_i^{mask} for each masked gate in $\mathcal{T}^{\text{mask}}$. Then a classification Multi-layer Perceptron, denoted as $\text{MLP}_{\text{class}}$, takes the masked node embeddings N_i^{mask} as input and outputs the predicted probability distribution over gate types. The ground-truth gate type for each masked node i is denoted by $\boldsymbol{y}_i^{\text{mask}}$. The classification loss is defined using the cross-entropy loss as follows:

$$\mathcal{L}^{\#2}_{\text{gate}} = -\sum_{i \in \mathcal{T}^{\text{mask}}} \boldsymbol{y}_i^{\text{mask}} \log \left(\text{MLP}_{\text{class}}(N_i^{\text{mask}}) \right). \tag{4}$$

Objective #2.2 Netlist graph contrastive learning. This objective captures global structural and functional information by clustering similar netlists and separating dissimilar ones. Positive samples are generated via functionally equivalent transformations of each netlist graph [29], while negative samples are distinct graphs within the same batch. The InfoNCE loss is also applied to minimize the

[2]Most registers maintain unchanged throughout design stages.

[3]We choose 2-hop to balance the expression expansion and runtime.

[4]We implemented Boolean transformation rules such as De-Morgan's law, distributive law, commutative law, associative law, etc.

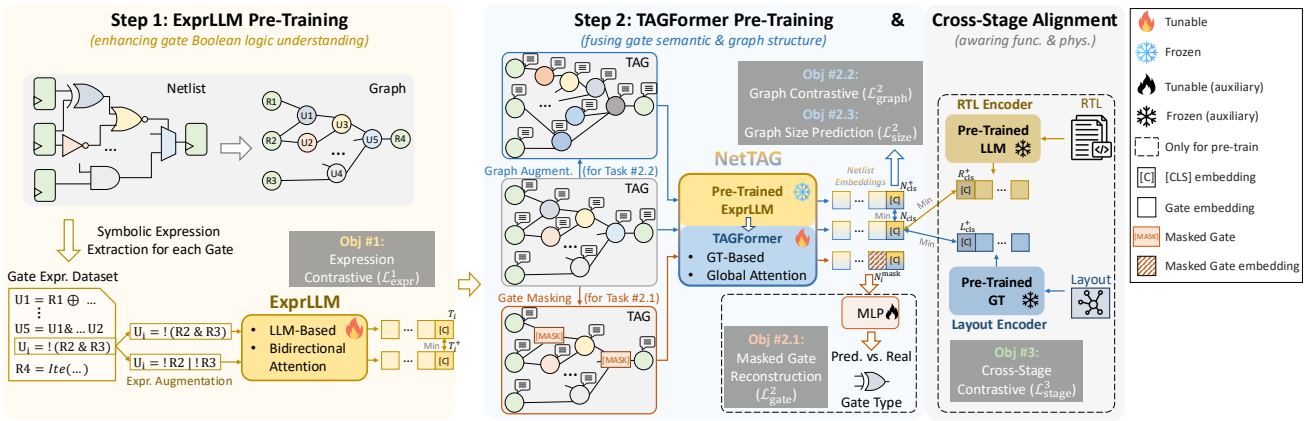


Fig. 4: NetTAG architecture and pre-training workflow. In step 1, we collect a dataset of gate symbolic expressions and pre-train ExprLLM to enhance its understanding of Boolean formulas. In step 2, ExprLLM is frozen, and we pre-train TAGFormer to fuse the gate semantics with the global graph structure. NetTAG embeddings are cross-stage aligned with those from pre-trained auxiliary RTL and layout encoders.

distance between positive pairs while maximizing separation from negative samples:

$$\mathcal{L}^{\#2}_{\text{graph}} = \mathcal{F}_{\text{CL}}(\boldsymbol{N}_{\text{cls}}, N^{+}_{\text{cls}}), \tag{5}$$

where N^{+}_{cls} denotes the positive sample embeddings, and $\boldsymbol{N}_{\text{cls}}$ includes the original and all other negative samples in the batch.

Objective #2.3 Netlist graph size prediction. This objective aims to predict the number of each type of gate in a given netlist graph based on the graph-level embedding, which is formulated as a regression problem. Denote $\boldsymbol{y}^{size}$ as the ground-truth gate counts for a netlist graph $\mathcal{G}_{\mathcal{N}}$. We employ an auxiliary model MLP_{regr}, which takes the graph-level embedding N_{cls} as input and outputs the predicted gate counts. The objective is to minimize the Mean Squared Error (MSE) between the predicted and actual gate counts, as formulated below:

$$\mathcal{L}^{\#2}_{\text{size}} = \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{y}^{size}_i - \text{MLP}_{\text{regr}}(N_{\text{cls}})\right)^2, \tag{6}$$

where n is the number of gate types, and each element in $\boldsymbol{y}^{size}$ corresponds to the predicted count for a specific gate type.

2) Pre-training beyond TAGFormer to enhance functional and physical awareness: **Objective #3 Cross-stage contrastive alignment.** Beyond the pre-training objectives within NetTAG's components, we introduce cross-stage alignment to integrate information from RTL-stage functionality and layout-stage physical implementation into NetTAG. This alignment enhances NetTAG's cross-stage awareness, benefiting both functional and physical downstream tasks, as further demonstrated in Section III-D. Using contrastive objectives, we align the circuit embeddings from RTL (i.e., R^{+}_{cls}), netlist (i.e., $\boldsymbol{N}_{\text{cls}}$), and layout (i.e., L^{+}_{cls}) within a shared latent space, as formulated below:

$$\mathcal{L}^{\#3}_{\text{align}} = \mathcal{F}_{\text{CL}}(\boldsymbol{N}_{\text{cls}}, R^{+}_{\text{cls}}) + \mathcal{F}_{\text{CL}}(\boldsymbol{N}_{\text{cls}}, L^{+}_{\text{cls}}). \tag{7}$$

To this end, we formulate the overall self-supervised pre-training objective of NetTAG as the following two-step process:

$$\mathcal{L}_{\text{NetTAG}} = \begin{cases} \mathcal{L}^1_{\text{expr}} & \text{(Step 1)} \\ \mathcal{L}^2_{\text{gate}} + \mathcal{L}^2_{\text{graph}} + \mathcal{L}^2_{\text{size}} + \mathcal{L}^3_{\text{align}} & \text{(Step 2)} \end{cases} \tag{8}$$

F. Fine-tuning for downstream tasks

NetTAG generates multi-grained embeddings for netlist elements, including combinational gates, register cones, and entire circuits. For sequential circuits, circuit-level embeddings are computed by summing the embeddings of all register cones, while for combinational circuits, their overall embeddings are directly obtained on the `[CLS]` node without the need for chunking into cones.

For downstream tasks, we fine-tune these embeddings with lightweight task models like MLPs or tree-based models (e.g., XGBoost). NetTAG's versatile embeddings support both regression and classification across functional and physical tasks for netlists. For functional tasks, the fine-tuning model predicts early RTL-stage information on netlists, such as gate functions and register types. As for physical tasks, it predicts late layout-stage metrics like register endpoint slack, power, and area. We will provide the detailed evaluation results in Section III-B.

III. Experimental Results

A. Experimental Setup

Data preparation. We collect circuits for pre-training from various sources, including benchmark RTL code from ITC99 [41], OpenCores [42], Chipyard [43], and VexRiscv [44]. All RTL designs are synthesized into netlists using Synopsys Design Compiler using NanGate 45nm technology library and then undergo physical design with Cadence Innovus. Gate and design quality metrics and statistics are obtained using Synopsys PrimeTime. For symbolic logic expressions, we use PySMT [37], a symbolic reasoning toolkit for formal verification, to construct and manipulate the Boolean expressions.

The statistical details of our dataset are shown in TABLE II. For the expression dataset used by ExprLLM, we collected 313k original expressions, each augmented with functionally equivalent transformations, resulting in a total of 626k expressions. As for the circuit netlist data, we collected 100k subcircuit cones after chunking, which were also functionally augmented to reach 200k samples in total. Additionally, we included 10k aligned RTL and layout cones for cross-stage alignment. When applying NetTAG to downstream tasks, we directly use the original task datasets if available. Otherwise, we utilize designs from the aforementioned open-source benchmarks.

TABLE II: Statistics of circuit expression and netlist dataset.

Source	Gate Expression		Circuit Netlist	
	# Data	# Tokens (Avg.)	# Data	# Nodes (Avg.)
ITC99	47k	6,960	4k	1,025
OpenCores	76k	212	55k	173
Chipyard	109k	9,849	20k	2,813
VexRiscv	81k	5,289	21k	901
Total	**313k**	**5,810**	**100k**	**855**

Implementation details. For the implementation of NetTAG, we initialize ExprLLM with LLM2Vec [38], an open-source LLM-based encoder based on Meta Llama-3.1-8B [45], with 8k maximum input tokens. For the graph transformer backbone of TAGFormer, we adopt SGFormer [46]. Additionally, we use NV-Embed [47] (maximum 32k input tokens) as the pre-trained RTL text encoder, and the layout encoder is pre-trained using a graph contrastive objective on SGFormer. Each MLP contains three layers with a hidden dimension of 256, and the output dimension of NetTAG is set to 768. We pre-train ExprLLM using LoRA on the full dataset for one epoch, followed by 50 epochs

of pre-training for TAGFormer. Afterward, fine-tuning is performed with lightweight models and task-specific labels. All experiments are conducted on 8 Nvidia 4090 GPUs and 4 Nvidia A6000 GPUs.

B. Performance on Various Downstream Tasks

We evaluate the pre-trained NetTAG on four representative netlist-stage tasks, including functional and physical tasks across combinational and sequential circuits. These tasks span multiple netlist granularities: combinational gates, register cones, and entire circuits, offering a thorough assessment of NetTAG's capabilities.

NetTAG consistently outperforms task-specific baselines in each task, demonstrating its ability to capture functional and physical circuit information. This highlights NetTAG's role as a foundation model that is capable of generating informative embeddings for netlists. Detailed evaluations for each task are as follows:

Task1: Combinational gate function identification. This task identifies the functional type of each netlist combinational gate (e.g., adder, multiplier, comparator) as described in the original RTL code. Accurate gate function identification is critical for applications such as reverse engineering, hardware security, and functional verification. We evaluate NetTAG on an open-source dataset from GNN-RE [14], ensuring no label-related text is included in the gate text attributes and using the same metrics for a fair comparison.

As shown in TABLE III, NetTAG significantly outperforms GNN-RE across all metrics, with an average improvement of 14% in accuracy (97% vs. 83%), 11% in precision, 14% in recall and F1-score. These improvements demonstrate NetTAG's robust ability to capture functional information on each gate, enabling superior generalization compared to the task-specific GNN-based method.

TABLE III: Performance comparison on Task1: combinational gate function identification.

Design	GNN-RE [14]				NetTAG			
	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)
1	79	82	79	74	97	97	97	97
2	96	96	96	96	100	100	100	100
3	94	94	94	94	100	100	100	100
4	78	83	78	78	100	100	100	100
5	91	92	91	90	99	99	99	99
6	74	78	74	68	94	94	94	93
7	80	80	80	80	84	87	84	81
8	89	90	89	87	95	96	95	96
9	65	77	65	67	100	100	100	100
Avg.	83	86	83	82	**97**	**97**	**97**	**96**

Task2: Sequential state/data register identification. This task distinguishes state registers from data path registers, which is essential for understanding high-level control and data flow within low-level netlists, supporting security analysis and verification. We evaluate NetTAG against ReIGNN [15], a supervised method specifically designed for this task based on GNN. The evaluation metrics include sensitivity and balanced accuracy (the average of sensitivity and true negative rate).

As demonstrated in TABLE IV, NetTAG significantly outperforms ReIGNN in both metrics. It achieves a 44% improvement in sensitivity (90% vs. 46%) for state register identification and a 13% increase in overall accuracy (86% vs. 73%), highlighting its superior capability of capturing structural and functional information from register cones.

Task3: Endpoint register slack prediction. This task focuses on predicting sign-off timing slack at the netlist stage to provide early feedback, thereby expanding the optimization space in the physical design flow. The prediction is highly challenging due to substantial graph topology changes during physical design optimizations, as discussed in [2]. We compare NetTAG with the state-of-the-art GNN-based method from [2], extending their post-placement method to work at the post-synthesis stage. Performance is evaluated using correlation coefficient (R) and mean absolute percentage error (MAPE).

TABLE IV: Performance comparison on Task2: state/data register identification & Task3: endpoint register slack prediction.

Design	Task 2				Task 3			
	REIGNN [15]		NetTAG		GNN*		NetTAG	
	Sens. (%)	Acc. (%)	Sens. (%)	Acc. (%)	R	MAPE (%)	R	MAPE (%)
itc1	50	72	100	98	0.89	13	0.94	9
itc2	100	92	100	100	0.91	10	0.93	9
chipyard1	30	65	80	79	0.72	17	0.7	20
chipyard2	30	65	90	86	0.86	12	0.95	9
vex1	50	74	82	74	0.94	11	0.93	12
vex2	32	60	86	82	0.97	18	0.97	9
opencores1	42	73	93	84	0.99	26	0.92	29
opencores2	37	80	92	82	0.93	30	0.98	26
Avg.	46	73	**90**	**86**	0.9	17	**0.92**	**15**

* We adapt the GNN model from [2] for netlist-stage slack prediction, as it was originally designed for the layout stage.

As shown in TABLE IV, NetTAG outperforms the customized GNN baseline [2], achieving more accurate slack prediction with R of 0.92 vs. 0.9 and a MAPE of 15% vs. 17%.

Task4: Overall circuit power/area prediction. This task predicts final layout power and area metrics at the netlist stage, offering early estimates for physical design. We compare NetTAG with the synthesis tool and a customized GNN model adopted from PowPrediCT [7], a layout-stage state-of-the-art optimization-aware power predictor.

For area prediction, NetTAG achieves a 4% MAPE without optimization and 11% with optimization, outperforming the GNN's 5% and 18% and the synthesis EDA tool's 5% and 34%. For power, it yields an 8% MAPE without optimization and 12% with optimization, surpassing the GNN's 12% and 19% and the EDA tool's 34% and 38%. These results highlight NetTAG's strong accuracy and robustness for circuit-level tasks.

TABLE V: Performance comparison on Task4: overall circuit power/area prediction.

Target	Metric†	EDA Tool		GNN*		NetTAG	
		R	MAPE (%)	R	MAPE (%)	R	MAPE (%)
Area	w/o opt	0.99	5	0.99	5	**0.99**	**4**
	w/ opt	0.95	34	0.95	18	**0.96**	**11**
Power	w/o opt	0.99	34	0.99	12	**0.99**	**8**
	w/ opt	0.73	38	0.76	19	**0.86**	**12**

† Post-layout labels are collected in two scenarios: without considering optimization (denoted as "w/o opt") and with optimization as illustrated in [7] (denoted as "w/ opt").
* We adapt the GNN model from [7] for netlist power and area prediction, as it was originally designed for the layout-stage power prediction.

C. Comparision with pre-trained netlist encoders

In addition to the task-specific methods, we also compare NetTAG with pre-trained netlist encoders. Since they are all limited to combinational AIG circuits, we evaluate them only on Task 1, using an AIG-format dataset. As shown in Fig. 5, NetTAG outperforms SOTA AIG-based encoders, including FGNN [29] and DeepGate3 [27], achieving the highest average performance across all metrics. Notably, NetTAG achieves superior accuracy on the AIG dataset compared with more diverse gate datasets in TABLE III for Task 1, underscoring its adaptability. We also evaluate the standalone ExprLLM component of NetTAG, which performs well using symbolic expressions, highlighting the power of gate semantic understanding. These results emphasize NetTAG's advantage in combining semantic and structural information through the TAG format.

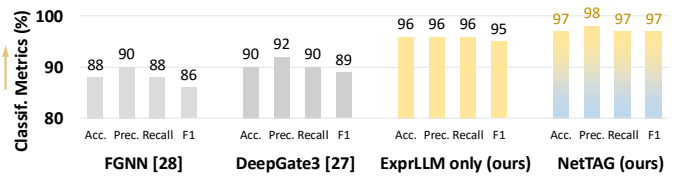


Fig. 5: Comparision with pre-trained AIG encoders on AIG dataset.

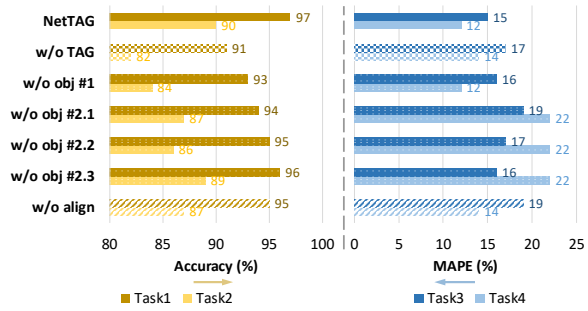


Fig. 6: Ablation study.

D. Ablation Study

This ablation study evaluates the contributions of three key aspects of NetTAG: TAG-based representation, self-supervised pre-training objectives, and cross-stage alignment. The analysis, shown in Fig. 6, highlights the impact of each component on the model's performance across the four tasks, with detailed analysis as follows:

Effectiveness of learning netlist via TAG. Removing text attributes and relying only on graph structure significantly reduces performance, especially on functional tasks. This indicates that semantic information from text attributes is crucial. Additionally, physical tasks also show a slight decline, suggesting that even structure-focused tasks benefit from the added semantic context in TAG.

Effectiveness of self-supervised pre-training objectives. The results demonstrate that the expression contrastive learning objective (#1) for ExprLLM has the greatest impact on functional tasks, suggesting it enhances the LLM's understanding of symbolic logic expressions. Objectives #2.1 (masked gate reconstruction) and #2.2 (graph contrastive learning) improve performance across both functional and physical tasks, highlighting their role in capturing both local and global netlist structure. Objective #2.3 (graph size prediction) shows the strongest effect on physical tasks, where a comprehensive understanding of the netlist's overall structure is essential.

Effectiveness of cross-stage alignment. Removing cross-stage alignment leads to a notable drop in performance across all four tasks. This alignment effectively enhances NetTAG's ability to integrate both early-stage functional and late-stage physical information, which is crucial for downstream tasks.

E. Runtime Analysis

In TABLE VI, we present NetTAG's runtime across various benchmarks. Most runtime is spent on preprocessing (i.e., converting netlists into TAG format) and node-level inference using ExprLLM. Despite these steps, NetTAG demonstrates around a 10x speedup over traditional physical design workflows using commercial EDA tools. There is significant potential for further runtime improvements. For example, symbolic expression extraction for each gate can be highly parallelized, reducing preprocessing time. Additionally, the ExprLLM inference time could be largely reduced by scaling up GPU resources, as this runtime depends on GPU performance and quantity.

IV. Discussion

A. Scalability: Performance Scaling with Model and Data Size

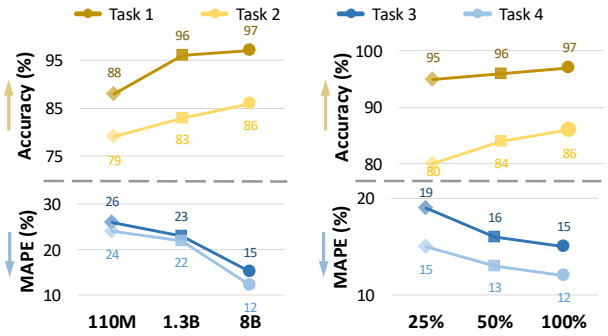
In Fig. 7, we study how the downstream task performance of NetTAG scales with both model size and pre-training data size.

TABLE VI: Runtime (minutes) Comparision.

Source	EDA Tool	Ours (Avg.)			
	P&R (Avg.)	Pre*	Infer		Total
			ExprLLM	TAGFormer	
ITC99	164	2	5	0	7
OpenCores	288	18	12	1	31
Chipyard	251	15	10	1	26
VexRiscv	207	8	5	2	15
GNNRE	/	4	2	0	6

* Preprocessing (chunking into cones and converting netlist into TAG).

The plot shows the performance of each task after fine-tuning. Scaling ExprLLM backbone from 110M parameters (BERT) to larger models like Meta Llama 3.1 with 1.3B and 8B parameters results in significant performance improvements across all four tasks. Similarly, expanding data size from 25% to 100% of the dataset consistently enhances performance. These results demonstrate the scalability of NetTAG, suggesting that further increases in model and data size could lead to even greater improvements in accuracy and generalization.



(a) Increasing model size (b) Increasing data size

Fig. 7: Performance scaling with model and data size.

B. Demo: Reasoning Netlist's Arithmetic Function with NetTAG

Fig. 8 shows how NetTAG improves functional reasoning for arithmetic netlists. Using an LLM (e.g., OpenAI's o1-preview) to analyze post-synthesis netlist Verilog texts, we assess its ability to interpret the arithmetic function from the flattened netlists. Without NetTAG's gate-level function identification, the LLM struggles with interpreting functionality from flattened netlists. When integrated with NetTAG, the LLM accurately infers that the netlist module compares two 2-bit values, performs addition and multiplication, and selects the result based on the comparison outcome.

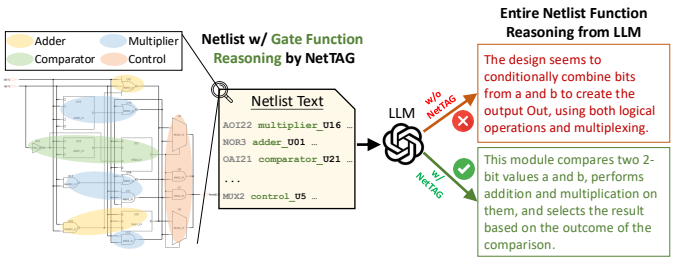


Fig. 8: A demo for reasoning arithmetic function from netlists.

V. Conclusion And Future Work

In this paper, we present NetTAG, a foundation model for netlist representation learning that accommodates diverse gate types and supports a range of functional and physical tasks. By formulating netlists as text-attributed graphs, NetTAG uniquely integrates gate semantics with graph structure through its multimodal architecture and self-supervised pre-training objectives. Future work will enhance NetTAG by extending pre-training to larger circuit datasets and exploring decoding methods to broaden task support capabilities.

VI. Acknowledgement

This work is supported by Hong Kong Research Grants Council (RGC) CRF Grant C6003-24Y, National Natural Science Foundation of China (NSFC) 62304192, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and National Supercomputing Center in Guangzhou Nansha Sub-center for computational resources.

REFERENCES

[1] W. Fang, S. Liu, H. Zhang, and Z. Xie, "Annotating slack directly on your verilog: Fine-grained rtl timing evaluation for early optimization," in *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024, pp. 1–6.

[2] Z. Wang, S. Liu *et al.*, "Restructure-tolerant timing prediction via multimodal fusion," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[3] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1207–1212.

[4] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[5] C. Xu, P. Sharma, T. Wang, and L. W. Wills, "Fast, robust and transferable prediction for hardware logic synthesis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 167–179.

[6] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, "How good is your verilog rtl code? a quick answer from machine learning," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2022, pp. 1–9.

[7] Y. Du, Z. Guo, X. Jiang, Z. Chai, Y. Zhao, Y. Lin, R. Wang, and R. Huang, "Powpredict: Cross-stage power prediction with circuit-transformation-aware learning," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024, pp. 1–6.

[8] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *Proceedings of 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[9] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "PRIMAL: Power inference using machine learning," in *Proceedings of 56th Annual Design Automation Conference (DAC)*, 2019.

[10] Z. Xie, Y.-H. Huang *et al.*, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[11] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proceedings of 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1821–1824.

[12] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. Wong, "Lay-net: Grafting netlist knowledge on layout-based congestion prediction," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[13] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie, "Gamora: Graph learning based symbolic reasoning for large-scale boolean networks," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[14] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 8, pp. 2435–2448, 2021.

[15] S. D. Chowdhury, K. Yang, and P. Nuzzo, "Reignn: State register identification using graph neural networks for circuit reverse engineering," in *Proceedings of IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[16] Z. He, Z. Wang *et al.*, "Graph learning-based arithmetic block identification," in *Proceedings of IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[17] L. Chen, Y. Chen *et al.*, "Large circuit models: opportunities and challenges," *Science China Information Sciences*, 2024.

[18] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, "A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda," *arXiv preprint arXiv:2504.03711*, 2025.

[19] W. Fang, S. Liu, J. Wang, and Z. Xie, "Circuitfusion: multimodal circuit representation learning for agile chip design," in *International Conference on Learning Representations (ICLR)*, 2025.

[20] K. Zhu, H. Chen, W. J. Turner, G. F. Kokai, P.-H. Wei, D. Z. Pan, and H. Ren, "Tag: Learning circuit spatial embedding from layouts," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2022, pp. 1–9.

[21] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Robust gnn-based representation learning for hls," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023.

[22] Z. Luo, T. S. Hy *et al.*, "De-hnn: An effective neural model for circuit netlist representation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024.

[23] C. Deng, Z. Yue *et al.*, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024.

[24] S. Yang, Z. Yang, D. Li, Y. Zhang, Z. Zhang, G. Song, and J. Hao, "Versatile multi-stage graph neural network for circuit representation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 20 313–20 324, 2022.

[25] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 667–672.

[26] Z. Shi, H. Pan *et al.*, "Deepgate2: Functionality-aware circuit representation learning," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[27] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: towards scalable circuit representation learning," *arXiv preprint arXiv:2407.11095*, 2024.

[28] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 61–66.

[29] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, Y. Huang, and B. Yu, "Fgnn2: A powerful pre-training framework for learning the logic functionality of circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[30] S. Khan, Z. Shi, M. Li, and Q. Xu, "Deepseq: Deep sequential circuit learning," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–2.

[31] W. Fang, S. Liu, H. Zhang, and Z. Xie, "A self-supervised, pre-trained, and cross-stage-aligned circuit encoder provides a foundation for various design tasks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025.

[32] Z. Pei, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[33] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *arXiv preprint arXiv:2402.00386*, 2024.

[34] H. Wu, Z. He *et al.*, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[35] X. He, X. Bresson, T. Laurent, A. Perold, Y. LeCun, and B. Hooi, "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[36] H. Liu, J. Feng *et al.*, "One for all: Towards training one graph model for all classification tasks," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[37] M. Gario and A. Micheli, "Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms," in *SMT workshop*, 2015.

[38] P. BehnamGhader, V. Adlakha, M. Mosbach, D. Bahdanau, N. Chapados, and S. Reddy, "Llm2vec: Large language models are secretly powerful text encoders," *arXiv preprint arXiv:2404.05961*, 2024.

[39] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "Mteb: Massive text embedding benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 2023, pp. 2014–2037.

[40] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[41] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers (ITC)*, 2000.

[42] *OpenCores: The reference community for Free and Open Source gateware IP cores*, https://opencores.org/.

[43] A. Amid, D. Biancolin *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, 2020.

[44] VexRiscv, "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," 2022. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[45] A. Dubey, A. Jauhri *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[46] Q. Wu, W. Zhao *et al.*, "Simplifying and empowering transformers for large-graph representations," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2024.

[47] C. Lee, R. Roy *et al.*, "Nv-embed: Improved techniques for training llms as generalist embedding models," *arXiv preprint arXiv:2405.17428*, 2024.

Can Short Hypervectors Drive Feature-Rich GNNs? Strengthening the Graph Representation of Hyperdimensional Computing for Memory-efficient GNNs

Jihe Wang
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
wangjihe@nwpu.edu.cn

Yuxi Han
Northwestern Polytechnical University
Engineering Research Center of Embedded System Integration, Ministry of Education
Xi'an, China
hyx956@mail.nwpu.edu.cn

Danghui Wang*
Northwestern Polytechnical University
Engineering Research Center of Embedded System Integration, Ministry of Education
Xi'an, China
wangdh@nwpu.edu.cn

***Abstract*—Hyperdimensional computing (HDC) based GNNs are significantly advancing the brain-like cognition in terms of mathematical rigorousness and computational tractability. However, the researches in this field seem to have a "*long vector consensus*" that the length of HDC-hypervectors must be designed to mimic that of cerebellar cortex, i.e., ten thousands of bits, to express human's feature-rich memory. To system architects, this choice presents a formidable challenge that the combination of numerous nodes and ultra-long hypervectors could create a new memory bottleneck that undermines the operational brevity from HDC. To overcome above problem, in this work, we shift our focus to rebuilding a set of more GNN-friendly HDC-operations, by which, short hypervectors are sufficient to encode rich features via enjoining the strong error tolerance of neural cognition. To achieve that, three behavioral incompatibilities of HDC with general GNNs, i.e., *feature distortion*, *structural bias*, and *central-node vacancy*, are found and successfully resolved for more efficient feature-extraction in graphs. Taken as a whole, a memory-efficient HDC-based GNN framework, called CiliaGraph, is designed to drive one-shot graph classifying tasks with only hundreds of bits in hypervector aggregation, which offers 1 to 2 orders of memory savings. The results show that, compared to the SOTA GNNs, CiliaGraph reduces the memory access and training latency by an average of 292× (up to 2341×) and 103× (up to 313×), respectively, while maintaining the competitively accuracy.**

***Index Terms*—GNN, hyperdimensional computing, memory, graph representation.**

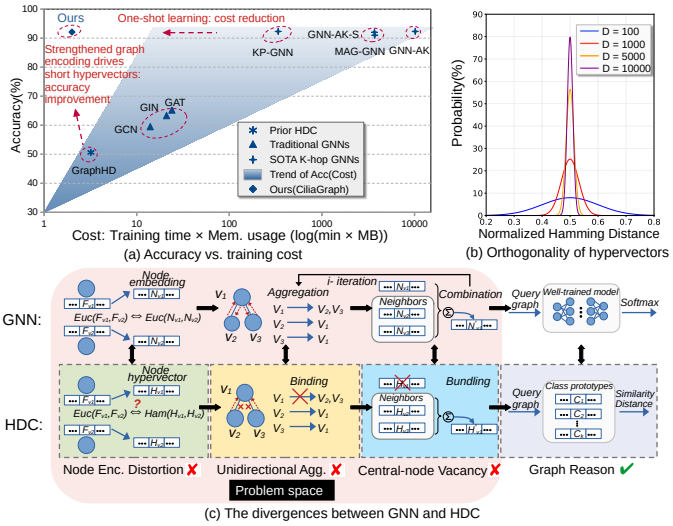


Fig. 1: Problem space of SOTA HDC and aims of our work.

I. Introduction

Hyperdimensional computing (HDC) utilizes long hypervectors to mimic the behaviors of the cerebellar cortex in brains, which successfully provides various desirable properties that other machine learning methods lack, such as simple operations, one-shot learning, and robustness to noise [1]. Accordingly, the brain-inspired HDC is introduced in traditional graph neural networks (GNNs) [2]–[5] to fully function its mathematical rigorousness [6] and computational tractability [7]. These HDC-based GNNs no longer require interminable training and expensive matrix operations, yet still achieve competitive model accuracy, as a result, being increasingly popular in graph cognitive tasks on edge devices [7]. Although the ultra-long hypervectors, e.g., tens of thousands bits, faithfully replicate the natural configuration in human brain, they pose a new challenge for system architects in terms of memory bottleneck, i.e., numerous nodes represented by long hypervector for each. Therefore, whether it is possible to encode rich features using shorter hypervectors remains an open question (Fig. 1(a)).

Hypervectors for rich features. Unlike NN-based feature encoding in general GNNs [8], HDC-based approaches commonly employ superposition of orthogonal hypervectors to fuse multiple features [9]. To forcibly convert numerical node features into hypervectors, two typical methods are widely adopted in their superposition: 1) ***random projecting*** [4], [5] and 2) ***leveled mapping*** [9]. The former directly scales the random hypervectors proportionally to their corresponding feature values and then sums them up to create a fused hypervector as the representation of multiple features, which always invokes a *majority function* [7] in final to revert the vectors to their binary forms, i.e., $(\{\mathbb{R}\}^D \rightarrow \{-1, 1\}^D$. The latter stochastically encodes feature values into pre-defined hypervectors as the weight to process "*bind-and-bundle*" operations (a.k.a. the "multiply-and-add" in HDC version) on corresponding orthogonal hypervectors, which removes the expensive *majority function*, meanwhile, retains the simple operation of binary. A common drawback of above methods is that features with clearly numerical implication become less distinguishable in hyper-dimensional spaces, particularly when a node has rich features, e.g., COIL-RAG [10] dataset even with 64 features. To compensate for the loss of feature representation, designers have been compelled to use a system-unfriendly choice that using a large amount of memory to hold much longer vectors [1], [4], [11], [12] for higher resolution of numerical features.

A key debate: Some fundamentalists of brain-inspire computing insisted on the necessity of ultra-long hypervectors for HDC-based GNNs. Their arguments come from a mathematical criterion that, in theory, hypervectors shorter than $10,000$-bits cannot ensure the high-quality of orthogonality (Fig. 1(b)), which has even been seen as the "*golden rule*" of any correct HDC [6], [11]. However, to system

*Corresponding author

architects, it's too expensive to use $10,000$-bits vectors to encode only one of node features. For example, to store one node with three features quantized to 8-bit FxP, its numerical embedding only needs 128 bytes in general GNNs [8], but unfortunately, the equivalent hypervector requires an unacceptable 29.3KB, which is almost out of reach at system level. Therefore, redundant hypervectors lead to severe inefficiency in memory, which casts doubt on the continued adherence of the "*golden rule*".

In this work, we find that the core difficulty of HDC-based GNNs is its inability to faithfully execute each steps involved in a standard GNN process [13], leading to three kinds of operational noises in below that significantly weaken the extraction of multiple features (Fig. 1(c)). ***Noise-1***: In feature encoding, by replacing the Euclidean distance with the simple Hamming one between hypervectors, both *random projecting* and *leveled mapping* introduce a distortion in the coding space, failing to accurately measure the relationships as the original numerical feature spaces. ***Noise-2***: The originally bidirectional and symmetric *aggregation* in GNNs has been simplified to a unidirectional tree-like *binding* in the existing HDC [3], [7], causing a structural bias of their graphs. ***Noise-3***: During *combination* stage, standard GNNs always fuse neighboring nodes with the central one (h^i) to generate a new embedding (h^{i+1}), however, existing HDCs, without exception, ignore central nodes via a socalled "equivalent" *bundling* [1], [4] which produces a *central-node vacancy* in the process of embedding fusion. As seen in Sec. II, these three divergences jointly result in the suppressed expressability of hypervectors, inevitably making low efficiency in memory usage.

To overcome the problems, we design a hypervector space that preserves the original numerical distribution of node features, meanwhile, enjoys the encoding efficiency of HDC. Based on the space, a fully symmetric on-edge *binding* is proposed to adhere the embedding aggregation in standard GNNs without the structural bias of graphs. To repair the vacated central nodes, a new concatenation operator is propose to merge periphery and central hypervectors before the final reason produced. Our proposed GNN framework, called CiliaGraph, demonstrates the feasibility of the aforementioned operations in one-shot graph classifying tasks using only 200-bit hypervectors. The results show that, compared with SOTA GNNs, CiliaGraph reduces memory usage and training time by 292× and 103× (up to 2341× and 313×) respectively. Our contributions include:

1. We identify the three sources of weak expression by hypervectors in HDC-based GNNs as the system-unfriendly compromise, i.e., long hypervectors, in the past.
2. CiliaGraph, including a set of new HDC operators, is proposed to mimics the information merging in GNNs, which demonstrates the feasibility of ultra short hypervectors in GNNs.
3. The results show the applicable of CiliaGraph in one-shot graph classifying tasks on widespread datasets and the great memory saving makes it well-suited for on-edge HDC-based GNNs.

II. Background and Motivation

A. HDC and the operators in GNNs

Traditional HDC models employ D-dimensional binary hypervectors as the fundamental elements for computation in $\mathcal{HD}$ space. Let's assume $\mathcal{H}_1, \mathcal{H}_2 \in \{-1, 1\}^D$ are two randomly generated hypervectors and their similarity $\delta(\mathcal{H}_1, \mathcal{H}_2) \approx 0$. ***Binding*** of $\mathcal{H}_1$ and $\mathcal{H}_2$, i.e., $\mathcal{H}_1 \otimes \mathcal{H}_2$, is component-wise multiplication (XOR in binary) and generates a new hypervector that is dissimilar to its two constituents, i.e., $\delta(\mathcal{H}_1 \otimes \mathcal{H}_2, \mathcal{H}_1) \approx 0$, which is used to find the association of two hypervectors [4]. ***Bundling*** is component-wise addition of two hypervectors, i.e., $\mathcal{H}_1 \oplus \mathcal{H}_2$, that memorizes its all

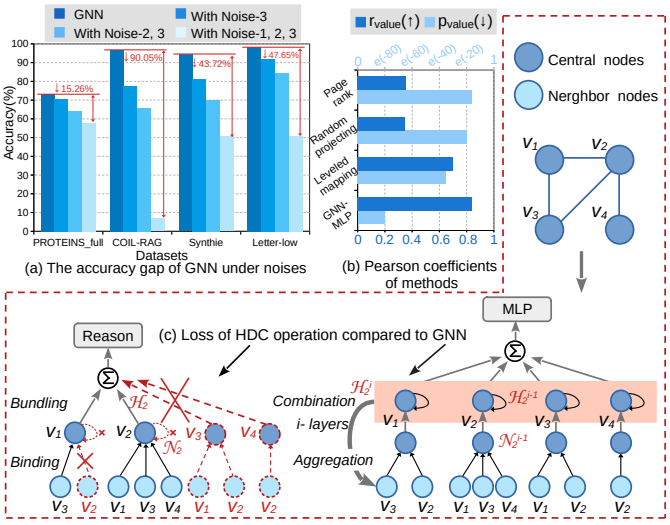


Fig. 2: A motivational analysis of accuracy loss under three noises introduced by HDC operators.

components, therefore, the result is similar to its every component, i.e., $\delta(\mathcal{H}_1 \oplus \mathcal{H}_2, \mathcal{H}_1) >> 0$. ***Reasoning*** is to measure the similarity of two hypervectors and generally utilizes Hamming distance or dot product [11]. The above three HDC primitives are adopted widely in the most HDC-based GNNs.

For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, current HDC-based GNNs aim to mimic the feature extraction of traditional GNNs (Fig. 1(c)), where, 1) the node embeddings are recoded with ultra-long hypervectors, 2) the aggregation can be alternated as their binding, 3) the combination is just the bundling of all neighboring hypervectors, and 4) the final MLP is done efficiently with HDC reasoning. However, these purportedly "equivalent" operations exhibit unmatched capabilities in feature extraction on graphs. The ablation study (Fig. 2(a)) clearly demonstrates a stepwise decline, or noise, in the model inference accuracy as we progressively replace GNN operators with their HDC counterparts, i.e., combination→bundling (10.27% ↓), aggregation→binding (9.34% ↓), and embedding→hypervector (29.57% ↓) respectively. Resultantly, the SOTA HDC-based GNNs are generally 49.17% less accurate than traditional GNNs [3], [12], which has driven designers to the short-sighted solution of employing longer hypervectors, akin to a poisoned chalice. In next, the three deeper causes of these declines are analyzed.

B. Hypervector: Distortion of Distance in Feature Encoding

Effective numerical encoding using HDC has always been a challenging problem, primarily due to the inherent tension of ensuring both strict *orthogonality* and correct *numerical distances* between two hypervectors [6]. In practice, more efforts focus on maintaining the orthogonality, rather than the distances [1], [7], because they speculated that hypervectors, a.k.a. discrete random bit sequences, could represent the numerical disparities of features well as long as they are long enough. Unfortunately, our a comparative study on distance-isomorphism reveals a widespread severe distortion the hypervector spaces (Fig. 2(b)). This study summons Pearson's coefficient [14] to quantify **1)** r – the correlation between "the hypervector distances in $\mathcal{HD}$ spaces" and "the embedding distances of numerical features," and **2)** p – the error of r. The results shows that PageRank, Random-Projecting, and Leveled-Mapping, as the three most popular encoders in HDC ($r \downarrow, p \uparrow$), seriously deviate the expected GNN features ($r \uparrow, p \downarrow$), which confirms our doubt that long hypervectors are disable to encode the numerical features in GNNs. In this work, a plastic random bit-flip (PRBF) encoding is proposed

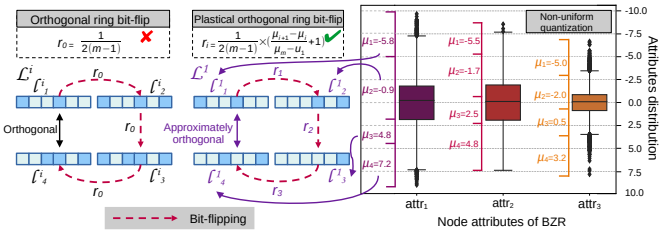


Fig. 3: Improved plastic orthogonal ring introduce a non-fixed flip rate compared to orthogonal ring.

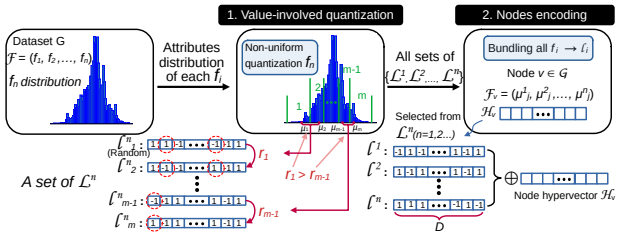


Fig. 4: The steps of plastic random bit-flip encoding

to fine-tune the random flips according to both feature values and their distributions, effectively overlaying numerical distance onto orthogonal hypervectors (Sec. III-A).

C. Binding: Structural Bias by Unidirectional Tree-like Aggregation

Standard GNNs are always capable of accurately learning graph structures, primarily by the bidirectional aggregation through edges, i.e., connectivity, which drives aggregation strictly adhering to the data-flow defined by a graph structure. In contrast, existing HDC-based one-shot solutions [3], which employ unidirectional binding to rapidly reduce long hypervectors to relieve memory pressure, simplify all complex graph structures into a tree that biases original graph structures. As shown in Fig. 2(c), while one edge in the original graph (e.g. $e_{1,2}$) triggers bidirectional binding in GNNs ($v_2 \rightarrow v_1$ for v_1 and $v_1 \rightarrow v_2$ for v_2), in HDC however, only the more important nodes (such as v_2) are responsible for collecting the neighbor features ($v_1 \rightarrow v_2$) and the reversed binding ($v_2 \rightarrow v_1$) are totally neglected. Reducing the symmetric GNN aggregations to half asymmetric ones limits the global structural information captured during the binding of HDC (Fig. 2(c)). In this work, a precise aggregation, i.e., *differential aggregation*, is proposed (Sec. III-B), which not only implements bidirectional symmetric aggregation as in GNNs but also leverages on-edge weights to modulate the strength of peripheral hypervectors, thereby achieving the full function of aggregating capabilities of GNNs.

D. Bundling: Vacancy of Central Nodes during Combination

The last drawback of HDC-based GNN lies in its perfunctory way in bundling, which abruptly overwhelms [4], [5], or even removes [3], the central nodes during the combination, resulting in a significant departure from the standard procedure of GNNs. As shown in Fig. 2(c), during the standard combination of GNNs, a node (e.g., v_2) stores a local embedding ($\mathcal{H}_2^{i-1}$) to memorize the embeddings of historical levels. By combining local embeddings with the aggregated peripheral ones of current layer, GNNs facilitate an embedding update of central node, e.g., $COMB(\mathcal{H}_2^{i-1}, \mathcal{N}_2^{i-1}) \rightarrow \mathcal{H}_2^i$, by which, peripheral features ($\mathcal{N}_2^{i-1}$) can incrementally contribute to a central node until all node features are well extracted. In opposite, as a one-pass algorithm, HDC inherently doesn't require storing the history of hypervectors, therefore, to save memory space, existing methods either exclude the features of central nodes from combination, e.g., $BUNDLE(\mathcal{N}_2) \rightarrow \mathcal{H}_2$, or degrade them equally with that of neighbors, e.g., $BUNDLE(\mathcal{N}_2 \oplus H_2) \rightarrow \mathcal{H}_2$. In this work, we propose a simple concatenation method, where, a central node and its peripheral nodes, as a whole, individually occupy half of the hypervector length to highlight central features like standard GNNs.

III. Strengthening the Representation of Hypervectors

A. Plastic Random Bit-Flip Encoding

Value-involved quantization for hypervectors. To overcome the distortion in hypervector encoding, inspired by a previous works [15], we propose an encoding method that utilizes the values of features to proportionally and randomly flip the bits to generate new hypervectors. Initially, the n features of a node constitute n independent hypervector spaces. For the i-th space $\mathcal{L}^i$, all possible values it can sample form a quantization space, which is defined by upper and lower bounds and a sampling distribution. Thus, the range between the upper and lower bounds can be divided into m quantization centers $(\mu_1^i, \mu_2^i, \ldots, \mu_m^i), \mu_x^i < \mu_y^i$ *when* $x < y$, called *clusters*, with each cluster mapped to a representative hypervector l_j^i. Consequently, $\mathcal{L}^i$ can be represented by m ordered hypervectors $\mathcal{L}^i = (l_1^i, l_2^i, \cdots, l_m^i)$ to encode the m quantization centers, that is, when the value of the i-th feature f_i falls into the j-th cluster μ_j^i, f_i is directly encoded as l_j^i. Under this method, the only remaining challenge is how to initialize an ordered set of hypervectors l_j^i to faithfully capture the distribution of feature values.

Plastical orthogonal ring. In order to achieve a quantization hypervector (l_j^i) that's equivalent to numerical space (μ_j^i), a orthogonal ring method is proposed in [7]. It constructs multi-level hypervectors $(l_2^i \cdots, l_{m-1}^i)$ with gradually and randomly flipping between two orthogonal vectors (l_1^i, l_m^i), ensuring the similarity between adjacent vectors is a fixed quantization step (Fig. 3), i.e., $\delta(l_j^i, l_{j+1}^i) = \frac{1}{2(m-1)}$ and $\delta(l_1^i, l_m^i) = 0$ (Note: the zero δ hints that there are half different bits between l_1^i and l_m^i). However, this method still introduces significant noise, which is primarily due to the poor accuracy in those dense regions, where finer granularity must be very useful in quantization. Therefore, an improved *plastical orthogonal ring* is built to allocate different bit-flipping rates (r_i) according to the numerical distance between two plastical clusters, rather than a fixed step, meanwhile ensuring $\delta(l_1^i, l_m^i) \approx 0$ (i.e., *approximately orthogonal*). By introducing a plastic factor $\frac{\mu_{i+1}-\mu_i}{\mu_m-\mu_1}$, each quantization step becomes flexible as:

$$r_i = \frac{1}{2(m-1)} \times \left(\frac{\mu_{i+1} - \mu_i}{\mu_m - \mu_1} + 1 \right) \quad \text{for} \quad 1 \leq i \leq m-1 \qquad (1)$$

where, the former term is the fixed step that divides half of a hypervector ($\frac{1}{2}$) into $m-1$ bit-flipping steps and the later term uses the plastic factor to decide the adaptation to each step (Fig. 3). This adaptation can provide more scalability for those dense regions, at the same time, guarantee the orthogonal ring like [7].

Non-uniform quantization with Hypervectors. Based on the plastical orthogonal ring, the non-uniform quantization in numeral space can be isomorphically mapped into the non-equidistant hypervector space. For example, in Fig. 4, the dense region is quantized as narrower clusters, dropped int which, the corresponding hypervectors also have larger similarity since they have undergone fewer bit flips during initialization. After obtaining the orthogonal hypervectors of all features ($f_i \rightarrow l^i, 1 \leq i \leq n$), those hypervectors are further bundled to generate the representation of a node, i.e., $\mathcal{H}_v = \oplus_{i=1}^n l^i, v \in \mathcal{V}$.

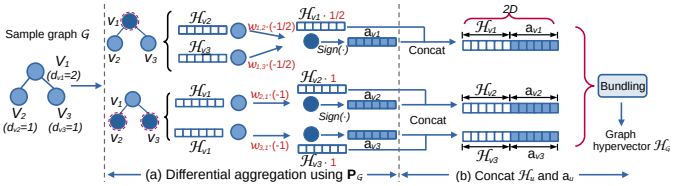


Fig. 5: Aggregation using hyper-weight matrix and equal concatenation.

B. Differential Aggregation for Structural Enhancement

Based on the discussion in Sec. II-C, to be equivalent to GNNs, an HDC-based aggregation should have the ability to 1) estimate the structures around a central node, 2) assess the strength of connections symmetrically between nodes, and 3) inject peripheral nodes into the central node as the aggregation (a_u). Recognizing the disability of the simple binding operation, we propose a novel differential scheme to take charge of the hypervector aggregation. Above all, in a graph, the feature distance between any two nodes can be represented by their similarity of hypervectors, i.e., $w_{u,v} = \delta(\mathcal{H}_u, \mathcal{H}_v), u, v \in \mathcal{V}$, which forms a weight matrix $\mathbf{W}_\mathcal{G} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ among nodes to involve all connecting strength. Then, the connectivity $t_{u,v}$ between a central node u and all other nodes, e.g., v, is defined as: 1) $t_{u,v} = \frac{1}{d_u}$ if $u = v$, 2) $t_{u,v} = -\frac{1}{d_u}$ if $e_{u,v} \in \mathcal{E}$, and 3) otherwise $t_{u,v} = 0$, where d_u is the degree of node u. Therefore, all $t_{u,v}$ constitute another transition matrix $\mathbf{T}_\mathcal{G} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ that asserts the peripheral structure of each node.

By a Hadamard product ($\circ$) of above two matrices, both of feature similarity and connectivity are merged into a sole ***hyper-weight matrix***, i.e., $\mathbf{P}_\mathcal{G} = \mathbf{W}_\mathcal{G} \circ \mathbf{T}_\mathcal{G}$, where $p_{u,v} = w_{u,v} \times t_{u,v}$. In short, the $\mathbf{P}_\mathcal{G}$ has three explainable properties in below. ***First***, $\mathbf{T}_\mathcal{G}$ structurally filter the similarity values of all elements in $\mathbf{W}_\mathcal{G}$, by which, the resultant matrix $\mathbf{P}_\mathcal{G}$ yields perfect alignment with graph structure. ***Second***, as the basis of u-centered aggregation, the u-th diagonal element $p_{u,u} = \frac{1}{d_u}$ is inversely proportional to the degree of node u, which hints that a central node could contribute less during aggregation when it has more neighbors. ***Third***, the off-diagonal elements in the u-th row, i.e., $p_{u,v} = -\frac{\delta(\mathcal{H}_u,\mathcal{H}_v)}{d_u}$ for $\forall u \neq v$, are the scaled-down similarities, in which, $\frac{1}{d_u}$ ensures an equal contribution of all neighbor as central, meanwhile, the negative sign ("$-$") is used to precisely extract the difference between a central and its environment. Our results in Sec. V-E show that the hyper-weight matrix is powerful in merging structure and similarity.

Aggregation using hyper-weight matrix. Since matrix $\mathbf{P}_\mathcal{G}$ incorporates both structural and distance information, we use it as the weight matrix for graph aggregation, such as a_u of node u, as shown in below:

$$a_u = Sign(\bigoplus_{v\in\mathcal{N}_u^*} (p_{u,v} \times \mathcal{H}_v)) \quad \text{with} \quad \mathcal{N}_u^* = \mathcal{N}_u \cup \{u\} \tag{2}$$

where, $\mathcal{N}_u$ contains all the neighbors of u, the bundling operation ($\oplus$) collects all weighted similarities, both central u and its periphery, as the aggregation on u, and the $Sign$ operation maps each elements to $+1$ or -1 according to the sign of the collected results for a convenient combination in next (Fig. 5(a)). In summary, the similarity and connectivity between centrals and peripheries are concentrated in the hyper-weight matrix to aggregate surrounding hypervectors, which faithfully mimics the embedding flow in traditional GNNs.

C. Concatenation for Lightweight Combination

As mentioned in Sec. II-D, current HDC methods always neglect central nodes, which weakens their bundling-based combination. To address this issue, in this work, we directly concatenate the feature hypervectors with its aggregation, i.e., $\tilde{\mathcal{H}}_u = CONCAT(\mathcal{H}_u, a_u)$, which allows a central to occupy half of its entire hypervector length and significantly enhances its role in the final *reasoning* (Fig. 5(b)). The effectiveness of this method stems from three sources: 1) a_u provides all peripheral situation of a node, including its neighbor similarity and connectivity; 2) as the initial feature, $\mathcal{H}_u$ memorizes the intrinsic properties of a central; and 3) the equal occupancies of the two information balance their impacts successfully. In addition, the ultra-short hypervectors used in our implementation removes the worry about the memory expansion by the twice length with $\tilde{\mathcal{H}}_u$ (Sec. IV-B).

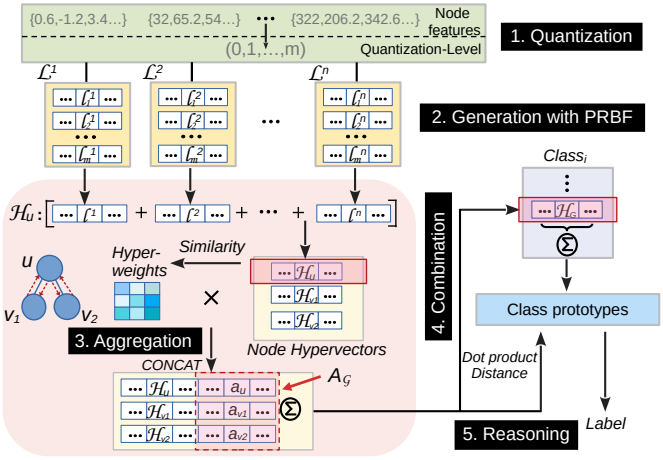


Fig. 6: The framework of CiliaGraph.

IV. Implementation of Memory-Efficient CiliaGraph

A. The Workflow of CiliaGraph

Based on the above operations, CiliaGraph, a new HDC-based graph processing framework, is proposed to achieve both memory-efficient and fast graph classifications, as shown in Fig. 6.

1) Quantization based on prior distribution. To determine the quantization levels mentioned in Sec. III-A, our approach necessitates a priori distribution of each feature that's pertinent to specific tasks, e.g., COIL-RAG, Synthie [10] ... The statistical outcomes are fed into a K-means algorithm [16] to be classified as m quantization centers, i.e., $(\mu_1^i, \mu_2^i, \ldots, \mu_m^i)$, which are associated with their hypervector $\mathcal{L}^i$ for a node i. This method is able to statically find fine-grained centers within sampling ranges, improving the representability compared to traditional uniform levels of quantization [7].

2) Generation of feature-rich hypervectors. To encode n numerical features of a node u into a hypervector, first of all, those features $F_u = (f_1, f_2, \cdots, f_n)$ are individually clustered into quantization centers and replaced with their hypervectors, i.e., $(l^1, l^2, \ldots, l^n)$, by being fetched from $\mathcal{L}^i$. After that, they bundled into a $\mathcal{H}_u$ hypervector, see Sec. III-A. In theory, this approach can merge and encode any number of feature elements without the memory explosion problem under feature-rich contexts.

3) Aggregation using symmetric flows. For symmetric aggregation like GNNs, CiliaGraph propagates hypervectors bidirectionally along all edges, resulting in a central aggregation process at each node. This flow is controlled by the Hyper-weight matrix and all aggregated results are stored in a aggregation matrix $A_\mathcal{G}(a_1, a_2, \cdots, a_{|\mathcal{V}|})$, occupying $|\mathcal{V}| \times D$ memory space. Due to the highly parallelizable nature with our vectorized aggregation, it can easily leverage almost all parallel architectures, such as multi-cores, GPUs and FPGAs.

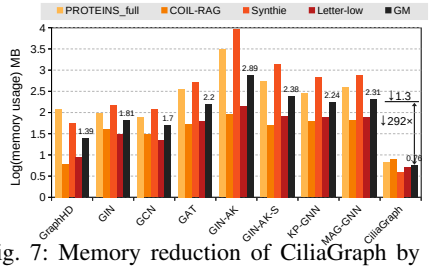


Fig. 7: Memory reduction of CiliaGraph by 1-2 orders compared to the eight SOTAs.

TABLE I: Memory, training time, and test accuracy of seven models. The top three are highlighted by **First**, **Second**, **Third**.

Dataset \ Methods	GraphHD (CPU/GPU)	GIN	GCN	GAT	GIN-AK	GIN-AK-S	KP-GIN	MAG-GNN	**CiliaGraph(Ours)** (CPU/GPU)
Training Time (s)↓									
PROTEINS_full	3.2 / 7.6	17.2	15.5	36.3	58.8	238.2	58.7	60.3	**1.3** / 1.4
COIL-RAG	4.1 / 14.4	42.4	35.6	94.3	135.8	492.3	102.6	120.5	**3.2** / 4.5
Synthie	1.8 / 3.1	8.5	7.4	11.4	65.2	156.7	30.1	38.8	**0.5** / 0.6
Letter-low	5.2 / 30.0	23.6	22.3	29.5	74.5	303.9	60.3	70.6	**1.9**/ 2.6
Test Accuracy (%)↑									
PROTEINS_full	57.84	68.23	70.72	65.55	67.86	65.78	**73.01**	**73.13**	**73.96**
COIL-RAG	6.79	90.75	89.41	91.03	**96.51**	**98.36**	95.01	95.79	**95.88**
Synthie	50.63	63.31	59.50	63.84	**94.15**	92.03	92.31	**94.35**	**93.67**
Letter-low	50.81	94.02	85.66	81.76	**98.47**	96.28	96.44	**97.42**	**97.76**

4) Combination with high memory-efficiency. A concatenation-based combination requires accessing two data in memory: the feature hypervector and its aggregated hypervector. Then, these hypervectors are concatenated together, i.e., $[\mathcal{H}_i, a_i]$, and directly forwarded to subsequent *reasoning* step. By representing l^i and a_i with only a few hundred bits, e.g., $D = 120$ in our experiments (Sec. V-C), CiliaGraph ensures that the concatenation remains memory-efficient.

5) Reasoning for one-shot graph classification. The reasoning is same to the classical HDC processing that bundles all graph nodes together, i.e., $\oplus_{i=1}^{|\mathcal{V}|}[\mathcal{H}_i, a_i]$, and calculates the dot product distances between the result and all class prototypes (a set of label hypervectors). The class with the smallest distance is finally selected as the reasoning output. In next, we present the theoretical evidence about why our short hypervectors outperforms conventional long vectors in memory usage.

B. Memory Efficiency of CiliaGraph: A Theoretical Analysis

The substantial reduction in hypervector length achieved by our CiliaGraph is theoretically grounded in below. For two unit vectors x and y, they are ϵ-*quasi-orthogonal* if $(|x \cdot y| \leq \epsilon)$ and $\epsilon \in [0, 1)$, where ϵ is an angle relaxation easing the strict orthogonality [17]. Intuitively, when using ϵ-*quasi-orthogonal* to replace the traditional strict one in HDC, a larger ϵ can accommodate more mutually orthogonal hypervectors under a fixed vector length D. The theories in [12] and [17] have established that the magnitude of ϵ-*quasi-orthogonal* bipolar hypervectors, i.e., $dim_\epsilon(D)$, grows exponentially with D, i.e., $dim_\epsilon(D) = e^{D\epsilon^2/2}$. Since our *plastical orthogonal ring* (Sec. III-A) only requires the approximate orthogonality (Fig. 3) between the first and last hypervectors in each feature space ($\delta(l_1^i, l_m^i) \approx 0$), it can hold a large amount of features with significantly reduced D. The formal proof is shown in below.

Theorem 1. *For a graph with n node attributes and m quantization levels, using plastic random bit-flip encoding with quasi-orthogonality threshold $\epsilon = \frac{2}{(m-1)}$, the minimum dimension D required to ensure at least $2n$ quasi-orthogonal hypervectors satisfies $D \approx \frac{(m-1)^2}{2} \ln 2n$.*

Proof. The Hamming distance between l_1^i and l_m^i for the i-th space $\mathcal{L}^i$ is $\sum_{i=1}^{m-1} r_i \times D = \frac{D \times m}{2(m-1)}$, indicating the number of differing bits. Converting these into unit vectors, the value of ϵ is obtained as $\epsilon = |l_1^i \cdot l_m^i| = |1 - 2 \times \frac{m}{2(m-1)}| = \frac{1}{(m-1)}$. To ensure robust quasi-orthogonality while reducing computational overhead, $\epsilon = \frac{2}{(m-1)}$ is set. Since l_1^i and l_m^i from each $\mathcal{L}^i$ must maintain quasi-orthogonality, and this property must hold across all n attribute groups, we require $dim_\epsilon(D) = 2n$. Thus, we derive the minimum required dimensions $D \approx \frac{(m-1)^2}{2} \ln 2n$. □

According to above theory, the optimal D for the typical datasets tested in this work are between $[37, 122]$, reducing $1 - 2$ order of memory usage compared to traditional HDC-based GNNs (Sec. V-A).

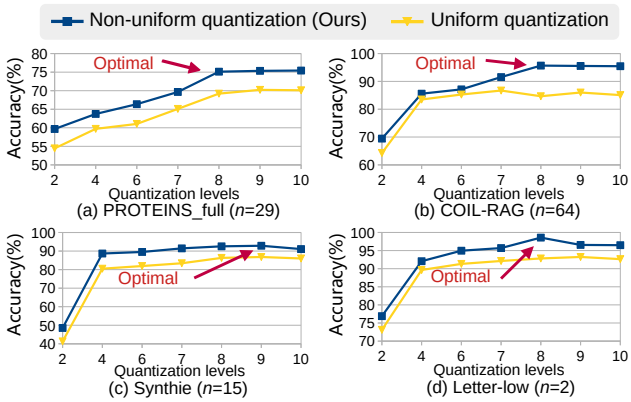


Fig. 8: Optimal quantization levels.

V. Experimental Results

To setup the experiments, seven real-world datasets selected from Tudataset testbed [10] to cover variant magnitudes of node features (n), including Letter-high (2), Letter-low (2), BZR (3), AIDS (4), Synthie (15), PROTEINS_full (29) and COIL-RAG (64).

For the baseline models, we tested: 1) HDC-GNN: GraphHD; 2) GNNs: GIN [18], GCN [19], GAT [20]; 3) SOTA k-hop GNNs: GIN-AK+, GIN-AK+-S [8], KPGIN [13] and MAG-GNN [21], where GIN-AK+-S serves as a sampling model for GIN-AK+ that reduces resource overhead. All GNNs are replicated on GPUs (duel NVIDIA GeForce RTX 4090), while HDCs are tested on both CPUs (AMD EPYC CPUs with 96 cores) and GPUs.

A. Memory reduction of CiliaGraph

As shown in Figure 7, the memory consumption of our proposed CiliaGraph is compared against eight baseline methods across four real-world graph datasets, with the vertical axis representing the logarithmic value of memory usage. By enjoying the ultra-short hypervectors ($D = 120$), CiliaGraph achieves an average memory saving of $292\times$, corresponding to a reduction of $1 - 2$ orders of magnitude in memory consumption. Notably, on the Synthie dataset, CiliaGraph exhibits only a 1.48% performance difference from the best-performing GNN model (Table I) while reducing memory usage by an impressive $2341\times$. This demonstrates that CiliaGraph is a memory-efficient graph classification model, making it well-suited for practical deployment and operation in resource-constrained environments such as on-edge graph processing.

B. Latency and Accuracy of Graph Classification Tasks

The ultra-shot hypervectors of CiliaGraph significantly reduce processing latency across both CPU and GPU platforms for the full graph classification tasks, while maintaining competitive accuracy (Table I). CiliaGraph achieves 85.09% higher accuracy than GraphHD on COIL-RAG, demonstrating its advantage via involving the intrinsic attributes of nodes. Particularly on the Synthie dataset [22], CiliaGraph's performance trails the best GNN model by only 1.48%, while offering a $2341\times$ reduction in memory usage and

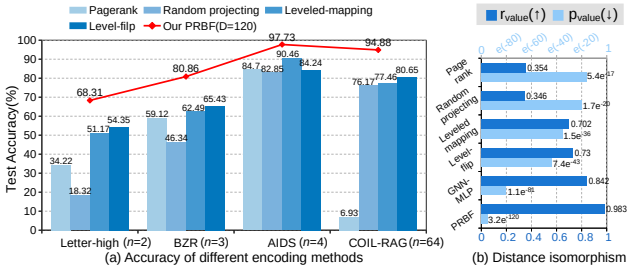


Fig. 9: The impact of different encoding methods.

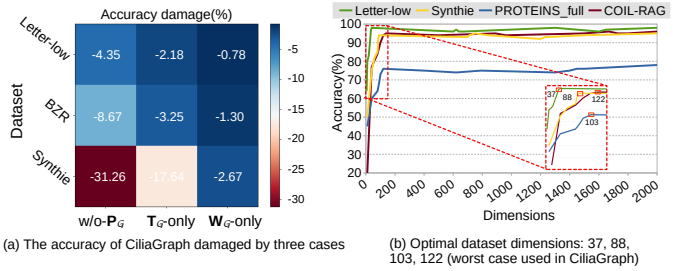


Fig. 10: Hyper-weights ablation and dataset-specified dimensions.

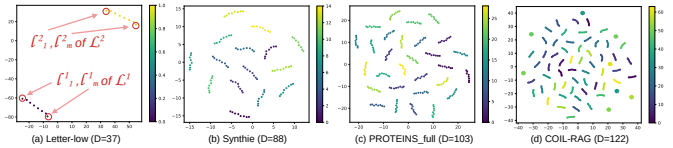


Fig. 11: t-SNE visualisation of initial $\mathcal{L}^i$. Both of coherence and orthogonality are stable across all optimal length of hypervectors (D).

a 313× speedup in training. On the GPU platforms, we didn't observe any acceleration of CiliaGraph on GPUs. The *nvprof* tool reports reveal that the computing reduction of CiliaGraph leads to the negligible *kernel* workloads that are even insufficient to hide the CPU-GPU transferring time for the hypervectors, which shows that CiliaGraph is such an energy-efficient model that general accelerators could be needlessly.

C. Picking Quantization Levels for the Worst Cases

To pick quantization levels can significantly affect data clustering and model accuracy. According to the formula provided in Theorem 1, the quantization level also determines the minimum required dimension. Thus, in all SOTA HDC solution, the dimension $D = 10,000$ is initially set to identify the optimal quantization levels. Fig. 8 illustrates the results for four datasets, with the yellow line indicating uniform quantization. Most datasets achieve the highest and most stable accuracy at $m = 8$. Subsequently, the formula changes to $D \approx \frac{(8-1)^2}{2} \ln 2n$. Therefore, even for the COIL-RAG dataset [10], [23], which has the largest number of node attributes at $n = 64$, the minimum required dimension is $D \approx \frac{(8-1)^2}{2} \ln (2 \times 64) \approx 118.87$ bits. For convenient in setting, D is uniformly approximated to 120, rather than 118.87, to standardize the effectiveness evaluation for all tests on CiliaGraph.

D. The Expression of Rich Features with PRBF Encoding

To assess the expressive power of the CiliaGraph node encoding method, we compare it against four alternative frameworks: 1) *PageRank*; 2) *Random projecting*; 3) *Leveled-mapping*; and 4) *Level-filp* with [15] flipping methods. All these four frameworks use the dimension $D = 10000$. As shown in Fig. 9(a), CiliaGraph achieves optimal performance across all tested datasets, with an average accuracy 15% higher than the *Leveled-mapping* and 39% higher than the *Pagerank*. These significant improvements evidence the effectiveness of our model and encoding methodology. Furthermore, the Pearson correlation coefficient [14] is employed to validate that CiliaGraph effectively maintains isomorphism (Fig. 9(b)) in feature distances between nodes before and after encoding, thereby preserving the original data features in $\mathcal{HD}$ space. In addition, the precise encoding is seen as a key reason to combat the Noise-1.

E. Ablation Experiment of Hyper-Weight Matrix

To validate the effectiveness of the more precise similarity weights and transition matrix introduced in Sec. III-B, CiliaGraph is compared to three ablating cases: 1) "w/o-$\mathbf{P}_\mathcal{G}$" is *without similarity weights matirx and transition matrix*; 2) "$\mathbf{T}_\mathcal{G}$-only" is *without similarity weights matrix*; and 3) "$\mathbf{W}_\mathcal{G}$-only" is *without transition matrix*. The accuracy differences of these frameworks are shown in Fig. 10(a). It is observed that the performance of all three frameworks progressively deteriorates, particularly for w/o-$\mathbf{P}_\mathcal{G}$ and $\mathbf{T}_\mathcal{G}$-only. This accentuates the inadequacies of basic edge encoding techniques which overlook the significance of connection strengths. The results of $\mathbf{W}_\mathcal{G}$-only demonstrate that integrating transition matrices with similarity weight matrices can enhance the model expressiveness, offering a refined depiction of inter-node connections. The results demonstrate that only the combined usage of $\mathbf{W}_\mathcal{G}$ and $\mathbf{T}_\mathcal{G}$ can prevent significant Noise-2.

F. The Optimal Dimensions for Individual Datasets

To further quantify the confidence of the minimal D proposed in Sec. IV-B, we first calculate the theoretical values of D via the Theorem 1 for each of datasets, including Letter-low (33.96), Synthie (83.33), PROTEINS_full (99.48), and COIL-RAG (118.87). Then, in measuring part, our experiments span D from the theoretical minimum 5 to 2000 and evaluate the CiliaGraph for the four datasets, as shown in Fig. 10(b). The accuracy curves indicate that they achieve their optimal performances at 37, 88, 103, and 122 respectively. The comparison between the theoretical and measured values hints that our theorem achieves predicting accuracy of 95.81% on average, which is a well predictor of hypervector length. In addition, a dimension reduction of features (t-SNE) in Fig. 11 shows that the generated initial hypervectors exhibit not only intra-cluster coherence (high k-means inertia) but also inter-group quasi-orthogonality among different features, which is the reliable short encoding exactly as expected in Sec. III-A.

VI. Conclusions

This paper presents a novel approach to address the memory challenges posed by long hypervectors in HDC-based graph processing. By introducing three new operators for feature-rich hypervectors, we improve the efficiency of feature representation and extraction, enabling HDC to operate more like traditional GNNs with significantly reduced memory requirements. The experiments indicate that the short vector length we predict is very close to the optimal in terms of minimizing memory usage.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 62272393 and the Aeronautical Science Foundation of China under Grant 20240043053002.

REFERENCES

[1] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-dimensional computing challenges and opportunities for ai applications," *IEEE Access*, vol. 10, pp. 97 651–97 664, 2021.

[2] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Simunic, "Hydrea: Utilizing hyperdimensional computing for a more robust and efficient machine learning system," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, pp. 1–25, 2022.

[3] I. Nunes, M. Heddes, T. Givargis, A. Nicolau, and A. Veidenbaum, "Graphhd: Efficient graph classification using hyperdimensional computing," in *DATE(2022)*, pp. 1485–1490.

[4] H. Li, F. Liu, Y. Chen, and L. Jiang, "Hypernode: An efficient node classification framework using hyperdimensional computing," in *ICCAD(2023)*, pp. 1–9.

[5] J. Kang, M. Zhou, A. Bhansali, W. Xu, A. Thomas, and T. Rosing, "Relhd: A graph-based learning on fefet with hyperdimensional computing," in *ICCD(2022)*, pp. 553–560.

[6] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, pp. 139–159, 2009.

[7] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE CASS Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[8] L. Zhao, W. Jin, L. Akoglu, and N. Shah, "From stars to subgraphs: Uplifting any GNN with local structure awareness," in *ICLR(2022)*.

[9] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–40, 2022.

[10] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *ICML 2020 workshop "Graph Representation Learning and Beyond"*, 2020.

[11] T. Yu, Y. Zhang, Z. Zhang, and C. M. De Sa, "Understanding hyperdimensional computing for parallel single-pass learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1157–1169, 2022.

[12] Z. Yan, S. Wang, K. Tang, and W.-F. Wong, *Efficient Hyperdimensional Computing*. Springer Nature Switzerland, 2023, p. 141–155.

[13] J. Feng, Y. Chen, F. Li, A. Sarkar, and M. Zhang, "How powerful are k-hop message passing graph neural networks," *Advances in NIPS*, vol. 35, pp. 4776–4790, 2022.

[14] Z. Šverko, M. Vrankić, S. Vlahinić, and P. Rogelj, "Complex pearson correlation coefficient for eeg connectivity analysis," *Sensors*, vol. 22, no. 4, p. 1477, 2022.

[15] I. Nunes, M. Heddes, T. Givargis, and A. Nicolau, "An extension to basis-hypervectors for learning from circular data in hyperdimensional computing," in *DAC(2023)*, pp. 1–6.

[16] E. Umargono, J. E. Suseno, and S. V. Gunawan, "K-means clustering optimization using the elbow method and early centroid determination based on mean and median formula," *Proceedings of the 2nd ISSTEC*, pp. 234–240, 2020.

[17] P. C. Kainen and V. Kůrková, "Quasiorthogonal dimension," in *Beyond traditional probabilistic data processing techniques: Interval, fuzzy etc. Methods and their applications*. Springer, 2020, pp. 615–629.

[18] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations (ICLR)*, pp. 1–14, 2017.

[20] Y. B. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, "Graph Attention Networks," *ICLR*, vol. 11731 LNCS, pp. 566–577, 2018.

[21] L. Kong, J. Feng, H. Liu, D. Tao, Y. Chen, and M. Zhang, "Mag-gnn: reinforcement learning boosted graph neural network," in *NIPS(2023)*.

[22] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, "Scalable kernels for graphs with continuous attributes," *Advances in neural information processing systems*, vol. 26, 2013.

[23] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *Structural, Syntactic, and Statistical Pattern Recognition December 4-6, 2008. Proceedings*. Springer, 2008, pp. 287–297.

ClusterKV: Manipulating LLM KV Cache in Semantic Space for Recallable Compression

Guangda Liu, Chengwei Li, Jieru Zhao†, Chenqi Zhang, Minyi Guo
School of Computer Science, Shanghai Jiao Tong University
† Corresponding author: zhao-jieru@sjtu.edu.cn

***Abstract*—Large Language Models (LLMs) have been widely deployed in a variety of applications, and the context length is rapidly increasing to handle tasks such as long-document QA and complex logical reasoning. However, long context poses significant challenges for inference efficiency, including high memory costs of key-value (KV) cache and increased latency due to extensive memory accesses. Recent works have proposed compressing KV cache to approximate computation, but these methods either evict tokens permanently, never recalling them for later inference, or recall previous tokens at the granularity of pages divided by textual positions. Both approaches degrade the model accuracy and output quality. To achieve efficient and accurate recallable KV cache compression, we introduce ClusterKV, which recalls tokens at the granularity of semantic clusters. We design and implement efficient algorithms and systems for clustering, selection, indexing and caching. Experiment results show that ClusterKV attains negligible accuracy loss across various tasks with 32k context lengths, using only a 1k to 2k KV cache budget, and achieves up to a 2× speedup in latency and a 2.5× improvement in decoding throughput. Compared to SoTA recallable KV compression methods, ClusterKV demonstrates higher model accuracy and output quality, while maintaining or exceeding inference efficiency. Our code is available at https://github.com/sjtu-zhao-lab/ClusterKV.**

I. Introduction

Large Language Models (LLMs) have been widely deployed in a variety of applications, such as natural language understanding, coding copilots and chatbots. As LLMs are used for more complex tasks, the need for long context length is rapidly increasing to handle tasks such as long-document QA, repository-level code understanding, and complex logical reasoning [1–3]. This drives LLMs to support larger context windows, expanding from the original 4k to 32k [4, 5], 128k [6] and even up to 1M [7]. While models can support larger context window with continual training or extrapolation [5], inference with long contexts poses significant challenges on efficiency. As the size of key-value (KV) cache increases linearly with the context length, it can exceed the capacity of GPU memory, leading to inference failure or extremely high latency. Since the autoregressive decoding stage is typically the performance bottleneck of LLM inference and is memory-bound [8], accessing the KV cache of all previous tokens results in increased latency for long-context inference.

To mitigate the issues, recent works compress KV cache by selecting a subset of tokens and utilizing their keys/values to approximate attention computation. The tokens are typically selected with fixed patterns [9] or based on their attention weights [10–14], which represent each token's importance in attention computation. In most compression methods, unimportant tokens are evicted permanently; once evicted, they are never recalled in subsequent decoding steps, as shown in Fig. 1b. However, the token importance is dynamically changing during decoding: tokens initially considered unimportant and evicted may become important at later steps and vice versa. Non-recallable KV cache compression greedily evicts tokens based on positions or attention weights of tokens at the current decoding step, failing to capture the dynamic feature. This can degrade model accuracy and output quality.

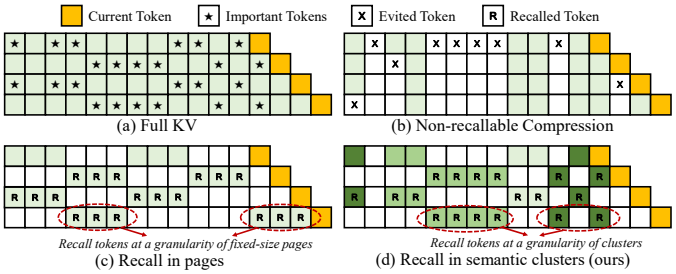


Fig. 1: Comparison of KV compression methods. Green boxes represent tokens selected for attention computation.

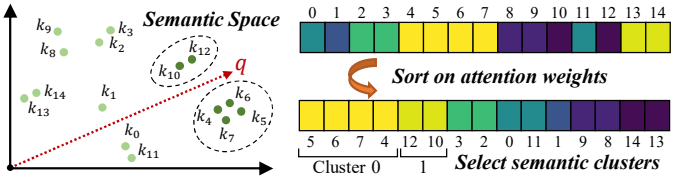


Fig. 2: Semantic space and attention weights of tokens in the last step of Fig. 1d. Lighter boxes indicate larger weights.

While making tokens recallable is promising, it incurs prohibitively high cost due to the computation of attention weights over all previous tokens. To reduce the cost, as illustrated in Fig. 1c, Quest proposed recalling at the granularity of pages [15]. A page consists of $page_size$ consecutive tokens, allowing Quest to reduce the selection overhead by a factor of $1/page_size$. However, since pages are divided simply by textual positions of tokens, internal fragmentation becomes an issue: a recalled page may contain unimportant tokens, wasting budget that could be allocated to truly important tokens.

To achieve efficient and accurate KV cache compression, we introduce ClusterKV, which recalls tokens at the granularity of token clusters in the semantic space, referred to as *semantic clusters*. As illustrated in Fig. 2, tokens that are close in their semantic space or key vector space exhibit similar attention weights. Therefore, as shown in Fig. 1d, ClusterKV

recalls tokens at the granularity of semantic clusters to ensure more precise token recall. Moreover, ClusterKV incorporates specialized system designs and optimized kernels to minimize recall overheads. Experimental results demonstrate that ClusterKV attains negligible accuracy loss across various tasks with context lengths up to 32k, using only a 1k to 2k KV cache budget. It also delivers up to a 2× speedup in latency and a 2.5× improvement in decoding throughput. Compared to state-of-the-art recallable KV compression methods, ClusterKV achieves significantly higher model accuracy and output quality, while maintaining or surpassing inference efficiency.

II. Background and Motivation

A. LLM Inference and KV Cache

LLMs encompass multiple Transformer layers, each containing a multi-head attention (MHA) module and a feed-forward network (FFN) with residual connections and normalization operations [16]. In MHA, the input tensor is linearly projected into query, key and value tensors ($Q, K, V \in \mathbb{R}^{N\times d}$) for each head, where N is the input length and d represents the number of channels or hidden dimensions per head. The MHA output is defined as $softmax(\frac{QK^T}{\sqrt{d}})V$, with outputs of all heads concatenated for subsequent FFN and normalization.

For generative inference, LLMs produce tokens in an autoregressive manner, appending each generated token to the input to generate the next token. To avoid the recomputation of K and V of previous tokens, these tensors are stored in memory for reuse, which is known as *KV cache*. The LLM inference includes two stages: prefill and decoding. The prefill stage processes the entire input sequence, computes KV cache and generates the first output token. During decoding, the query vector q of the latest generated token and K, V of previous tokens are used to compute attention for generating the next token, formulated as $softmax(\frac{qK^T}{\sqrt{d}})V$, where $q \in \mathbb{R}^{1\times d}$, $K, V \in \mathbb{R}^{L\times d}$ and L is the context length of previous tokens.

B. Long Context Inference and KV Cache Compression

Long context is an emerging trend of LLM inference, such as long-document QA [1, 17], repository-level code understanding [2] and complex logical reasoning [3]. Moreover, supported context windows of LLMs are extending rapidly, from 4k tokens of GPT3.5 and Llama-2 to 32k [5], 128k [6] and even 1M tokens [7]. However, long-context inference incurs significant memory and computation cost. The size of KV cache and complexity of attention computation during decoding increase linearly with context length, resulting in low inference efficiency or even inference failures.

Recent works reveal the sparsity of attention computation, i.e., only a small subset of tokens contributes to most of attention outputs [10, 13]. This observation enables KV cache compression by selecting a subset of tokens to approximate attention computation, which is formulated as $softmax(\frac{qK_S^T}{\sqrt{d}})V_S$, where $K_S, V_S \in \mathbb{R}^{B\times d}$ represents the keys and values of selected tokens, and B is the budget size of KV cache after compression. By setting a fixed budget, the KV cache size and decoding cost remain stable, regardless of the context length.

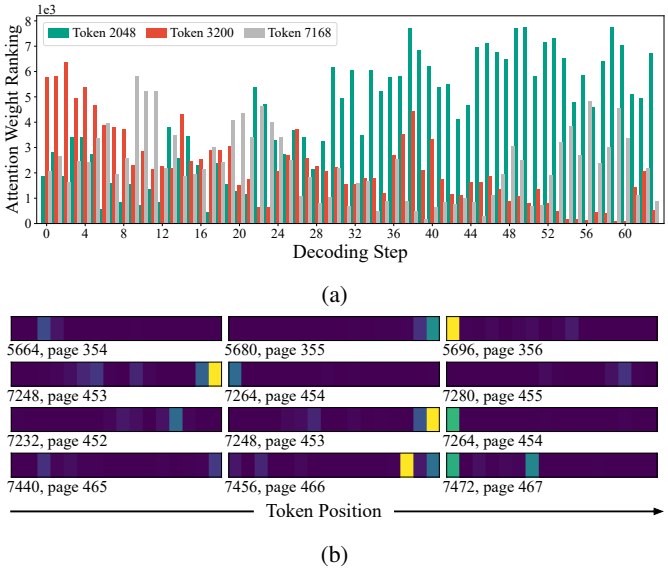


Fig. 3: (a) Variation in token importance across decoding steps with a context length of 8192. (b) Internal fragmentation of important tokens at the granularity of pages ($page_size = 16$).

Typically, K_S and V_S are selected based on attention weights (qK^T), as tokens with higher attention weights contributes more to the attention computation [10–12].

C. Related Work and Motivation

KV cache compression should be recallable. While selecting a subset of KV cache for computation ensures inference efficiency, computing all the attention weights for selection introduces substantial overhead. Therefore, most existing works only compute attention weights over tokens that have been selected, rather than for all previous tokens [10–12]. In this way, keys and values of tokens not selected at one decoding step are permanently evicted from the KV cache and never recalled in later inference steps, as shown in Fig. 1b.

However, we observe that the token importance changes dynamically during inference. Token that are unimportant with low attention weights at one decoding step can become important in later steps, and vice versa. Figure 3a shows changes in attention weight rankings during 64 decoding steps of Llama-3-8B. For instance, token 3200 is initially unimportant but becomes crucial after 20 steps, while the opposite occurs for token 2048. And importance of all tokens can fluctuate throughout inference, as seen with token 7168. Therefore, non-recallable compression inevitably overlooks some tokens with dynamically changing importance, leading to reduced model accuracy and a decline in output quality.

Defects of existing recallable KV compression methods. However, achieving recallable compression by computing attention weights with all previous tokens incurs an unacceptable cost of $O(Ld)$, which is comparable to attention computation with full KV cache and negates the benefits of compression. To reduce the cost, InfiniGen [18] reduces the dimensions of q and K by generating partial query and key weights offline using singular value decomposition, and projecting hidden states with these partial weights. However, it requires generating

and storing partial keys in addition to original keys, and the selection cost still scales linearly with the context length L.

Quest [15] selects tokens at the granularity of pages, which consist of several consecutive tokens, as depicted in Fig. 1c. To estimate importance of a page, it uses the per-channel maximal keys of all tokens within a page to compute attention weights. This approach reduces the selection cost to $O(Ld/page_size)$ as only one attention weight is needed for every $page_size$ tokens. However, recalling pages which are simply divided by token positions leads to internal fragmentation of important tokens. A selected page may contain only a few important tokens, while including unimportant ones for attention computation and wasting the KV cache budget. We illustrate this issue using attention heatmaps of Llama3-8B with an 8k context length in Fig. 3b, where lighter cells represent more important tokens with higher attention weights. As shown, each page of 16 tokens contains only one or two important tokens, and in some cases, two pages or 32 tokens are required to include two consecutive important tokens.

III. Algorithm Design

A. Problem Formulation and Design Rationale

Problem Formulation. For approximated attention computation with selected tokens formulated in Sec. II-B, let K_S to be $(k_{i_1}, k_{i_1}, ..., k_{i_B})^T$, where $I_T = \{i_1, i_2, ..., i_B\}$ denotes the indices of selected tokens. Our goal is to select tokens which contribute most to attention weights, thereby approximating the original computation as closely as possible. Specifically, I_T is supposed to be $\arg\max \sum_{i \in I_T} qk_i^T$, which corresponds to selecting tokens with top-B largest attention weights.

Design Rationale. Our observation is that *tokens which are close in semantic space tend to have similar attention weights for a given q*. Therefore, we propose KV selection at the granularity of semantic clusters. We first apply clustering to tokens in the semantic space. Then, we compute attention weights only with respect to the cluster representations, rather than individual tokens, and select clusters with the largest attention weights. Since the number of clusters is typically an order of magnitude smaller than the number of tokens, cluster-based selection significantly reduces recall overhead.

B. Clustering in the Semantic Space

Semantic Distance. As for a given q, attention weights are associated only with the key tensors, we measure the semantic distance between tokens by calculating the distance between corresponding key vectors. Furthermore, we find that *cosine similarity* is more suitable than L2 or inner product distances. This is due to the presence of outlier channels with large magnitudes in key vectors [19], which can cause drastic changes in L2 or inner product distances. Thus, we define distance between token i and token j in the semantic space as $\mathcal{D}(i,j) = 1 - \frac{\langle k_i, k_j \rangle}{|k_i| \cdot |k_j|}$, where distance is smaller for vectors with larger cosine similarity.

Clustering. We apply a simple K-means algorithm for clustering over key vectors as shown in Fig. 4 [20]. We first randomly sample key vectors as the initial centroids.

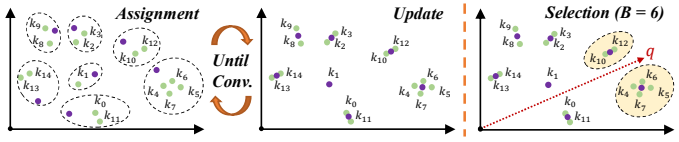


Fig. 4: Clustering and selection process. Green dots represent key vectors, and purple dots represent centroids of clusters.

Subsequently, we alternatively perform the assignment and update steps until convergence. In the assignment step, each key vector is assigned to the nearest centroid and given a corresponding *cluster label* based on the distance $\mathcal{D}$, i.e., assigned to the centroid with the maximum cosine similarity. Then in the update step, the mean of keys assigned to the same centroid are used as the new centroid. The algorithm converges when the assignment no longer changes, and keys assigned to the same centroid form a *semantic cluster*, with the centroid as the cluster representation.

During LLM inference, we first apply clustering to the key vectors of prompt tokens *after the prefill stage*. However, we note an exception for the initial tokens, referred to as attention sinks [9]. These tokens typically appear as outliers in the clustering process, as they are distant from other tokens in the semantic space. Therefore, we always retain the first 16 tokens and apply clustering to the subsequent tokens, generating C_0 centroids. We set $C_0 = \frac{L}{80}$, as our experiments indicate that for a 32k context, using 400 clusters achieves a balance between efficiency and accuracy. For tokens generated *in the decoding stage*, clustering is applied every m decoding steps to the key vectors of m generated tokens, creating C_+ new centroids. Since clustering keys of generated tokens together with those from the prefill stage can incur significant overhead, we instead apply clustering *within* the generated tokens only. To amortize the cost, we set C_+ and m to 4 and 320, respectively.

C. Selection at the Granularity of Semantic Clusters

Selection. We denote cluster centroids as $\mu_1, \mu_2, ..., \mu_C \in \mathbb{R}^d$. To select important tokens for a given query q, we sort those centroids based on their attention weights, i.e., $q\mu_i^T$, in descending order. While keys are clustered using cosine similarity distance, the distance between query vector and centroids is measured with inner product, as it better aligns with attention weight computation. Intuitively, keys assigned to clusters with centroids that have larger attention weights tend to have larger attention weights for the given q. Therefore, we retrieve the sorted centroids and collect KV of tokens from the corresponding clusters, until the top-B most important tokens are selected, as shown in Fig. 4.

D. Efficiency Concerns

Cluster-based selection avoids the internal fragmentation of important tokens and can achieve higher accuracy compared to page-based selection. However, it incurs efficiency concerns.

Concern 1. For clustering, the computational cost is $O(n_i CLd)$ where n_i is the number of iterations until convergence and C is the number of clusters, which is higher than the cost of obtaining page representations, such as per-channel maximal key vectors in Quest [15], which is $O(Ld)$.

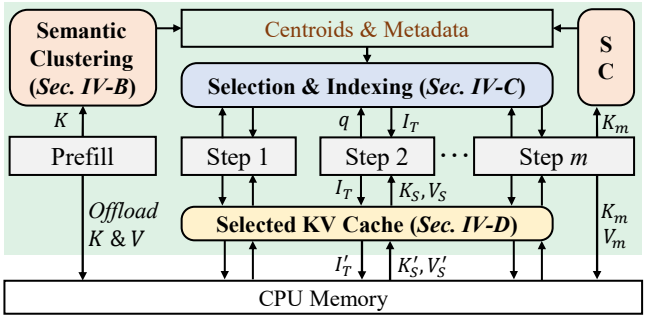


Fig. 5: System overview of ClusterKV. The green box represents components running on the GPU.

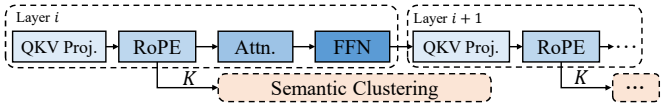


Fig. 6: Overlapping of clustering with other operations.

Concern 2. For selection in page-based methods (Fig. 1c), since page size is fixed and tokens within a page are consecutive, the number of needed pages can be easily computed as $\frac{B}{page_size}$, and the indices of selected tokens can be directly derived from the indices of selected pages. However, for semantic clusters, the sizes can vary, and the token positions within a cluster are dynamic and discontinuous. Therefore, the number of clusters needed and the indices of selected tokens cannot be easily determined in the same way as for pages.

Our system design addresses both concerns in Sec. IV.

IV. System Design & Implementation

A. System Overview

The system overview is shown in Fig. 5. During *the prefill stage*, the key tensors are processed with semantic clustering (SC) on GPU, producing centroids and corresponding metadata. The generated KV tensors are offloaded to CPU memory. During *the decoding stage*, as described in Section III-C, attention weights of the query vector q and cluster centroids are computed to determine the importance of each cluster. The results, along with clustering metadata, are used to generate indices (I_T) for selected tokens, which are then used to load selected KV (K_S, V_S) from CPU memory to GPU memory for attention computation. A cache for selected KV is maintained on GPU, so only KV not already cached (K'_S, V'_S) need to be loaded. For every m decoding steps, the clustering and KV offloading are performed for the m generated tokens.

B. Semantic Clustering

ClusterKV optimizes the efficiency of clustering at both system and kernel levels.

System-level Optimization. As shown in Fig. 6, ClusterKV applies clustering asynchronously, launching intermediately after the keys are computed from the QKV projection and RoPE modules. This allows clustering to be overlapped with other operations, including attention and FFN computation of the current layer, as well as the QKV projection and RoPE of the following layer. By overlapping these processes, ClusterKV minimizes the overhead associated with clustering.

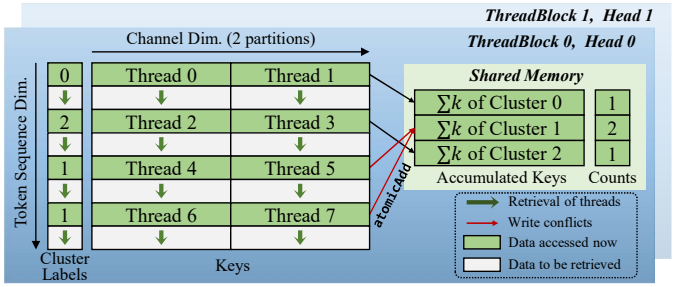


Fig. 7: Execution of the centroid update kernel. In this example, a *ThreadBlock* consists of 8 threads ($BlockSize = 8$), and the channel dimension is divided into two partitions.

Kernel-level Optimization. Since clustering needs to be applied individually for each head, a key optimization is to achieve batched clustering across heads. For the assignment step, which primarily involves `argmin` operations and matrix multiplications between keys and centroids, efficient batched Torch kernels are available [21]. Therefore, we focus on optimizing the centroid update step and implement a custom CUDA kernel where different heads are processed in parallel by individual *ThreadBlocks*, as shown in Fig. 7. The kernel retrieves keys, accumulates keys assigned to the same cluster, records corresponding accumulation counts in the shared memory, and computes the means as new centroids.

A potential issue is write conflict in shared memory, as accumulated keys and counts of the same cluster need to be computed using `atomicAdd`. To mitigate this, we apply several optimizations. As shown in Fig. 7, since distant tokens tend to be assigned to different clusters, we arrange threads in a strided manner along the sequence dimension to minimize the occurrence of concurrent processing of keys within the same cluster. Moreover, we partition the channel dimension into P partitions, with each partition processed by one thread. This results in $\frac{BlockSize}{P}$ keys being processed in parallel. There is a trade-off when choosing P: using a larger P reduces the number of keys processed in parallel and increases the number of iterations along the sequence dimension, while using a smaller P increases the number of iterations along the channel dimension within a thread, and potentially increases the occurrence of write conflicts. To determine the optimal value of P, we perform offline profiling of different P values. We find that for $BlockSize = 512$ using $P = 16$ or $P = 32$ for 128 channels yields optimal performance.

C. Selection and Indexing

The details are shown in Fig. 8. After clustering, ClusterKV stores cluster centroids and corresponding metadata, including cluster sizes, prefix sum, and sorted indices. In this example, k_0 and k_5 are assigned to cluster 2, k_1 is assigned to cluster 0, and k_2, k_3 and k_4 are assigned to cluster 1. The sizes of three clusters can then be obtained. Cluster labels and key indices are sorted by labels, and the sorted indices are stored.

During decoding, ClusterKV computes attention weights between the query vector q and cluster centroids μ, sorting the clusters in descending order of attention weights to determine

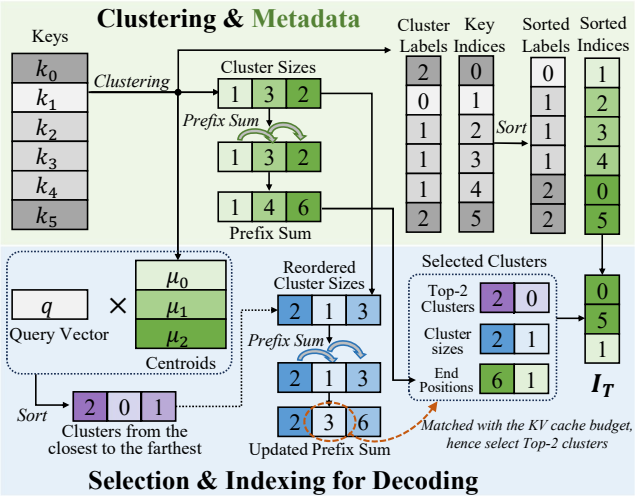


Fig. 8: Process of selection and indexing. Green boxes represent stored metadata from clustering. Here, KV budget is 3.

the closest clusters to q. Next, ClusterKV gathers and reorders their corresponding cluster sizes, and compute prefix sum. If we set the KV cache budget to three, which matches the second prefix sum, top-2 closest clusters will be selected. Then the labels, sizes, and end positions of selected clusters are gathered to determine the indices of selected tokens, I_T. Note that for cases where the summation of selected cluster sizes exceeds the budget, ClusterKV trims tokens from the last selected cluster to adhere to the budget limit.

Similarly to clustering, ClusterKV implements efficient CUDA kernels, processing the indexing of KV heads in parallel with individual *ThreadBlocks*. Metadata related to cluster sizes, which is accessed frequently, is stored in shared memory for improving performance.

D. Caching KV of Selected Tokens

ClusterKV maintains a cluster-granularity cache on GPU to store KV of selected tokens, reducing unnecessary data transfers from CPU memory to GPU memory and enhancing performance. During the decoding stage, the cache retains the KV of selected tokens from the last R decoding steps, along with the labels of corresponding selected clusters. At the current decoding step, the labels of selected clusters are compared with the cluster labels reserved by the cache. Only the KV of clusters not present in the cache are loaded from CPU memory. The cache introduces a trade-off between memory usage and caching effectiveness. In practice, we find that setting $R = 1$, i.e., retaining only the KV states from the last decoding step, strikes a good balance.

V. Evaluation

A. Experimental Setup

We evaluate ClusterKV on an NVIDIA Ada 6000 GPU. To assess model accuracy and output quality, we conduct experiments on eight datasets from LongBench [1], including 2WikiMQA, TriviaQA, HotpotQA, MultiFieldQA, MuSiQue, NarrativeQA, Qasper and GovReport. These datasets cover a variety of tasks, such as single-doc QA, multi-doc QA,

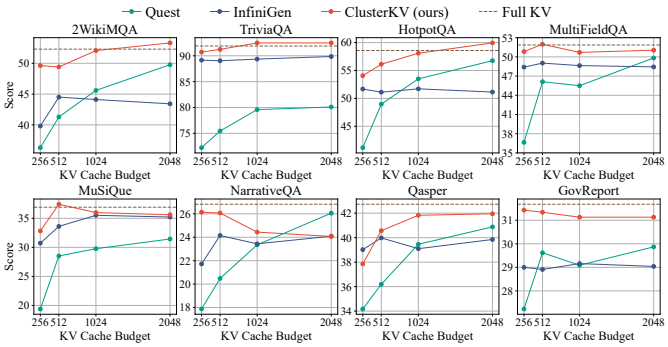


Fig. 9: Results on LongBench of different methods.

TABLE I: Average scores on eight LongBench datasets.

Method \ Budget	256	512	1024	2048
Quest	35.63	40.83	43.23	45.59
InfiniGen	43.69	45.04	45.13	45.14
ClusterKV (ours)	**46.69**	**48.02**	**48.34**	**48.7**
Full KV	49.01			

few shot learning and summarization, with context lengths reaching up to 32k. Additionally, we evaluate ClusterKV on the PG19 dataset for the language modeling task [22]. We use the state-of-the-art long-context GLM4-9B-Chat model for evaluation of model accuracy [7], which supports a context window of up to 128k tokens.

We compare ClusterKV with two state-of-the-art KV cache compression methods: Quest [15] and InfiniGen [18]. Most configurations of these methods remain as in their original settings, such as *page_size* for Quest and the partial weight ratio and selection threshold for InfiniGen. To align with settings of Quest which does not apply selection on the first two layers of the model, we also disable selection and use the full KV cache for the first two layers in ClusterKV and InfiniGen, in both model accuracy and inference performance evaluations. As both Quest and InfiniGen only support efficient inference for Llama-architecture models, we use Llama-3.1-8B for evaluation of inference performance.

B. Model Accuracy

Results on LongBench. Figure 9 presents the results on LongBench. ROUGE-L is used as the score metric for GovReport and F1 score is used for other tasks. We evaluate methods under the KV cache budgets of 256, 512, 1024 and 2048 tokens. ClusterKV outperforms Quest and InfiniGen in most settings, and achieves accuracy comparable to that of using the full KV cache with budgets as low as 1k to 2k tokens.

The average scores across the eight tasks under different budgets are summarized in Table I, where ClusterKV demonstrates significant improvement over Quest and InfiniGen.

Results on Language Modeling. We report the perplexity of the language modeling task in Fig. 10. The prompts are from the PG19 test set, with input lengths ranging from 1 to 32000 tokens. The KV cache budget is uniformly set to 1024 for ClusterKV, Quest, and InfiniGen. As shown, ClusterKV closely aligns with the full KV, with a perplexity deviation

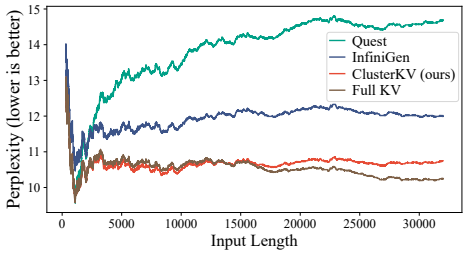


Fig. 10: Perplexity of language modeling with a KV cache budget of 1024 tokens.

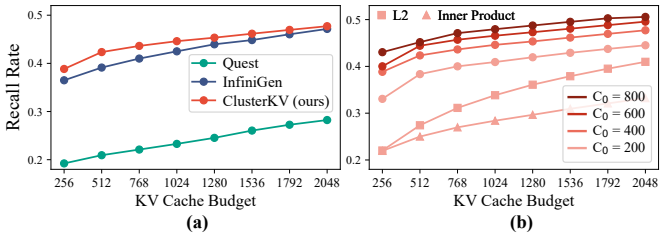


Fig. 11: Recall rate of important tokens: (a) comparison between different methods, and (b) comparison between different configurations of ClusterKV.

up to 0.5, while Quest and InfiniGen show deviations of approximately 4 and 2, respectively.

Recall Rate of important tokens. We extract a sample from the NarrativeQA dataset with a context length of 32k and compute the recall rates of important tokens during inference for different methods. The recall rate is defined as $\frac{|I_T \cap I_T^{true}|}{|I_T^{true}|}$, where $|I_T| = |I_T^{true}| = B$, I_T represents the indices of selected tokens, and I_T^{true} denotes the indices of tokens with the top-B largest attention weights. We report the recall rates averaged across layers, heads and decoding steps, and the budget B is varied from 256 to 2048 in increments of 256.

As shown in Fig. 11a, ClusterKV achieves higher recall rates across all budgets. We further explore the impact of different configurations for ClusterKV, including clustering distance metrics and the number of clusters C_0. In Fig. 11b, cosine similarity used by ClusterKV outperforms both L2 and inner product distance. In addition, increasing C_0 improves recall rates, while the improvements become less significant when $C_0 > 400$. Consequently, we use $C_0 = 400 = \frac{L}{80}$ as a balanced choice for accuracy and efficiency.

C. Inference Efficiency

Comparison with inference using full KV cache. We evaluate the inference latency of ClusterKV under various KV cache budgets and compare it against the full KV cache configuration. The experiments are conducted with prompt lengths (P) ranging from 8k to 32k and decoding lengths (D) ranging from 256 to 1024. As shown in Fig. 12, ClusterKV achieves significant efficiency improvements. For $P = 32k$ and $D = 1024$, the speedup in latency is 2× with a budget of 1024 tokens, respectively. Additionally, the decoding throughput increases by up to 2.5×. We also analyze the time spent during prefill. As shown, the clustering overhead of ClusterKV

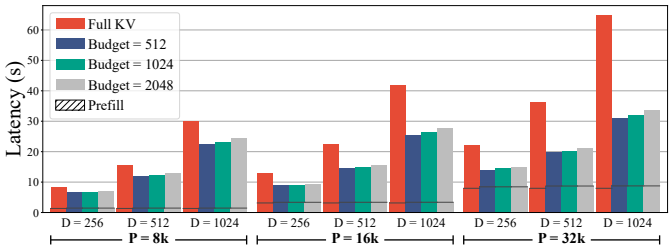


Fig. 12: Comparison of inference latency between ClusterKV and the full KV cache configuration.

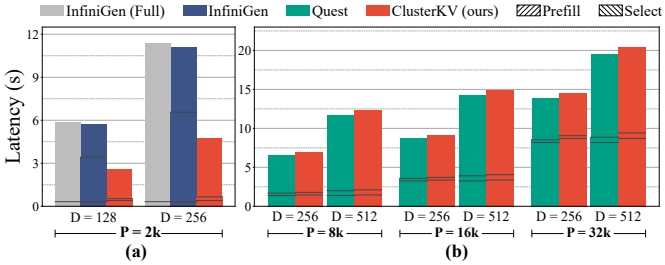


Fig. 13: Comparison of inference latency: (a) ClusterKV vs. InfiniGen with a budget of 256 tokens, and (b) ClusterKV vs. Quest with a budget of 1k tokens.

is minimal, accounting for only 6% to 8% of prefill and less than 2% of the total inference time.

Comparison with SoTA compression methods. InfiniGen is implemented based on FlexGen [23], which exclusively supports OPT [24] models with a 2k context window from scratch and is hard to adapt to other models, so we compare ClusterKV with InfiniGen using OPT-6.7B, as shown in Fig. 13a. ClusterKV achieves an average speedup of 2.3×. InfiniGen's latency is comparable to that of inference using the full KV cache, due to the high computational cost of its per-token selection, while ClusterKV's selection overhead accounts for only about 5% of the total decoding latency.

We present the comparison between ClusterKV and Quest in Fig. 13b, using Llama-3.1-8B model with a budget of 1k tokens. As shown, ClusterKV achieves performance very close to Quest, with latency deviations up to 5%, while ClusterKV delivers significantly higher model accuracy.

Effectiveness of caching. We analyze the hit rates of our cluster-granularity cache during inference, using a 32k-tokens sample from the NarrativeQA dataset. The average hit rates are 63% and 74% for $R = 1$ and $R = 2$, respectively. Compared to directly loading from CPU memory, the caching mechanism improves decoding throughput by 2.3× and 3×, respectively.

VI. Conclusion

This paper introduces ClusterKV, achieving efficient and accurate KV cache compression, recalling tokens at the granularity of semantic clusters. ClusterKV preserves model accuracy while significantly enhancing inference efficiency.

VII. Acknowledgment

This work is sponsored by the National Natural Science Foundation of China (62472273, 62232015).

REFERENCES

[1] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, and J. Li, "Longbench: A bilingual, multitask benchmark for long context understanding," *arXiv preprint arXiv:2308.14508*, 2023.

[2] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "SWE-bench: Can language models resolve real-world github issues?" in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=VTF8yNQM66

[3] OpenAI, "Openai o1 system card," 2024. [Online]. Available: https://cdn.openai.com/o1-system-card-20240917.pdf

[4] Meta, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[5] D. Li, R. Shao, A. Xie, Y. Sheng, L. Zheng, E. Gonzalez, I. Stoica, X. Ma, and H. Zhang, "How long can open-source llms truly promise on context length?" June 2023. [Online]. Available: https://lmsys.org/blog/2023-06-29-longchat

[6] OpenAI, "Gpt-4o system card," 2024. [Online]. Available: https://cdn.openai.com/gpt-4o-system-card.pdf

[7] T. GLM, A. Zeng, B. Xu, B. Wang, C. Zhang, D. Yin, D. Rojas, G. Feng, H. Zhao, H. Lai, H. Yu, H. Wang, J. Sun, J. Zhang, J. Cheng, J. Gui, J. Tang, J. Zhang, J. Li, L. Zhao, L. Wu, L. Zhong, M. Liu, M. Huang, P. Zhang, Q. Zheng, R. Lu, S. Duan, S. Zhang, S. Cao, S. Yang, W. L. Tam, W. Zhao, X. Liu, X. Xia, X. Zhang, X. Gu, X. Lv, X. Liu, X. Liu, X. Yang, X. Song, X. Zhang, Y. An, Y. Xu, Y. Niu, Y. Yang, Y. Li, Y. Bai, Y. Dong, Z. Qi, Z. Wang, Z. Yang, Z. Du, Z. Hou, and Z. Wang, "Chatglm: A family of large language models from glm-130b to glm-4 all tools," 2024.

[8] P. Patel, E. Choukse, C. Zhang, A. Shah, I. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *ISCA*, June 2024.

[9] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *ICLR*, 2024.

[10] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. A. Wang, and B. Chen, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 34 661–34 710.

[11] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "Snapkv: Llm knows what you are looking for before generation," 2024. [Online]. Available: https://arxiv.org/abs/2404.14469

[12] M. Adnan, A. Arunkumar, G. Jain, P. J. Nair, I. Soloveychik, and P. Kamath, "Keyformer: Kv cache reduction through key tokens selection for efficient generative inference," *MLSys*, 2024.

[13] Y. Zhao, D. Wu, and J. Wang, "Alisa: Accelerating large language model inference via sparsity-aware kv caching," 2024. [Online]. Available: https://arxiv.org/abs/2403.17312

[14] Z. Ning, J. Zhao, Q. Jin, W. Ding, and M. Guo, "Inf-mllm: Efficient streaming inference of multimodal large language models on a single gpu," 2024. [Online]. Available: https://arxiv.org/abs/2409.09086

[15] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," *ICML*, 2024.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[17] K. Ataallah, C. Gou, E. Abdelrahman, K. Pahwa, J. Ding, and M. Elhoseiny, "Infinibench: A comprehensive benchmark for large multimodal models in very long video understanding," 2024. [Online]. Available: https://arxiv.org/abs/2406.19875

[18] W. Lee, J. Lee, J. Seo, and J. Sim, "InfiniGen: Efficient generative inference of large language models with dynamic KV cache management," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 155–172. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/lee

[19] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "Kivi: A tuning-free asymmetric 2bit quantization for kv cache," *ICML*, 2024.

[20] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025522014633

[21] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 929–947. [Online]. Available: https://doi.org/10.1145/3620665.3640366

[22] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, "Compressive transformers for long-range sequence modelling," 2019. [Online]. Available: https://arxiv.org/abs/1911.05507

[23] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, Z. Xie, B. Chen, C. Barrett, J. E. Gonzalez, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," 2023. [Online]. Available: https://arxiv.org/abs/2303.06865

[24] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022. [Online]. Available: https://arxiv.org/abs/2205.01068

Harnessing Conventional Video Processing Insights for Emerging 3D Video Generation Models: A Comprehensive Attention-aware Way

Tianlang Zhao*[1], Jun Liu*[1,2], Xingyang Li*[1],
Li Ding[1], Jinhao Li[1,3], Shuaiheng Li[1], Jinbo Hu[1], Guohao Dai[1,2,3†]
[1]Shanghai Jiao Tong University [2]Infinigence-AI [3]SII
† Corresponding author: daiguohao@sjtu.edu.cn

***Abstract*—Video Generation Models based on 3D full attention (3D-VGMs) have significantly enhanced video quality. However, their inference overhead remains substantial, primarily due to the high computational cost of the attention mechanism, which accounts for over 75% of computations. Inspired by the success of conventional video processing, where video compression exploits similarities among patches, we point out that *the attention mechanism can also harness the benefits from similarities among tokens*. Nonetheless, two critical problems arise: (1) How can similarities be efficiently acquired in real-time? (2) How can workload balance be maintained when similar tokens are randomly distributed?**

To address these problems and leverage similarities for 3D-VGMs, we propose SIMPICKER, a comprehensive attention-aware algorithm-hardware co-design for 3D-VGMs. Our core methodology is to fully utilize similarities in attention through both coarse-grained and fine-grained approaches while adopting dynamic adaptive strategies to leverage them. *From the algorithm perspective*, we propose a speculation-based similarity exploitation algorithm, allowing real-time importance speculation on the frame level, which is coarse-grained, and the token level, which is fine-grained. *From the micro-architecture perspective*, we propose a buffered lookup table-based (LUT-based) multiplication architecture for FP-INT multiplication and further eliminate potential bank conflicts to accelerate unimportant attention computation. *From the mapping perspective*, SIMPICKER proposes an adaptive grouping strategy in speculation to tame workload imbalance caused by randomly distributed similar tokens and allow seamless integration of our algorithms. Extensive experiments show that SIMPICKER achieves an average of 5.21×, 1.45× speedup and 17.92×, 1.63× energy efficiency compared to the NVIDIA A100 GPU and the state-of-the-art accelerators.

I. INTRODUCTION

Video Generation Models (VGMs) [1]–[3] have successfully created photorealistic videos using diffusion models. These models show a strong understanding of physical laws [4], marking a milestone towards advanced world models [5]. Diffusion transformer-based models [6]–[8] have shown superior performance in video generation tasks, tackling efficiency and scalability problems that challenge traditional UNet-based structures [9], [10]. The original 2D+1D attention VGMs (2D+1D VGMs) [7], [8], [11] handle spatial and temporal dimensions separately while resulting in temporal inconsistency of videos [12]. 3D-VGMs [12]–[15] are more powerful alternatives which integrate spatial and temporal information, producing higher-quality videos with better consistency.

Despite their superior performance, 3D-VGMs demand significantly more computational resources. Fig. 1 (a) illustrates the computation gap between 2D+1D VGMs and 3D-VGMs, showing that 3D-VGMs have 2×, 4× computation compared to their 2D+1D counterparts in medium-length videos and long videos. Additionally, the primary computational bottleneck shifts to the attention, which consumes over 75% of aggregate computations. The data is evaluated on CogVideoX [12] and Latte1 [8], aligning all the parameters.

Video processing technologies [16]–[18] have achieved remarkable effects due to their ability to leverage spatial and temporal similarities between patches. For instance, Discrete Cosine Transform (DCT) [19] exploits similarities between patches by converting spatial data into the frequency domain with the most energy concentrated in low-frequency components, significantly reducing high-frequency details and minimizing data redundancy. Guided by this, **we can harness the benefits from similarities among tokens for 3D-VGMs**.

However, two challenges hinder their potential for accelerating the inference of 3D-VGMs. First, **how can similarity be efficiently acquired in real time?** Existing works acquire approximate maps of attention importance via low-bit matrix multiplication [20], [21], but this approach results in significant accuracy loss of over 5% while offering nearly no acceleration in 3D-VGMs. Second, **how can workload balance be maintained when similar tokens are randomly distributed?** Although many works [22]–[24] have reused one token's results for other similar tokens based on similarities, they incur heavy workload imbalance due to randomly distributed similar tokens (computing utilization $<$37%).

To tackle the above challenges, we propose SIMPICKER, a comprehensive attention-aware algorithm-hardware co-design. Our core methodology is to fully utilize similarities in attention by both fine-grained and coarse-grained approaches and adopt dynamic and adaptive strategies to tap their full potential. The contributions can be summarized as follows:

- We introduce a **speculation-based similarity exploitation** algorithm to exploit similarities on the frame/token level. We identify the important parts in attention by pre-

*Equal Contribution.

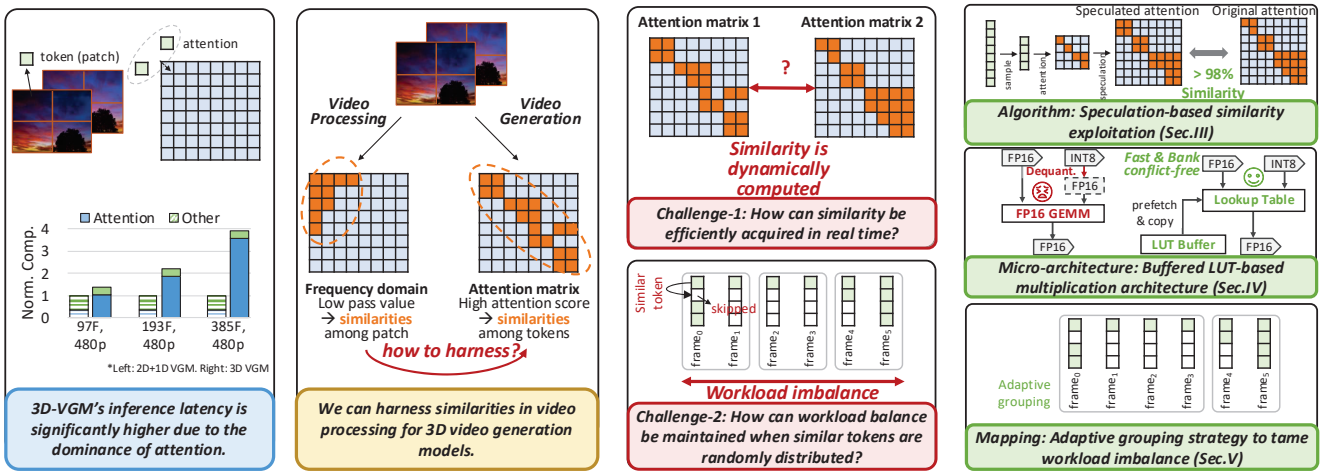


Fig. 1: Overview of SIMPICKER, a comprehensive attention-aware algorithm-hardware co-design in 3D-VGMs.

computing a small frame-level importance map so we can perform mixed-precision multiplication for unimportant parts. Also, we speculate on the input tokens and leverage their similarities to reuse results. By our speculation-based similarity exploitation algorithm, SIMPICKER can reduce over 65% FP-FP multiplications to FP-INT multiplications in attention computation, thus achieving 2.01× speedup compared with the original VGM.

- We propose a **buffered lookup table-based(LUT-based) multiplication architecture**, SIMCORE, to provide support for our algorithm. SIMCORE supports FP-INT multiplication and eliminates the bank conflicts during parallel accesses by a prefetching strategy of LUT Buffer. Our architecture achieves 1.45× speedup compared to naive design without buffered LUT.
- We propose an **adaptive grouping strategy** to tame workload imbalance introduced by randomly distributed similar tokens, allowing seamless integration between our frame-level and token-level speculations, accelerating the model by 1.10× on average.

Extensive experiments on various models and datasets demonstrate that SIMPICKER achieves an average of 5.21×, 1.45× speedup and 17.92×, 1.63× energy efficiency compared to the NVIDIA A100 GPU, and the SOTA accelerators.

II. BACKGROUND AND MOTIVATION

A. Diffusion Transformer and 3D Full Attention

3D-VGMs commonly utilize DiT as their backbone for its scalability and performance. Fig. 2 illustrates 3D-VGMs' structure and the noise prediction process, which takes CogVideoX [12] as an example. At a specific diffusion timestep [25], the input comprises a partially denoised video and prompt, both of which undergo tokenization and embedding to form a continuous token sequence $X^{2\times TN\times C}$. Here, 2 corresponds to the batch size, T denotes the number of video frames, N represents the number of tokens per frame, and C

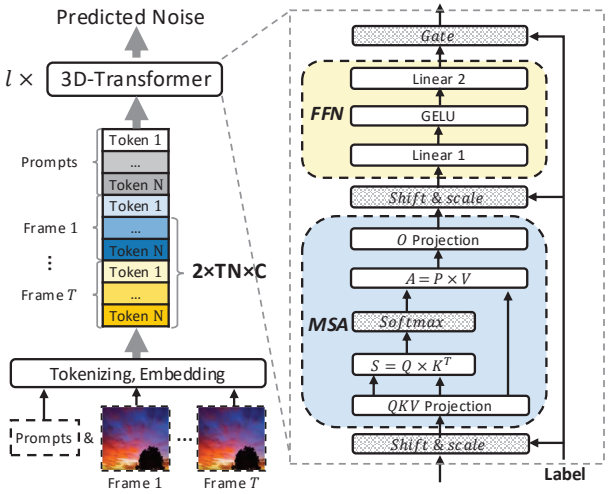


Fig. 2: 3D-VGM's structure and the noise prediction process.

indicates the hidden dimension. The token sequence is then processed through l layers of DiTs, which compute attention across all tokens from different frames.

A DiT includes a multi-head self-attention (MSA), a feed-forward network (FFN), and label embedding operators. The computing flow of MSA can be expressed by $QKV = XW_{QKV}$, $S = QK^T$, $P = Softmax(\frac{S}{\sqrt{d}})$, and $A = PV$.

B. Motivation

As illustrated in Fig. 1 (a), the primary bottleneck lies within the attention computation, which constitutes over 75% of the total computational amount. This finding prompts us to mitigate the overhead associated with the attention mechanism. Drawing inspiration from successful video processing techniques, our approach exploits the similarities and speculates importance at the frame and token levels. Also, we need dynamic and adaptive strategies to tackle the workload imbalance problems caused by similarity exploitation and fully utilize our integrated framework.

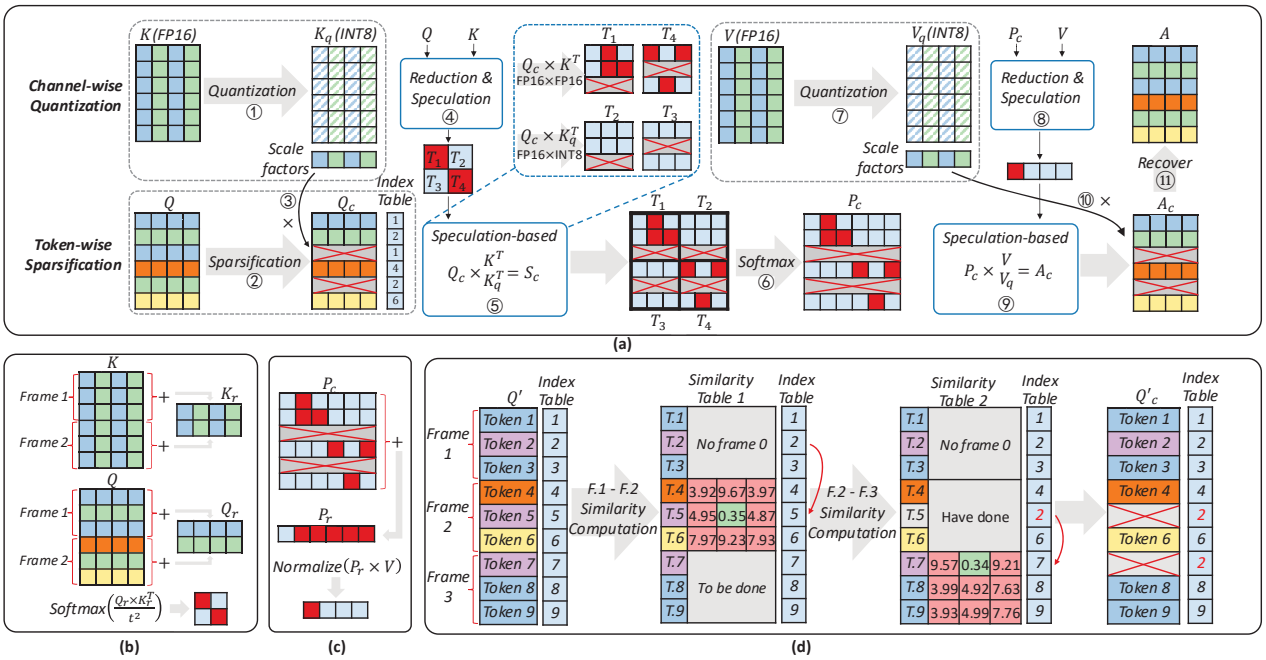


Fig. 3: (a) Overview of the algorithm. We speculate token-level similarities in Q by computing similarities and employ a speculation-based matrix multiplication algorithm for attention. (b) Speculation method for computing QK^T. (c) Speculation method for computing PV. (d) A simple example of temporal token-level similarity exploitation.

III. Speculation-based Similarity Exploitation

A. Algorithm Overview

Fig. 3 demonstrates the overview of our speculation-based similarity exploitation approach. We exploit similarities and speculate important attention parts by pre-computing a small frame-level importance map. Besides, we exploit similarities in the input tokens of the query matrix Q, thereby sparsifying it to accelerate the following computations.

B. Speculation-based Tile-level Similarity Exploitation

Importance Speculation in Attention. Guided by the observation that a small fraction of tiles holds most attention scores, we can perform importance speculation to accelerate attention computation. We aim to speculate $Z = XY$ at the tile granularity of $h \times w$, where Z can be P or A matrix in attention. We first reduce X by computing the sum over every h consecutive rows and Y over every w consecutive columns, generating the reduced matrices X_r and Y_r. We then compute the reduced product $Z_r = X_r Y_r$ and process Z_r row by row, greedily selecting the largest values until their sum exceeds a pre-defined proportion of the aggregated sum. This provides an importance map Z_{pred}, an indicator of importance for the actual computation of $Z = XY$. Fig. 3 (b) and Fig. 3 (c) illustrate the reduction and speculation process for $P = Softmax(\frac{QK^T}{\sqrt{d}})$ and $A = PV$. The ideal granularity for P's computation is $N \times N$. For A, the optimal tile size is $M \times 1$, where M is the total number of tokens.

The overhead of speculation relative to original matrix multiplication is approximately $\frac{M+N^2}{NM}$ for P and $\frac{M+d}{Md}$ for A. For the normal settings of 480p, 96-frame videos for CogVideoX, they are 3.92% and 1.57%, respectively, which are negligible. Despite some slight differences between our speculated importance map for P and P's actual importance map introduced by the $Softmax$ operator, they incur nearly no accuracy loss as most parts of them are the same.

Speculation-based Attention Computation. We quantize the K, V matrices channel-wise with symmetric 8-bit quantization for their channel-wise distribution pattern, obtaining scale factors and the quantized forms K_q and V_q. We do not quantize Q, as our token-level similarity exploitation sparsifies its structure. During the computation of QK^T and PV, we choose to perform FP-INT multiplication for unimportant parts and FP-FP multiplication for the rest, based on the frame-level importance map acquired during the aforementioned speculation. Thus, we significantly reduce over 65% FP-FP multiplications to FP-INT multiplications.

C. Speculation-based Token-level Similarity Exploitation

We exploit Q's token-level similarity to reduce computations in row dimension by speculating that tokens with similar values also have similar outputs after attention. We adopt such speculation in temporal and spatial dimensions in every layer, allowing full utilization of token-level similarity.

At the temporal dimension, the token-level similarity exploitation is performed frame by frame. We set t as a pre-defined window length, and for every token we process, we choose a $t \times t$ window in the previous frame and perform a similarity check with every token inside. We use the L2 norm to represent the similarity between two tokens. Among the $t \times t$ similarity values, we compare the minimum value with a pre-defined threshold τ. The computation of this token is reduced

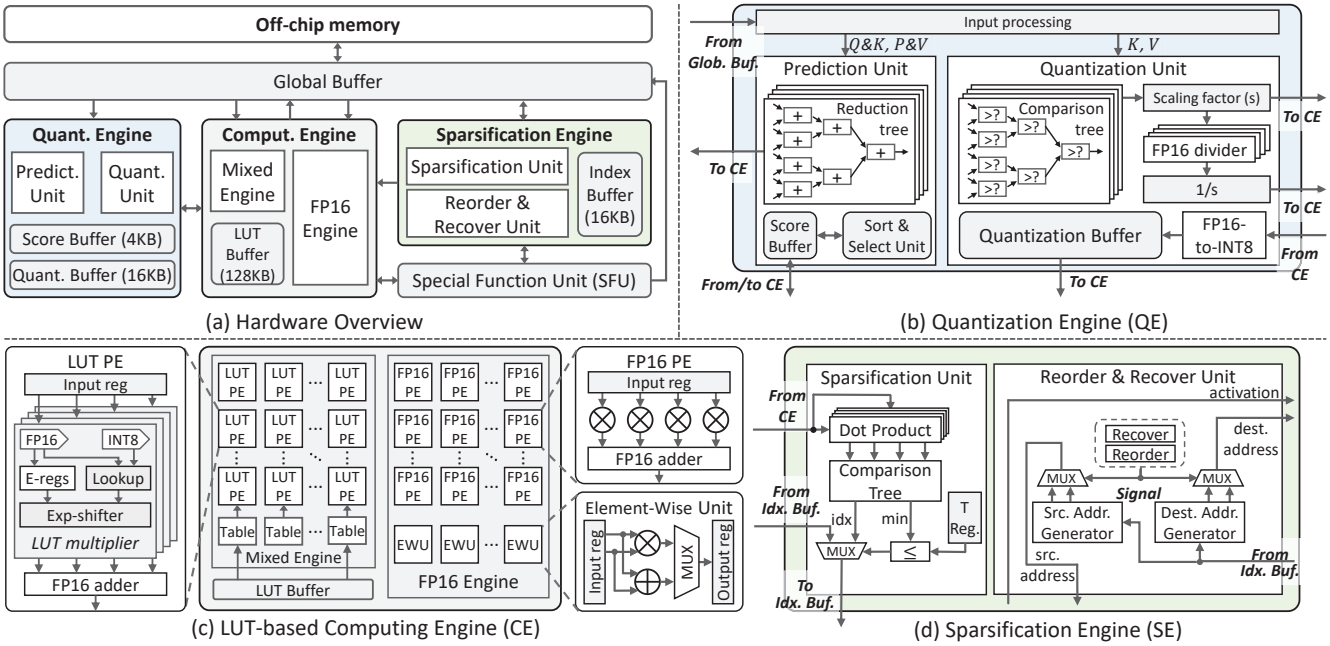


Fig. 4: (a) Our hardware architecture, SIMCORE, features (c) a LUT-based Computing Engine, and units that support our algorithm: (b) Quantization Engine, (d) Sparsification Engine

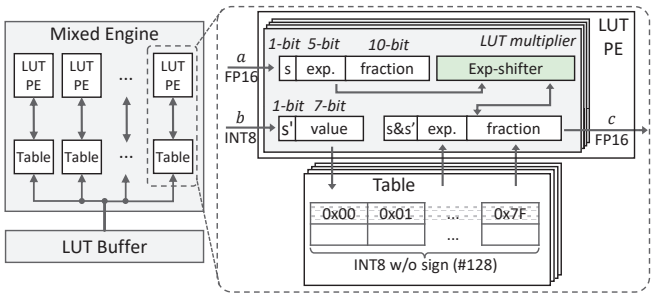


Fig. 5: The LUT-based multiplier. Each LUT multiplier is paired with a local table that prefetches and copies the intermediate results for multiplication, eliminating bank conflicts.

if the minimum value is less than τ. Fig. 3 (d) illustrates our temporal similarity exploitation.

In the spatial domain, we exploit token-level similarity by grouping tokens in each frame. We choose a reference token within each group and compute the similarity between every other token and the reference token in this group. For every non-reference token, we similarly check the threshold and adopt the result of the reference token if it is met.

IV. BUFFERED LUT-BASED MULTIPLICATION ARCHITECTURE

A. Overview of SIMCORE

We present SIMCORE, an end-to-end 3D-VGM accelerator featuring a buffered LUT-based Computing Engine that accelerates the computation of unimportant attention. SIMCORE also includes additional architectural supports for the importance speculation in attention and the token-level similarity exploitation. Fig. 4 shows our hardware overview.

B. LUT-based Computing Engine

The LUT-based Computing Engine (LUT-based CE), as illustrated in Fig. 4 (c), includes an FP16 Engine that manages FP-FP computations and a Mixed Engine that handles FP-INT multiplication with a LUT Buffer. The FP16 Engine deploys Element-Wise Units (EWU) to perform element-wise computations for quantization and sparsification. It also contains FP16 PEs that handle FP-FP matrix multiplication.

The Mixed Engine contains an array of LUT PEs, as depicted in Fig. 5. We employ the observation that the products between the 7-bit INT8 magnitudes and the 10-bit FP16 fractions can be precomputed, and they only occupy 128KB on-chip buffer size. By storing these precomputed products in the LUT Buffer, the FP-INT multiplication can be simplified into one lookup in LUT and one shift in the exponent, thus significantly reducing computations.

However, bank conflicts must be solved since different LUT PEs may require the same intermediate results, which stalls our LUT-based CE and deteriorates efficiency. Our solution utilizes a group of local tables for each PE, whose multipliers share the same FP16 operands. Before each FP-INT MAC, we prefetch intermediate results by copying the table from the LUT Buffer to each PE's local tables. We leverage double-buffer trick so that the cost of our prefetch strategy can be hidden by the computation time of the multiplier.

C. Additional Architectural Support

Shown in Fig. 4 (b), the Quantization Engine is mainly composed of a Prediction Unit (PU) and a Quantization Unit (QU). The PU aggregates input matrices via a Reduction Tree and multiplies the reduced matrices. It sorts the matrix, greedily selects the highest values with its Sort&Select Unit(S&SU), and stores the importance map in the Score Buffer. The

QU quantizes K and V into the 8-bit one's complement representation, where the last 7 bits represent magnitude.

Fig. 4 (d) shows the Sparsification Engine, which is composed of a Sparsification Unit (SU), a Reorder & Recover Unit (R&RU), and an Index Buffer. After the query matrix Q is generated, the SU exploits token-level similarity by computing the similarities and letting the Comparison Tree find the highest similarity, which is compared to the threshold in the T register. If the condition is satisfied, the Index Buffer will be updated. The R&RU reorders Q to sparsified Q_c and recover A from sparsified A_c. For both reordering and recovering, We traverse the index table, generating each index's source and destination addresses. Then, the data will be transferred according to the index and the addresses.

V. Adaptive Grouping Strategy

A. Algorithm-Hardware Mapping Overview

We describe the MSA dataflow on SimCore. Other phases in 3D-VGMs primarily involve FP-FP operations and element-wise operations and do not use the devised engines.

Initially, Q and K are reduced, and the importance map for P is generated in the Prediction Unit with the assistance of the Special Function Unit (SFU) and S&SU. Concurrently, the SU and QU produce sparsified Q_c and quantized K_q, respectively. The CE then computes S and P row-by-row, leveraging both the FP16 Engine and Mixed Engine, guided by the importance map of P. This row-by-row computation maximizes parallelism and enables the fusion of $P = Softmax\left(\frac{S}{\sqrt{d}}\right)$ and $A = PV$. As every t rows of S are computed in parallel, they are forwarded to the SFU to generate P. This row slice of P is sent to the QE to obtain the importance map for A. Here, we employ an adaptive grouping strategy for P (Sec. V-B). Subsequently, V columns are reordered and grouped based on the importance map for optimized resource utilization. Upon completing the full attention process, the Recovery Unit in the SE restores the A_c matrix to its original form.

B. Adaptive Grouping Strategy to Tame Workload Imbalance

Randomly distributed similar tokens lead to workload imbalances in the sparsified Q_c matrix. For instance, during the attention computation phase, the workload of each PE is imbalanced, necessitating the compression of Q_c to utilize computational resources fully. Such imbalance also occurs in the importance map speculation for A, where we need to recover P_c into P for speculation fully, and the distribution of P_c's indices is random and uneven. Moreover, the recovery incurs significant memory overhead due to the separation of similar token pairs in different frames.

We propose an adaptive grouping strategy to tame workload imbalance and integrate our frame-level and token-level speculations seamlessly. Instead of recovering P from P_c, we directly reduce P_c for subsequent speculation, as our observations show that the value distribution in each column of A in even, allowing such a strategy with no accuracy loss.

TABLE I: Accuracy of SimPicker in 3D-VGMs.

Component	Method	CLIPSIM ↑	VBench ↑
Open-Sora-Plan v1.2	Baseline	0.196	0.305
	SimPicker (Ours)	0.196	0.305
CogVideoX-5B	Baseline	0.304	0.311
	SimPicker (Ours)	0.307	0.312

VI. Evaluation

A. Experimental Setup

Hardware Setup. We implement SimCore using Verilog. We set CE comprising 16×8 LUT PEs with 8 LUT multipliers in each, 8×8 FP16 PEs with 8 multipliers in each, and 4×8 EWUs. The configurations of the QE and SE are aligned with CE's parallelism, enabling pipelined operator processing. The on-chip buffer includes a 208KB Global Buffer, 256KB Lookup Table, 128KB Lookup Table buffer, 16KB Index buffer, 16KB Quantization buffer, and 4KB Score buffer.

The area and power of logic units are evaluated at 1 GHz frequency using Synopsys Design Compiler under a 32nm process. Cacti 7 [26] are used for on-chip buffer evaluations.

Models and Datasets. We evaluate SimPicker on two SOTA 3D-VGMs, CogVideoX-5B [12] and Open-Sora-Plan v1.2 [14]. CogVideoX-5B concatenates text prompt tokens and video tokens together without MCA, while Open-Sora-Plan v1.2 still uses cross-attention to capture the correlation between prompt and video. We evaluate the impact of SimPicker on model accuracy on the VBench dataset [27], a carefully designed benchmark containing comprehensive prompts that describe the scene in detail.

Tasks and Metrics. We evaluate the quality of videos generated by SimPicker with two metrics, CLIPSIM [28] and the evaluation model in VBench [27]. CLIPSIM employs CLIP [28] to evaluate the similarity between the text prompt and the generated video. The evaluation metric of VBench gives a comprehensive analysis from multiple perspectives.

Baselines. To compare the performance and energy efficiency of SimPicker with SOTA works, we run open-sourced 3D-VGMs on the NVIDIA A100 GPU [29]. The inference latency and power of A100 are measured by python time library [30] and CUDA event API [31]. We also compare several SOTA ASIC accelerators, including two DiT-based works (InterArch [22] and CMC [32]) and two attention-based works (Sanger [20] and FACT [21])uu. For a fair comparison, we unify the number of FP16 MACs, on-chip buffer capacity, and frequency with those of SimCore. The off-chip bandwidth is set to 128GB/s. We design and implement a cycle-level behavioral simulator to model the operational execution of different accelerators. The CODEC auxiliary module in CMC is excluded from the evaluation.

B. Evaluation of Model Accuracy

We choose the original models with FP16 precision as the baseline. For SimPicker, the token-level similarity threshold is set to 2.5, while the proportion of the aggregated sum of important tiles in speculation is 0.65. Table. I shows that SimPicker has no accuracy loss on both metrics.

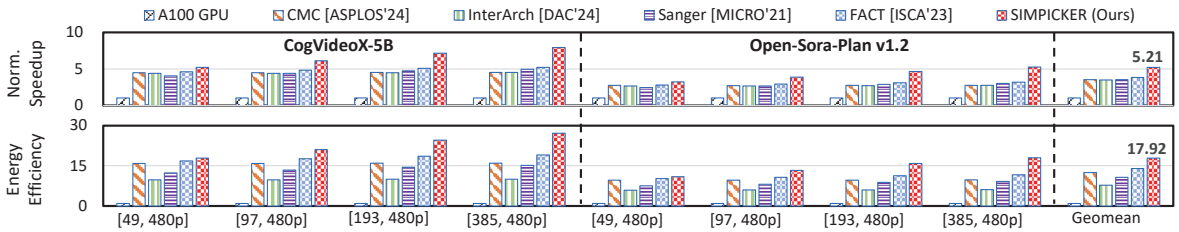


Fig. 6: Overall Performace of SIMPICKER. The horizontal axis represents [frames, resolution] of the generated video.

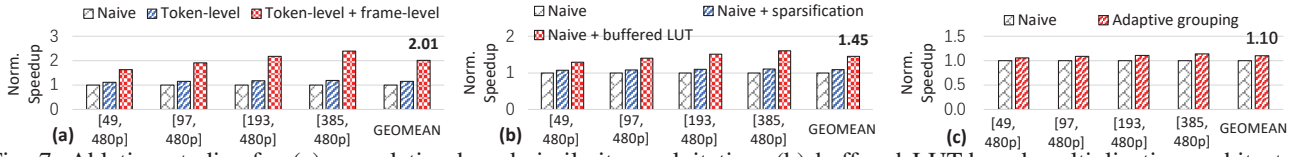


Fig. 7: Ablation studies for (a) speculation-based similarity exploitation, (b) buffered LUT-based multiplication architecture, and (c) adaptive grouping strategy. The horizontal axis represents [frames, resolution].

TABLE II: The area and power of SIMCORE.

Component		Area (mm^2)	Power (mW)
Computing Engine	16×8 LUT PEs	1.23	47.10
	8×8 FP16 PEs	2.71	323.07
	4×8 EWUs	0.04	3.58
Quantization Engine	Prediction Unit	0.02	1.40
	Quantization Unit	0.03	1.89
Spasification Engine	Sparsification Unit	0.14	16.36
	Reorder and Recovery Unit	0.06	2.69
Special Function Unit		0.23	11.02
628KB On-chip buffer		1.47	61.67
Total (32nm)		5.93	468.79

C. Hardware Analysis

The area and power of SIMCORE are shown in Table. II. The total area of SIMCORE is 5.93 mm^2, and the power is 468.79 mW. The Computing Engine is the main module, accounting for the largest proportion of 67%. The additional Quantization Engine and Sparsification Engine are only 0.25 mm^2, accounting for 4.21% of the total area.

D. Overall Performance

For both models, we evaluate the end-to-end latency and the power consumption among NVIDIA A100 GPU [29] and the ASIC accelerators under the same prompt and workload.

Performance with GPUs. For NVIDIA A100 GPU, SIMPICKER achieves 5.21× speedup on average due to our-buffered LUT-based FP-INT multiplication architecture that efficiently supports mixed-precision multiplication. Besides, SIMPICKER also achieves an average of 17.92× energy efficiency since our adaptive grouping strategy fully utilizes the hardware computing resources. The gap between SIMPICKER and GPU further expands when the video length increases, showing that SIMPICKER is also suitable for scalable settings.

Performance with ASIC accelerators For the SOTA ASIC accelerators, our SIMPICKER achieves an average 1.49× speedup and 1.87× energy efficiency compared to DiT-based accelerators CMC and InterArch. For attention-based accelerators, we get an average 1.41× speedup and 1.33× energy efficiency for Sanger and FACT.

E. Ablation Study

Speculation-based similarity exploitation algorithm. We test SIMPICKER's end-to-end latency with a naive algorithm and the algorithm that only exploits token-level similarity in our architecture under the setting of no accuracy loss. As shown in Fig. 7 (a), SIMPICKER increases the speedup by an average of 2.01×, which shows that our algorithm can effectively exploit the similarity in both frame-level and token-level. To be mentioned here, GPU cannot fully support our algorithm since there's an FP-INT multiplication.

Buffered LUT-based multiplication architecture. Experiments are taken to evaluate the end-to-end latency of our model compared to the naive one and a naive one equipped with a sparsification engine. The naive architecture only has an FP16 Engine and a Quantization Engine. As shown in Fig. 7 (b), such design significantly impacts the end-to-end latency of 1.45× on average, which indicates that our architecture can effectively support our algorithm.

Adaptive grouping strategy. As illustrated in Fig. 7 (c), we evaluate SIMPICKER's latency and breakdown with and without the adaptive reduction strategy. Results show the adaptive grouping strategy achieves a 1.10× speedup compared to a workload-imbalanced one. That's because token-level similarities are harder to exploit in 3D-VGMs due to their capabilities of generating more continuous video than their 2D+1D counterparts, leading to a low ratio of similar tokens.

VII. CONCLUSION

In this paper, we propose SIMPICKER, a comprehensive attention-aware algorithm-hardware co-design for 3D-VGMs inspired by video processing methods. SIMPICKER achieves 5.21×, 1.45× speedup and 17.92×, 1.63× energy efficiency compared to NVIDIA A100 GPU and SOTA accelerators. We hope that more works can be inspired and existing methods in video processing can be harnessed to boost video generation.

VIII. ACKNOWLEDGEMENT

This work was sponsored by the National Natural Science Foundation of China (No. U21B2031), Shanghai Rising-Star Program (No. 24QB2706200).

REFERENCES

[1] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, *et al.*, "Video generation models as world simulators," 2024.

[2] A. Blattmann, T. Dockhorn, S. Kulal, D. Mendelevitch, M. Kilian, D. Lorenz, Y. Levi, Z. English, V. Voleti, A. Letts, *et al.*, "Stable video diffusion: Scaling latent video diffusion models to large datasets," *arXiv preprint arXiv:2311.15127*, 2023.

[3] W. Hong, M. Ding, W. Zheng, X. Liu, and J. Tang, "Cogvideo: Large-scale pretraining for text-to-video generation via transformers," *arXiv preprint arXiv:2205.15868*, 2022.

[4] B. Kang, Y. Yue, R. Lu, Z. Lin, Y. Zhao, K. Wang, G. Huang, and J. Feng, "How far is video generation from world model: A physical law perspective," 2024.

[5] S. Yang, J. Walker, J. Parker-Holder, Y. Du, J. Bruce, A. Barreto, P. Abbeel, and D. Schuurmans, "Video as the new language for real-world decision making," *arXiv preprint arXiv:2402.17139*, 2024.

[6] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182, 2023.

[7] Z. Zheng, X. Peng, T. Yang, C. Shen, S. Li, H. Liu, Y. Zhou, T. Li, and Y. You, "Open-sora: Democratizing efficient video production for all," March 2024.

[8] X. Ma, Y. Wang, G. Jia, X. Chen, Z. Liu, Y.-F. Li, C. Chen, and Y. Qiao, "Latte: Latent diffusion transformer for video generation," *arXiv preprint arXiv:2401.03048*, 2024.

[9] S. Yu, K. Sohn, S. Kim, and J. Shin, "Video probabilistic diffusion models in projected latent space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[10] K. Kim, H. Lee, J. Park, S. Kim, K. Lee, S. Kim, and J. Yoo, "Hybrid video diffusion models with 2d triplane and 3d wavelet representation," *arXiv preprint arXiv:2402.13729*, 2024.

[11] O. Bar-Tal, H. Chefer, O. Tov, C. Herrmann, R. Paiss, S. Zada, A. Ephrat, J. Hur, G. Liu, A. Raj, *et al.*, "Lumiere: A space-time diffusion model for video generation," *arXiv preprint arXiv:2401.12945*, 2024.

[12] Z. Yang, J. Teng, W. Zheng, M. Ding, S. Huang, J. Xu, Y. Yang, W. Hong, X. Zhang, G. Feng, *et al.*, "Cogvideox: Text-to-video diffusion models with an expert transformer," *arXiv preprint arXiv:2408.06072*, 2024.

[13] P. Gao, L. Zhuo, Z. Lin, C. Liu, J. Chen, R. Du, E. Xie, X. Luo, L. Qiu, Y. Zhang, *et al.*, "Lumina-t2x: Transforming text into any modality, resolution, and duration via flow-based large diffusion transformers," *arXiv preprint arXiv:2405.05945*, 2024.

[14] P.-Y. Lab and T. A. etc., "Open-sora-plan," Apr. 2024.

[15] J. Xu, X. Zou, K. Huang, Y. Chen, B. Liu, M. Cheng, X. Shi, and J. Huang, "Easyanimate: A high-performance long video generation method based on transformer architecture," *arXiv preprint arXiv:2405.18991*, 2024.

[16] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[17] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[18] M. M. Alam, T. Nguyen, M. Hagan, and D. Chandler, "A perceptual quantization strategy for hevc based on a convolutional neural network trained on natural images," p. 959918, 09 2015.

[19] Wikipedia contributors, "Discrete cosine transform — Wikipedia, the free encyclopedia," 2024. [Online; accessed 20-November-2024].

[20] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, (New York, NY, USA), p. 977–991, Association for Computing Machinery, 2021.

[21] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[22] X. Wang, Z. Song, and X. Liang, "Interarch: Video transformer acceleration via inter-feature deduplication with cube-based dataflow," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, (New York, NY, USA), Association for Computing Machinery, 2024.

[23] X. Zhao, X. Jin, K. Wang, and Y. You, "Real-time video generation with pyramid attention broadcast," *arXiv preprint arXiv:2408.12588*, 2024.

[24] P. Selvaraju, T. Ding, T. Chen, I. Zharkov, and L. Liang, "Fora: Fast-forward caching in diffusion transformer acceleration," *arXiv preprint arXiv:2407.01425*, 2024.

[25] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[26] A. S. Naveen Muralimanohar and V. Srinivas, "Hewlettpackard/cacti." https://github.com/HewlettPackard/cacti Accessed May 22, 2023.

[27] Z. Huang, Y. He, J. Yu, F. Zhang, C. Si, Y. Jiang, Y. Zhang, T. Wu, Q. Jin, N. Chanpaisit, Y. Wang, X. Chen, L. Wang, D. Lin, Y. Qiao, and Z. Liu, "VBench: Comprehensive benchmark suite for video generative models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[28] C. Wu, L. Huang, Q. Zhang, B. Li, L. Ji, F. Yang, G. Sapiro, and N. Duan, "Godiva: Generating open-domain videos from natural descriptions," *arXiv preprint arXiv:2104.14806*, 2021.

[29] N. Corporation, *NVIDIA A100 Tensor Core GPU Architecture*, 2020. Last accessed on November 14, 2024.

[30] Python., "Python time library.." https://docs.python.org/3/library/time.html.

[31] I. NVIDIA, "Cuda event api. https://docs.nvidia.com/cuda/cuda-runtime-api/index.html," 2024.

[32] Z. Song, C. Qi, F. Liu, N. Jing, and X. Liang, "Cmc: Video transformer acceleration via codec assisted matrix condensing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, (New York, NY, USA), p. 201–215, Association for Computing Machinery, 2024.

FPGA-TrustZone: Security Extension of TrustZone to FPGA for SoC-FPGA Heterogeneous Architecture

Shupeng Wang[1*], Xindong Fan[2*], Xiao Xu[1], Shuchen Wang[1], Lei Ju[2], and Zimeng Zhou[1,2,3†]
[1]School of Cyber Science and Technology, Shandong University
[2]Quan Cheng Laboratory, Jinan, China
[3]State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, 266237, China

***Abstract*—To address the growing security issues faced by ARM-based mobile devices today, TrustZone was adopted to provide a trusted execution environment (TEE) to protect sensitive data. Such TrustZone-based models have been proven to be effective, but they target CPU architectures and do not work for the security of widely used heterogeneous computing platforms such as FPGAs. To solve this issue, we propose a comprehensive SoC-FPGA security framework, FPGA-TrustZone, to support FPGA TEE by extending the security of ARM TrustZone. Experiments on real SoC-FPGA hardware development boards show that FPGA-TrustZone provides high security with low performance overhead.**

***Index Terms*—Trusted execution environment (TEE), SoC-FPGA, ARM TrustZone, security extension**

I. Introduction

In the era of big data, emerging applications like deep learning, intelligent traffic control, and large-scale database management demand information processing capabilities beyond what traditional CPUs can provide. Consequently, new high-performance computing platforms like GPUs, FPGAs, and processing-in-memory chips have emerged [1], [2]. Specifically, FPGAs offer high parallelism, low energy consumption, and programmability [3], making them increasingly popular in the commercial sector. To integrate FPGAs' computing capabilities into traditional CPU-based systems, heterogeneous SoC-FPGA architectures (e.g., Xilinx ZCU102, Xilinx ZCU104, Xilinx Zynq-7010) have been widely deployed in recent years.

Ongoing confidential data leaks have caused significant economic losses and psychological harm, raising widespread concern. At present, data security has become the main concern in the digital age besides performance and energy consumption. Trusted execution environment (TEE) is a trending topic in modern hardware-assisted security architectures for solving data security problems, and it has gained significant attention from both industry and academia. TEEs provide an isolated, protected environment for sensitive data and code, using hardware isolation and encryption to prevent unauthorized access or tampering. A large number of TEE designs implemented on various instruction set architectures including Intel SGX [4], Keystone [5] and ARM TrustZone [6]. ARM TrustZone offers hardware-level protection and isolation, and it is widely used in embedded systems and mobile devices [7], [8].

However, mainstream TEEs protect only CPUs, leaving heterogeneous computing platforms vulnerable. As FPGA-based heterogeneous platforms become more common, they face increasing attacks. Researchs [9]–[11] have explored extending TEE security features to FPGAs, including hardware isolation and secure environments for third-party applications. Therefore, existing solutions face challenges like insufficient security protection and high overhead. This paper proposes FPGA-TrustZone, extending ARM TrustZone's security features to SoC-FPGA heterogeneous architectures, ensuring FPGA resources are protected. The main contributions of this paper are summarized below:

- We propose a TEE for SoC-FPGA, in which it protects sensitive data and code from external attacks with the TrustZone extension.
- To support security isolation between FPGA programs, a comprehensive FPGA security framework, including the FPGA Secure Monitor, Secure Monitor-Extension and BRAM Protector, is designed.
- We implement a system prototype using an SoC-FPGA board for functional verification and performance analysis. The experimental results demonstrate that the proposed framework can guarantee security while the additional cost is acceptable.

The rest of the paper is organized as follows. Section II introduces the background and the related work for TEEs in FPGA. Section III presents the threat models. Section IV describes the proposed system and design details. Section V shows the experimental setup and evaluations. Section VI concludes this paper.

II. Background and Related Work

ARM TrustZone is an ARM architecture technology designed to enhance security by dividing the system into secure and non-secure execution environments (i.e., TEE and REE), widely used in embedded systems and mobile devices. It enhances system security by isolating external and memory resources at the hardware level. TrustZone virtualizes the CPU

This work is supported by Shandong Provincial Natural Science Foundation (Grant No.ZR2024MF084), National Natural Science Foundation of China (Grant No.62432005), Department of Science & Technology of Shandong Province (Grant No. SYS202201), Quan Cheng Laboratory (Grant No. QCLZD202302), Taishan Scholars Program (No. tsqn202211281).

*Shupeng Wang and Xindong Fan are co-first authors of the article.
†Corresponding author: Zimeng Zhou(Email: zhouzimeng@sdu.edu.cn).

core into secure and non-secure states to manage transactions in the Secure and Normal Worlds separately. A high-permission controller, called the secure monitor, controls the operation of these cores. By integrating software and hardware, ARM TrustZone offers a robust security solution for mobile devices and embedded systems, safeguarding sensitive data and critical tasks from attacks, and enhancing user privacy.

To meet diverse application requirements and evolving market demands, SoC-FPGA heterogeneous platforms have emerged. The SoC-FPGA platform integrates FPGA and CPU on a single chip. The integration of FPGAs with CPUs has unlocked new avenues for innovation in fields like data analytics, machine learning, and embedded systems [12], [13]. This integration offers greater flexibility, configurability, and performance across various applications. However, SoC-FPGA also brings additional security risks. Thus, protecting the security of the SoC-FPGA heterogeneous architecture is crucial.

Yanling Wang et al. [14] propose a scheme called RCTEE, offering remote users a runtime-customizable TEE on a cloud SoC-FPGA. This scheme supports secure and dynamic IP deployment and invocation. SGX-FPGA [11] proposes a framework that connects SGX enclaves and FPGA, creating a secure hardware isolation path between the CPU and FPGA to ensure the confidentiality and integrity of heterogeneous CPU-FPGA systems. In SoC-FPGA architecture, Ambassy [9] uses an FPGA terminal to build a secondary TEE for third-party Trusted Applications, extending TrustZone's use. This not only shields trusted applications in the secondary TEE from the Normal World but also protects both secure and normal operating systems from malicious third-party applications. However, the secondary TEE does not address the trusted boot of the FPGA. TEEOD [10] proposes isolation tools to create secure environments for common applications and creates a secure processor for each isolated environment. However, there are different problems such as high overhead, high communication cost, insufficient external storage security and so on when the existing schemes are really applied to the actual applications. This is the challenge we strive to address in this paper.

III. Threat Model

In this work, we target an emerging new system architecture, the SoC-FPGA heterogeneous architecture, which does not have a commonly adopted threat model definition in the community. The integration of FPGAs in SoCs introduces new attack surfaces that need to be considered in TEE systems. These attack surfaces can be categorized into four aspects:

1) *Internal attacks on the FPGA.* Modern FPGAs support concurrent execution for multiple tasks. The concurrent execution introduces a risk of stealing confidential models or data by leveraging internal FPGA resources. And attackers embed malicious logic into FPGA designs, such as through third-party IP or unverified code modules, to disrupt system functionality, steal or leak sensitive data [14].

2) *Using a compromised FPGA to attack CPU-side resources.* FPGAs in mobile devices typically have access permissions to the CPU-side memory, which contains sensitive data. Then attackers can inject malicious logic into the FPGA-side, directly accessing the memory or registers of the CPU-side, thereby stealing or tampering with sensitive data, which is more likely to cause system crashes.

3) *Invaded CPU attacks the FPGA.* Attackers can control the Normal World on the CPU-side through malicious code injection, thereby achieving deep penetration into the entire system, directly affecting the security of the system [10]. Then the invaded CPU can exploit its control over resources to attack the FPGA through malicious code, communication protocol vulnerabilities, etc., illegally accessing FPGA-side resources, changing logic functions, stealing sensitive user data, and disrupting system integrity and user data confidentiality.

4) *External attacks between CPUs and FPGAs.* In the SoC-FPGA system architecture, the AXI bus serves as a critical communication channel between CPUs and FPGAs. This interface plays an important role in coordinating complex operations, but it introduces new vulnerabilities in data security such as corrupting communication signals and response signals [15]. And attackers can use various means to monitor, interfere with, and steal user data transmitted on the AXI bus.

IV. System Design

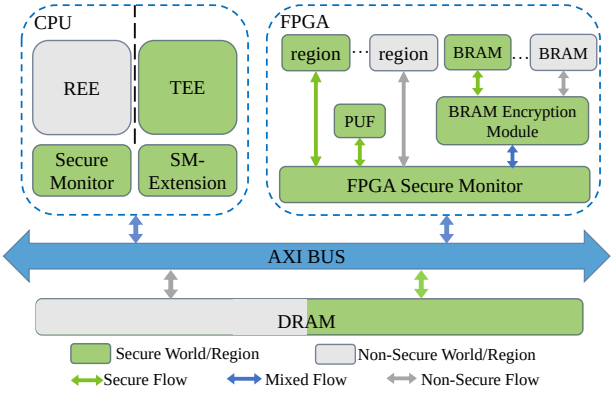


Fig. 1. Overall Framework of FPGA-TrustZone.

We propose FPGA-TrustZone, which builds FPGA TEE by extending the security features of ARM TrustZone to the SoC-FPGA platform. As shown in Fig. 1, FPGA-TrustZone consists of three essential components: *FPGA Secure Monitor* (*FPGA-SM*), *Secure Monitor-Extension* (*SM-Extension*), and *BRAM protector*. We design the *FPGA-SM* to provide a lightweight inner resource isolation mechanism for the FPGA by extending the security features of ARM TrustZone. The *SM-Extension* working on the CPU-side is designed to cooperate with the work of the *FPGA-SM*. *BRAM protector* utilizes the constructed physical unclonable function (*PUF*) to support the *BRAM Encryption Module* to encrypt the BRAM to protect sensitive information from unauthorized access and tampering.

A. Challenges of Protecting Soc-FPGA

To develop the FPGA-TrustZone for the secure SoC-FPGA heterogeneous architecture, we have three challenges to overcome:

- First, to protect the data and code of the FPGA program at runtime, FPGA-TrustZone needs to ensure that the data and code are secure when they exist on the FPGA.
- Second, as FPGA resources are shared by multiple applications, we need to ensure proper security isolation. Specifically, we need to not only enforce security isolation between CPU and FPGA applications, but also the isolation among FPGA applications.
- Third, FPGA resources are limited; therefore, FPGA-TrustZone should be lightweight and not significantly affect the performance of FPGA applications.

In the following sections, we will discuss how we address these challenges in detail.

B. FPGA Secure Monitor

FPGA-SM is the core component of the FPGA-TrustZone in FPGA. In order to ensure the security of the data code of the FPGA program during the lifecycle on the FPGA, FPGA-SM implements three functions: trusted boot, security isolation and secure transmission.

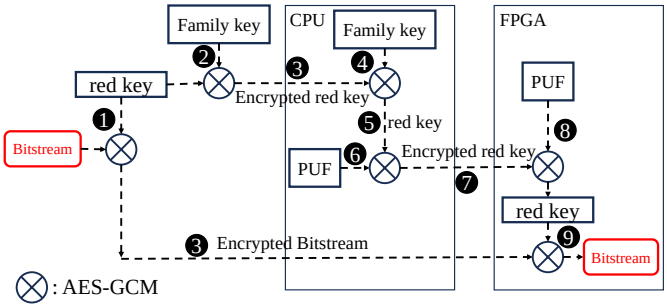


Fig. 2. Trusted Boot of the FPGA.

Trusted Boot. FPGA-TrustZone implements a trusted boot for the FPGA, protecting the system's lifecycle security, including both the startup and runtime. The trusted boot of the FPGA revolves around how to securely transfer Bitstream stored in external memory to the FPGA-side, while also ensuring that adversaries cannot obtain hardware designs or other sensitive information in the Bitstream. Therefore, it is necessary to encrypt the data during the boot process. At the same time, Bitstream also holds sensitive data such as the FPGA-TrustZone encryption keys and other firmware code. So, the data stored in the external memory also needs to be encrypted. Therefore, we designed a trusted boot process as shown in Fig. 2. FPGA-TrustZone uses AES-128bit encryption with Galois/Counter Mode (GCM) for the entire trusted boot process, which defends against replay attacks. First, ❶FPGA-TrustZone encrypts Bitstream and other mirror partitions belonging to CPU using a standards-compliant AES key (called red key). After that, ❷FPGA-TrustZone uses the Family Key interface offered by the Vitis design tools to encrypt the red key. The Family Key is only saved in the on-chip memory in the ARM core. It is a constant AES key value hard-coded into the devices.

The encrypted red key and other ciphertexts are packaged into a boot image and placed in external memory. When CPU boots, ❸the Configuration Security Unit(CSU) in CPU loads the image from external memory and decrypts the red key❹, which is then used to decrypt any other image partitions belonging to CPU except Bitstream❺. After that, ❻FPGA-TrustZone uses the Key-Encryption Key generated by the PUF in the CSU to encrypt the red key and ❼ sends it along with the encrypted Bitstream to the FPGA, where the PUF on the FPGA decrypts the red key❽, and then decrypts the Bitstream with the red key❾, and the FPGA is trusted to run. All Zynq UltraScale+ devices have a built-in physically unclonable function, which can generate a cryptographically strong, device-unique encryption key. This key cannot be read by a user, allowing for a heightened level of key security.

Security Isolation. To enforce security isolation among FPGA applications, FPGA-SM implements security isolation for the shared FPGA resources. Then, FPGA-TrustZone uses FPGA-SM to isolate memory, IP, and other FPGA components, enforcing access restrictions to prevent interference or malicious tampering during runtime, thereby enhancing data security.

After receiving a transaction from CPU requesting the creation of an isolation zone, *FPGA-SM* first determines whether the transaction is secure or non-secure through the AxPROT bit in the transaction, and then creates a secure or non-secure zone (called region). The AxPROT bit can also indicate whether the transaction is privileged and whether the transaction is a data transaction or an instruction transaction. The hardware basis for isolation between the area is the regional identifier (called AxRegion) of the AXI bus protocol. With the identifier, the *FPGA-SM* can manage up to 16 regions, and it can control each region independently. Each region does not affect each other. Using AxPROT to divide FPGA resources into secure and non-secure parts does not achieve isolation between regions. In order to refine isolation to each region, FPGA-TrustZone divides the hardware resources on the SoC-FPGA platform, assigning a DeviceID to each hardware, and transactions issued from that hardware will come with a DeviceID. When creating a region, the *FPGA-SM* creates a device access list belonging to the region. The device list maintains the devices that can access the region. In this way, the FPGA-TrustZone applies isolation to each region.

Secure Transmission. It can be used to perform secure communication with the CPU. In this component, *FPGA-SM* can decrypt the encrypted transactions transmitted from the CPU-side and transmits them to the corresponding region for processing. The transactions on FPGA-side are encrypted and transmitted to CPU-side through AXI bus. And it can provide a dynamic configuration interface to support dynamic configuration of *SM-Extension* at runtime.

In order to check whether a transaction has permission to access the target region, we implement a *Transaction monitor* in *FPGA-SM*. *Transaction monitor* acts as a checkpoint for passing transactions. FPGA-TrustZone specifies that a secure transaction can access its own secure region and all non-secure regions. When performing a security check, *Transaction monitor* first checks whether the security level of a transaction

is the same as that of the target region. If the security level is acceptable and the target region is secure, *Transaction monitor* continues to check whether the DeviceID carried by the transaction is in the device access list of the target region. If all the checks pass, the transaction is forwarded to the target region, else the *Transaction monitor* poisons the transaction and sends an interrupt request (IRQ) to the platform management unit (PMU). Specifically, *Transaction monitor* has two poison methods: 1) Return an error code (default). 2) Modify the destination address of the transaction so that it redirects to an empty area.

C. Secure Monitor-Extension

The implementation of the FPGA-SM is inseparable from the help of the CPU-side, i.e., the *SM-Extension*, which provides the security features of ARM TrustZone, the dynamic configuration of application security requirements, and the handling of secure transmission. *SM-Extension* is the extension of Secure Monitor, the core component of ARM TrustZone, to implement the secure communication between CPU and FPGA. It can ensure the security and credibility of the system.

Dynamically configure. *SM-Extension* proposes multiple interfaces for applications. An application can use these interfaces to create, configure, use, and destroy a FPGA region and it can control the entire life cycle of the region.

Secure Transmission. *SM-Extension* is able to handle secure and non-secure transactions from FPGA to prevent potential security threats and attacks. It is used for encrypting and decrypting transactions transmitted between CPU and FPGA, to ensure the security of data transmission, and minimize the additional overhead of performance. It uses the symmetric key algorithm AES-128bit with Electronic CodeBook mode. The key is stored in the *SM-Extension* and *FPGA-SM* through hardcoding. It can also handle transactions from FPGA. After the FPGA sends the transaction to CPU, SM-Extension first checks its security level, then decrypts the data with the corresponding key and forwards the data to the corresponding applications.

D. BRAM Protector

In order to prevent the leakage of BRAM by malicious means, FPGA-TrustZone proposes *BRAM Protector* to protect sensitive information. BRAM Protector implements the *PUF* to generate the Initialization Vector (IV) values of the AES-CBC algorithm to ensure the uniqueness and randomness of the IV. Then BRAM Protector can encrypt BRAM to protect sensitive information in the system from unauthorized access and tampering.

The PUF is a security feature based on physical properties that generate unique identifiers using small differences in chip manufacturing processes. The PUF can be used for various security applications, including device authentication and key generation, as well as other security fields, providing reliable solutions for various security applications. Here, PUF is used to generate IV values in the AES-CBC algorithm. The IV generated by the PUF can satisfy uniqueness and randomness well. Our design combines Arbiter PUF and RO PUF, including ring oscillators, latches, and arbitrators. The arbiter is responsible for controlling the operational status of the ring oscillator and latch, while the ring oscillator and latch are responsible for generating the output data of the PUF.

BRAM Encryption Module is used to encrypt BRAM. The structure is shown in the Fig. 3. In order to reduce the clock overhead of BRAM storage and ensure the security of sensitive data, *BRAM Encryption Module* provides optional data encryption services. Regions can use the IS_ENCRY register to indicate whether the data transaction needs to be encrypted. Each transaction has an additional IS-ENCRY bit to indicate whether the transaction needs to be encrypted. If IS-ENCRY=1, encryption or decryption is required. If IS_ENCRY = 1, it needs to be encrypted. We add a *BRAM Encryption Module* between the BRAM and regions. When the region does not enable memory encryption, data from the region to BRAM will not have any additional delays.

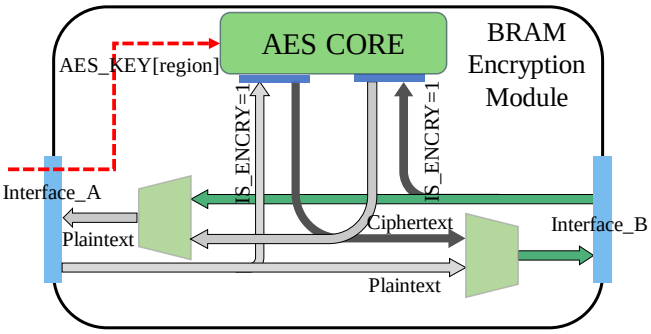


Fig. 3. BRAM Encryption Module.

BRAM Encryption Module uses the AES encryption core with a cipher block chaining (CBC) mode for encryption. Additionally, we consider the security of AES keys. FPGA-TrustZone stores AES keys in FPGA-SM, keeping them transparent to regions. Regions cannot see ciphertext and can only access plaintext decrypted by the *BRAM Encryption Module*, making it difficult to derive the key from plaintext alone. Encrypting BRAM data enhances confidentiality and security, protecting against physical and logical attacks. This ensures the storage security of the FPGA.

E. Data Flow Analysis

The FPGA-TrustZone security architecture encompasses both boot-time and runtime security. Boot-time security is ensured by ARM TrustZone and the FPGA trusted boot process. Runtime security relies on system components to protect programs and data from leakage or tampering. Fig. 4 provides a detailed overview of the data flow in the FPGA-TrustZone architecture. The PMU continuously monitors the system for abnormal behavior and illegal operations during data transmission.

During interaction between CPU applications and FPGA regions, secure-sensitive data is first encrypted by *SM-Extension*, ensuring it remains in ciphertext form during transmission, thereby enhancing resistance to side-channel attacks. Upon

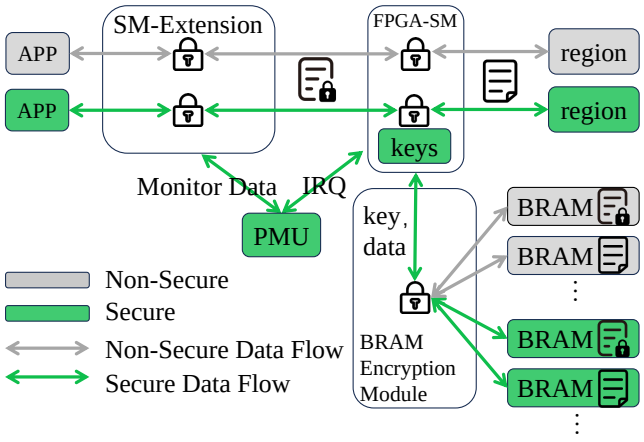


Fig. 4. Secure Data Flow.

TABLE I
ANALYSIS OF SECURITY ABILITIES OF THREE ARCHITECTURES

Ability	FPGA-TrustZone	SGX-FPGA	TPM-TrustZone	PUF-TrustZone
CPU Secure Boot	✓	✓	✓	✓
FPGA Secure Boot	✓	✓	✓	✗
BRAM Encryption	✓	✓	✓	✗
Monitoring Access	✓	✓	✗	✗
PMU IRQ Analysis	✓	✗	✗	✗
Poison Illegal Access	✓	✗	✗	✗
Cold-boot Attack Resistant	✓	✗	✓	✓

data arrival, *FPGA-SM* performs a legitimacy check on the transmission transaction. If the transaction is illegal, *FPGA-SM* sends an IRQ to the PMU and poisons the transaction. If the transaction is legal, *FPGA-SM* decrypts the data and transfers it to the corresponding region for processing. After processing, *FPGA-SM* encrypts the results and sends them to the CPU's *SM-Extension* for decryption and return to the application. Additionally, the region can use the *BRAM Encryption Module* to encrypt its BRAM. This allows sensitive data to be stored in encrypted form in BRAM, further enhancing system security.

V. EXPERIMENTAL EVALUATION

We implement a prototype system for the FPGA-TrustZone security framework on the Zynq UltraScale+XCZU9EG-2FFVB1156E MPSoC (ZCU102). Based on the prototype system, this section conducts two types of experiments to comprehensively evaluate the effectiveness and performance of FPGA-TrustZone: 1) security evaluation, in which we specifically analyzed how the security function implemented by FPGA-TrustZone is to resist the threat model proposed in Section III; 2) performance analysis, in which we analyze the cost of resources and time of FPGA-TrustZone.

A. Security Evaluation

The security of FPGA-TrustZone is verified through various situations, various applications and various illegal means to access the secure and non-secure regions. And we compare the security of FPGA-TrustZone with other secure FPGA architectures including SGX-FPGA ,TPM-TrustZone [16] and PUF-TrustZone [17] in Table I. FPGA-TrustZone demonstrates prominent advantages.

Moreover, FPGA-TrustZone can effectively prevent the threat model proposed in Section III: 1) *Internal attacks on the FPGA:* FPGA-TrustZone divides the FPGA-side into secure and non-secure areas, providing logical resource isolation and preventing malicious IPs from unauthorized access. And, FPGA-TrustZone implements a trusted boot process, ensuring system integrity and security. Data stored in external memory is encrypted, ensuring it exists as ciphertext. Even if attackers obtain the ciphertext, recovering the plaintext is difficult, so as to protect firmware code and key data from leakage or theft. Encrypting the Bitstream ensures data confidentiality during transmission and storage, thereby preventing tampering during transmission.

2) *Using a compromised FPGA to attack CPU-side resources:* FPGA-TrustZone resists attacks on the CPU-side by using ARM TrustZone to secure sensitive data and critical operations within a protected environment. Additionally, *SM-Extension* and *FPGA-SM* control FPGA access to the CPU, preventing threats by monitoring and verifying access transactions.

3) *Invaded CPU attacks the FPGA:* The *FPGA-SM* strictly controls CPU access, preventing unauthorized users from tampering with the FPGA security area and protecting the system from potential harm.

4) *External attacks between CPUs and FPGAs:* The encrypted data transmission can effectively prevent attacks such as eavesdropping and tampering, thereby ensuring data confidentiality.

TABLE II
RESOURCE USAGE OF THE INITIAL HARDWARE DESIGN, PUF, BRAM ENCRYPTION MODULE AND FPGA-SM.

Component	LUT	FF	LUTRAM	BUFG	BRAM
PUF	96 0.04%	292 0.05%	0 0%	0 0%	0 0%
BRAM Encryption Module	6765 2.47%	4939 0.90%	0 0%	4 0.99%	0 0%
FPGA-SM	3435 1.25%	2046 0.37%	0 0%	0 0%	0 0%
The Initial Hardware Design	24321 8.87%	25595 4.67%	3098 2.15%	6 1.49%	4 0.44%

B. Performance Evaluation

Then we measure the performance overhead of FPGA-TrustZone on the ZCU102 development board. The main hardware resources used on devices are as follows:

1) LookUp Table (LUT) and Flip-Flop (FF), used to implement logical functions.

2) LookUp Table RAM (LUTRAM) and BRAM, used to store data.

3) Global clock buffer (BUFG), used for clock signal management, allocation, and processing.

TABLE III
TIME OVERHEAD OF CRITICAL OPERATIONS.

Operation	Time Overhead
IV_generation	0.5 clock cycles
AES_CBC Algorithm	14.5 clock cycles
Memories access	2 ms
Poisoning illegal access	202 ms

TABLE IV
PERFORMANCE EVALUATION.

Benchmarks	Baseline	FPGA-TrustZone	Additional Overhead
Vadd	92439ns	113037ns	22.28%
Matmul	11665ns	13834ns	18.59%
Matmul: partition	1706ns	2042ns	19.70%

Firstly, we evaluate the resource overhead of FPGA-TrustZone. PUF, BRAM Encryption Module and FPGA secure monitor as the three major components. Here, *Used* represents the hardware resources used in the designed solution, while *Available* means the total number of configurable resources for ZCU102 development board. Then *Utilization Ratio=Used / Available*.

We evaluate the resource usage of three main components and the initial hardware design on the ZCU102 development board. Table II presents the hardware resource utilization. Results indicate that the resource cost brought by each component no more than 3% and the overall resource consumption does not exceed 9%. Users can customize hardware designs as needed, provided resource usage stays within available limits. Since the FPGA-TrustZone framework maintains a constant footprint across different SoC-FPGAs, its relative resource usage decreases on platforms with larger capacities.

In addition, we evaluate the time cost of part key operations related to the FPGA-side, including the BRAM encryption module's time consumption, illegal access poisoning time, and memory access operation time. Table III shows the results.

We test runtime performance using three Xilinx benchmark programs: vector addition, matrix multiplication, and matrix multiplication (partition). Table IV presents a comparison between our solution and an unsecured SoC-FPGA. Results show that FPGA-TrustZone introduces a 22.28% overhead for basic operations, which is acceptable given its protection of data storage, computation, and transmission security. Additionally, FPGA-TrustZone incurs an 18.59% overhead for parallel computation, indicating its suitability for parallel execution. The overhead introduced while protecting the overall security of the SoC-FPGA architecture is acceptable.

In summary, FPGA-TrustZone is feasible in terms of overhead on the ZCU102 development board.

VI. CONCLUSION

This paper proposes FPGA-TrustZone, a solution that extends Arm TrustZone's security features to the FPGA, achieving hardware resource isolation and protecting sensitive data in Soc-FPFA architecture. FPGA-TrustZone introduces three important components: FPGA Secure Monitor, Secure Monitor-Extension, and BRAM Protector. FPGA-TrustZone monitors transactions in real-time using Secure Monitor-Extension and FPGA Secure Monitor, verifying the legitimacy of access transactions and poisoning unauthorized access. This approach effectively enhances the overall security of the system. In addition, to further ensure the security of data, FPGA-TrustZone introduces BRAM Protector for encrypting data stored in BRAM. Our implementation and evaluation of actual hardware indicate that FPGA-TrustZone provides relatively low resource and performance overhead while ensuring security, providing a solid guarantee for system security and performance.

REFERENCES

[1] B. Li, B. Yan, and H. Li, "An overview of in-memory processing with emerging non-volatile memory for data-intensive applications," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 381–386.

[2] F. Liu, Y. Liang, and L. Wang, "A survey of the heterogeneous computing platform and related technologies," *Proc. DEStech Trans. Eng. Technol. Research IMEIA*, vol. 1, pp. 6–12, 2016.

[3] J. Hoozemans, J. Peltenburg, F. Nonnemacher, A. Hadnagy, Z. Al-Ars, and H. P. Hofstee, "Fpga acceleration for big data analytics: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 30–47, 2021.

[4] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[5] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: an open framework for architecting trusted execution environments." Association for Computing Machinery, 2020.

[6] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[7] T. Sun, B. Li, Y. Teng, Y. Gao, and W. Dong, "dtee: A declarative approach to secure iot applications using trustzone," in *2024 23rd ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2024, pp. 200–212.

[8] X. Xie, H. Wang, Z. Jian, T. Li, W. Wang, Z. Xu, and G. Wang, "Memory-efficient and secure dnn inference on trustzone-enabled consumer iot devices," *arXiv preprint arXiv:2403.12568*, 2024.

[9] D. Hwang, S. Yeleuov, J. Seo, M. Chung, H. Moon, and Y. Paek, "Ambassy: A runtime framework to delegate trusted applications in an arm/fpga hybrid system," *IEEE Transactions on Mobile Computing*, 2021.

[10] S. Pereira, D. Cerdeira, C. Rodrigues, and S. Pinto, "Towards a trusted execution environment via reconfigurable fpga," *arXiv preprint arXiv:2107.03781*, 2021.

[11] K. Xia, Y. Luo, X. Xu, and S. Wei, "Sgx-fpga: Trusted execution environment for cpu-fpga heterogeneous architecture," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 301–306.

[12] Y.-C. Lin, B. Zhang, and V. Prasanna, "Hitgnn: High-throughput gnn training framework on cpu+ multi-fpga heterogeneous platform," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[13] Y. Sun, H. Liu, X. Liao, H. Jin, and Y. Zhang, "Fpga-based acceleration architecture for apache spark operators," *CCF Transactions on High Performance Computing*, pp. 1–14, 2023.

[14] Y. Wang, X. Chang, H. Zhu, J. Wang, Y. Gong, and L. Li, "Towards runtime customizable trusted execution environment on fpga-soc," *arXiv preprint arXiv:2307.04375*, 2023.

[15] E. M. Benhani, L. Bossuet, and A. Aubert, "The security of arm trustzone in a fpga-based soc," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1238–1248, 2019.

[16] M. Gross, K. Hohentanner, S. Wiehler, and G. Sigl, "Enhancing the security of fpga-socs via the usage of arm trustzone and a hybrid-tpm," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 1, pp. 1–26, 2021.

[17] C. Aitchison, R. Buckle, A. Ch'ng, C. Clarke, J. Malley, and B. Halak, "On the integration of physically unclonable functions into arm trustzone security technology," in *2020 European Conference on Circuit Theory and Design (ECCTD)*, 2020, pp. 1–4.

Re^{4}PUF: A Reliable, Reconfigurable ReRAM-based PUF Resilient to DNN and Side Channel Attacks

Ning Lin[1,2,3,4,5,*], Yi Li[1,*], Yangu He[1,*], Songqi Wang[1], Hegan Chen[1,5], Kwunhang Wong[1,5], Chuxin Li[1], Jichang Yang[1], Yifei Yu[1], Meng Xu[1], Yongkang Han[7], Rui Chen[2], Xiaoming Chen[6], Xiaoxin Xu[2,3], Jianguo Yang[2,3], Dashan Shang[2,3,†], Zhongrui Wang[4,5,†]
[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China; [2]State Key Lab of Fabrication Technologies for Integrated Circuits, Institute of Microelectronics of the Chinese Academy of Sciences (CAS), Beijing, China; [3]Key Laboratory of Microelectronic Devices and Integrated Technology, Institute of Microelectronics of CAS, Beijing, China; [4]School of Microelectronics, Southern University of Science and Technology, Shenzhen, China; [5]AI Chip Center for Emerging Smart Systems (ACCESS), Hong Kong, China; [6]Institute of Computing Technology of CAS; [7]Zhangjiang Lab, Shanghai, China.
*Equal contribution to this work. †Corresponding author: wangzr@sustech.edu.cn;shangdashan@ime.ac.cn.

***Abstract*—Resistive random-access memory (ReRAM) based Physical Unclonable Functions (PUFs) have emerged as an attractive hardware security primitive due to their low energy consumption and compact footprint. However, the reliability of existing ReRAM-based PUFs is challenged by read noise and temperature variations, as well as their resistance to Deep Neural Network (DNN) modeling attacks and Side Channel Attacks (SCAs). In this paper, we propose a novel 3T2R ReRAM-based reconfigurable PUF to address these challenges. By adopting the digital 3T2R voltage division cell design, we improve its reliability against ReRAM read noise and temperature variations, while the adjustable analog supply voltage of inverters enables quick, low-cost reconfigurability without reprogramming ReRAMs, effectively mitigating DNN modeling and SCA vulnerabilities. Our Re^{4}PUF chip has been experimentally validated, achieving a low Bit Error Rate (BER) of 1% at 85°C, a 7.59-fold reduction compared to existing ReRAM-based PUFs. It also demonstrates robust resistance to both DNN modeling attacks (MLP and Transformer) and SCAs, with success rates of approximately 50% and less than 70%, respectively.**

***Index Terms*—ReRAM; PUF; SCA; DNN modeling attack; Reconfigurable.**

I. Introduction

The rapid progress of smart IoT necessitates enhanced security features, such as unique, device-specific fingerprints for key generation [1]. ReRAM-based PUFs uses inevitable variation in fabrication and redox reaction as the entropy source, which is promising for the high integration density, low operating energy consumption, reduced fabrication costs. However, these attributes also pose significant challenges to reliability, particularly in harsh environments. For instance, elevated temperatures of 85°C can result in a substantial BER of up to 11.5% [2]. Moreover, adversaries may model PUFs by collecting Challenge-Response Pairs (CRPs) and employing DNNs to replicate the functionality of the PUF, thereby impersonating the associated identity. This vulnerability is further intensified by SCAs, where attackers can probe ReRAM conductance [3].

ReRAM-based PUFs. In the early stages, ReRAM-based PUFs mimicked CMOS PUFs (e.g., arbiter PUFs [4], [5]). For instance, Ref. [4] leverages the random conductance of memristors to form two paths with different conduction latencies, generating a final 1-bit response through a D Flip-Flop. Subsequent advancements saw researchers organizing

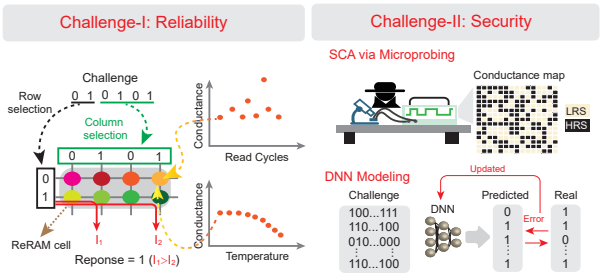


Fig. 1. Challenges faced by ReRAM-based PUFs: (I) Reliability issues arising from read noise and temperature variations. (II) Security vulnerabilities due to DNN modeling attacks and SCAs via microprobe techniques.

ReRAM into 2D and later 3D crossbar structures to serve as memory-based PUFs [2], [6]–[9]. These efforts primarily focused on exploiting the inherent randomness of ReRAM as an entropy source. For instance, Ref. [8] proposes using metal-oxide memristive crossbars to implement PUFs. In their design, the challenge signal serves as both the row and column selection signal, applying voltage to the ReRAM device at the intersection of these signals. The current of the two (even-numbered) columns is then compared to output the corresponding 1-bit 0/1 response.

Challenges. Both arbiter and memory-type ReRAM-based PUFs encounter significant challenges related to reliability and security, as illustrated in Fig. 1.

Regarding reliability, an ideal PUF should produce identical responses upon the same challenge, or a zero BER. However, the reliability of these PUFs is compromised by cycle-to-cycle (i.e., read) variations and sensitivity to temperature changes.

In terms of security, these ReRAM-based PUFs have been validated against various machine learning (ML) and DNN models, including logistic regression, support vector machines, and simple RNNs [2], [6]–[8]. However, their theoretical resistance to ML or DNN modeling attacks is limited due to the strong fitting capabilities of these models [10], which have seen significant advancements in recent years [11].

Current Solutions. To address reliability issues, existing solutions often incorporate additional peripheral circuitry or reprogramming strategies to enhance stability. For instance, Ref. [12] employs post-processing techniques such as Temporal Majority Voting (TMV) and Masking [13], which reduce the BER from 6% to zero. Similarly, Ref. [6] utilizes con-

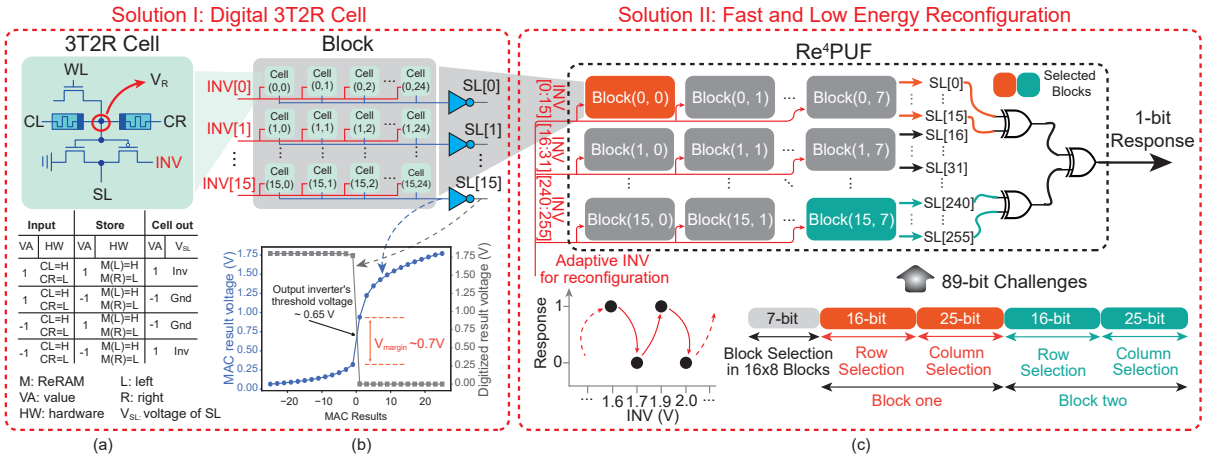


Fig. 2. Re^{4}PUF architecture. (a-b) A novel digital 3T2R cell in a 16×25 block. ReRAM cells form a voltage divider, with output binarized by an inventor. This mitigates reliability issues caused by read noise and temperature variation. (c) Fast and low energy reconfiguration to resist DNN modeling and SCA, by adapting inverter voltage INV in the option set (1.6V, 1.7V, 1.9V, 2.0V)

tinuous programming of each cell for calibration, lowering the BER from 8.4% to approximately 0.014%. However, the inclusion of additional peripheral circuitry undoubtedly introduces extra energy and area overhead, and reprogramming often requires iterative write-verification resulting in significant latency overhead.

To address the security issue, a promising Concealed ReRAM-based PUF [3], [12] has been proposed. This approach restores private conductance information during use and erases it when not in use, thereby mitigating SCAs. However, the repeated erasing and rewriting (i.e., SET and RESET programming of ReRAM) introduces additional latency and energy overhead. Furthermore, ReRAM devices have finite programming endurance [14]. The process of reprogramming also introduces write noise, resulting in inconsistencies with the original PUF. Additionally, the Concealed ReRAM-based PUF does not offer improvements in reliability.

Our Solution. To address these concerns, we propose a ReRAM-based strong PUF that employs an innovative 3T2R cell design. This 3T2R cell uses a voltage division circuit, significantly improves reliability and achieves a 1% BER at 85°C. Our design includes a lightweight reconfiguration mechanism, achieved by adjusting the supply voltage of inverters in the 3T2R cells. This effectively counters powerful DNN modeling attacks and SCAs. Specifically, our innovations are as follows,

- We identify two primary challenges for ReRAM-based PUFs: reliability and security. Additionally, we introduce two novel criteria for comprehensively evaluating the performance of ReRAM-based PUFs: resistance and reconfiguration.
- To address the reliability issue, we propose a new 3T2R cell design utilizing a voltage division circuit (refer to Solution I in Fig. 2 (a-b)). This design significantly reduces the BER, achieving a remarkable 1% BER at extreme temperatures of 85°C, compared to the traditional weighted-current summation (WCS) designs.
- To tackle the security challenge, we propose a reconfiguration method that eliminates hardware programming by adjusting the inverter supply voltage INV (refer to Solution II in Fig. 2 (c)). This lightweight approach effectively counters DNN modeling attacks (MLP and Transformer) and SCAs, with success rates of around 50% and less than 70%, respectively.

The remainder of the paper is organized as follows: Sections II and III present the proposed PUF design and its workflow. Section IV showcases the experimental results. Finally, Section V provides the conclusion.

II. Re^{4}PUF Design

A. Architecture design

Digital 3T2R cell. Existing ReRAM-based PUFs rely on analog memristor circuits to produce responses [2], [3], [6]–[8]. Typically, these systems utilize WCS to compare bitline (BL) currents with either the current generated from another BL or a reference current, thereby generating a response bit. However, the reliability of these systems is compromised by factors such as read noise and operating temperature fluctuations, which lead to undesirable conductance drift and variation.

Inspired by the voltage division cell design in [15], [16], which substantially enhances reliability in the face of memristor variations, we propose an innovative 3T2R cell (Fig. 2 (a)). This cell integrates two ReRAMs in complementary states, an inverter, and a programming transistor. The output of the ReRAM voltage divider (refer to V_R in Fig. 2 (a)) is binarized by the inverter, effectively performing a multiplication operation between the input (refer to CL/CR in Fig. 2 (a)) and the stored conductance. The binary inverter within the 3T2R cell further mitigates fluctuations in ReRAM resistance, such as read noise and temperature variations. The output voltages from all cells are aggregated on a source line (SL), which, after passing through a binary inverter, yields an output of either "0" or "1" (Fig. 2 (b)).

Fast and low energy reconfiguration. Our designed Re^{4}PUF consists of 16×8 blocks. Cells on the same SL share

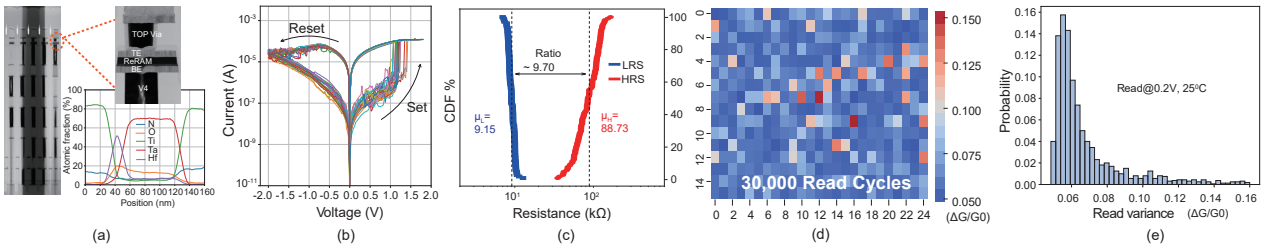


Fig. 3. (a) Cross-sectional Transmission Electron Micrograph (TEM) of a TaOx/HfOx-based ReRAM cell and the Energy-dispersive X-ray Spectroscopy (EDS) compositional profile. (b) DC I-V sweeps over 50 switching cycles. (c) LRS/HRS cumulative resistance distribution for 50 switching cycles. (d-e) The conductance variation heatmap and histogram of all cells within a block after being read 30,000 times at a voltage of 0.2V. ΔG stands for the variance conductance, and G0 is initial conductance.

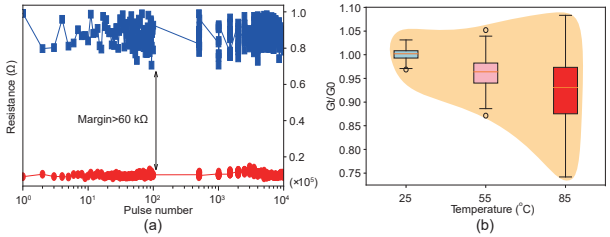


Fig. 4. (a) The endurance of LRS/HRS ReRAM cells. (b) Changes in conductance after baking for 24 hours at 25°C, 55°C and 85°C, Gt represents the measured conductance after baking, G0 is initial conductance.

the same inverter supply voltage INV. Different SLs have different INV voltages. Each combination of INV voltages of all SLs yields a unique PUF. Therefore, adjusting the INV vector reconfigures the PUF (Fig. 2 (c)). Here, each SL opts for one of four INV biasing voltages (1.6V, 1.7V, 1.9V, 2.0V), resulting in a vast selection space of 4^{256} for 256 SLs. An 89-bit challenge is partitioned into three segments: block selection (7-bit, choosing 2 blocks), row selection (32 bits, 16 bits per block), and column selection (50 bits, 25 bits per block). The output signals from the two blocks are individually subjected to two XOR operations, which are combined with an additional XOR operation to generate a 1-bit response (Fig. 2 (c)).

B. Device characteristics

Fabrication. ReRAM cells are integrated via back-end-of line (BEOL) process between Via4 and Metal5 of 180nm technology. They comprise top and bottom TiN electrodes (TE and BE) and a TaO_x/HfO_x switching layer (Fig. 3 (a)).

Entropy source. The operational principle of ReRAM hinges on the reversible switching between a low resistance state (LRS) and a high resistance state (HRS). The transition from HRS to LRS, termed the SET process, and the reverse transition, known as the RESET process, are both driven by the application of voltages exceeding specific thresholds. These processes involve stochastic formation and dissolution of conductive filaments within the memory material. Our experimental 50 cycles I-V sweeps, affirming the reproducibility of this bipolar switching behavior. The threshold voltages for SET and RESET are approximately 1.3V and -1.7V, respectively. Notably, the I-V characteristics for each cycle exhibit variations due to stochastic filament formation/rupture, leading to cycle-to-cycle (i.e., read variance) and device-to-device variation (i.e., write variance) (Fig. 3 (b)).

Write & Read variance. For write variance, HRS and LRS follow Gaussian distributions, with mean values of 9.15 kΩ and 88.73 kΩ respectively. This results in an on/off ratio of 9.7 (Fig. 3 (c)). This discrepancy between the expected and the actual programmed conductance values in both LRS and HRS underscores the intrinsic variability of ReRAM, providing the entropy source.

The read variance of ReRAM cells over 30,000 cycles is illustrated in Fig. 3 (d). The histogram indicates a maximum read variance of 0.16 (Fig. 3 (e)). This read variance can induce instability in PUF responses, presenting substantial challenges to the reliability.

Endurance & Temperature. To investigate the endurance of ReRAM cell, cyclic SET and RESET operations were conducted using a voltage pulse with an amplitude of +1.3/-1.6V and a width of 300/400ns. Simultaneously, a reading voltage pulse with an amplitude of 0.3V and a width of 5ns was applied to the ReRAM cell to measure its resistance. The results (Fig. 4 (a)) demonstrate that ten thousand SET and RESET cycles do not affect the performance of the ReRAM cell.

The conductance variance after baking at 25°C, 55°C, and 85°C for 24 hours exhibits significantly increased variation and a reduced mean value (Fig. 4 (b)). This observation suggests that higher temperatures have a pronounced impact on ReRAM-based PUFs, potentially leading to inconsistent responses and increased BERs.

III. Re^{4}PUF Workflow

The Re^{4}PUF workflow consists of three stages: initialization, reconfiguration, and CRPs generation, as illustrated in Fig. 5 (a).

Initialization. During the initialization phase, the conductance of the 3T2R ReRAM cells are programmed to random binary bits using the intrinsic programming stochasticity of ReRAM. The initialized system is ready for reconfiguration or CRPs generation.

Reconfiguration. Following initialization, the system enters the reconfiguration phase. During reconfiguration, the ReRAMs are not reset; instead, the inverter's voltage INV for each SL is set to specific values (1.6V, 1.7V, 1.9V, and 2.0V), which leads to a unique mapping between challenges and responses.

CRPs Generation. In the CRPs generation stage, the system processes the 89-bit challenge signal. The selection signals

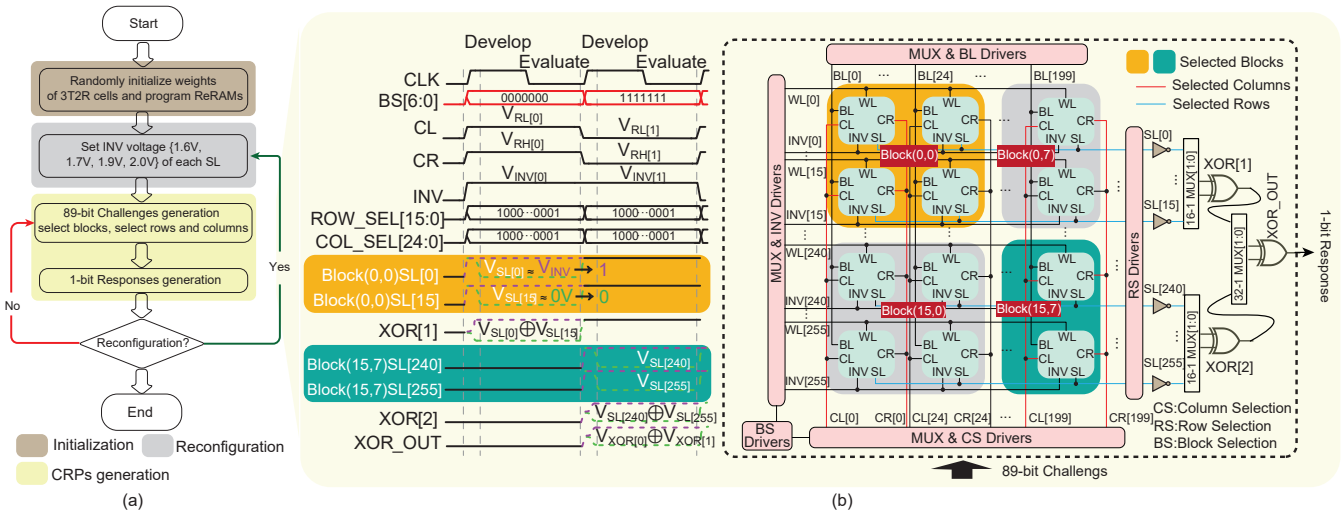


Fig. 5. (a) The workflow includes initialization, reconfiguration and CRPs generation. (b) During the CRPs generation, two blocks are activated by the BS[6:0] signal from the challenge signal. COL_SEL[24:0] signals from the challenge signal is applied to the CL and CR. The chosen block's outputs are selected by ROW_SEL[15:0] signals from the challenge and then undergo XOR operations, which produces the final 1-bit PUF response.

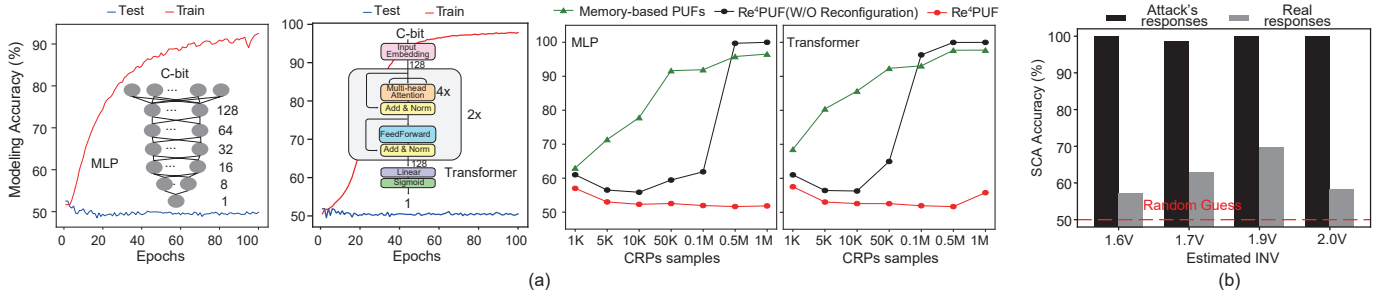


Fig. 6. (a) Re^{4}PUF successfully defends against DNN modeling, including MLP and Transformer. (b) In SCA, attackers fail to predict real responses due to the absence of true INV.

and their interactions during this phase can be further visualized by the waveform shown in Fig. 5 (b), which illustrates how the challenge signals propagate and interact within the ReRAM array. The BS[6:0] of the challenge activates specific blocks within the 16×8 ReRAM block array. The BS[6:3] signals select one of the 16 rows within the block array, while the BS[2:0] signals indicate one of the 8 columns. Following this, the COL_SEL[24:0] signal from the challenge determines which two columns, CL (left column) and CR (right column), within the selected block will be activated. Additionally, the ROW_SEL[15:0] signal from the challenge input refines the selection of row outputs from the activated blocks, further determining which rows will contribute to the response. Once the challenge signals are applied to the selected columns and rows, the activated ReRAM blocks produce two output bits via the two SLs, they undergo XOR operations to produce a unique 1-bit response specific to the applied challenge.

After generating the 1-bit response, the system assesses whether the reconfiguration is required or not.

IV. Experimental Results

The ReRAM of the Re^{4}PUF, featuring a cell size of 0.08μm × 0.08μm and composed of TiN/TaO_x/HfO_x/TiN layers, was fabricated using standard 0.18μm technology through the following steps: First, the TiN layer was deposited via physical vapor deposition to serve as the bottom electrode. Next, the TaO_x and HfO_x layers were sequentially deposited on top of the TiN layer using atomic layer deposition. Finally, a second TiN layer was deposited as the top electrode, using the same method as for the bottom layer. The electrical characteristics of the ReRAM cell and the performance of the PUF were evaluated using a Keysight B1500A Semiconductor Device Analyzer and a SUMMIT 12000 Probe Station.

A. Resistance

Resistance measures the effectiveness against modeling attacks and SCAs. For modeling attacks, this includes ML models like SVM, as well as powerful DNN models such as MLP and Transformer models [11]. Modeling accuracy ranges from 50% (indicating strong resistance) to 100% (indicating weak resistance). For SCAs, attackers can read ReRAM array conductance using probes [3].

Re^{4}PUF effectively defends against DNN modeling, including MLP and Transformer (refer to Fig. 6 (a) for model details). Increasing training epochs or the number of training samples does not improve the accuracy on test CRPs, which remains around 50%, akin to random guessing. Compared to traditional memory-based PUF in [8] (simulated here for comparison) and non-reconfigurable Re^{4}PUF, the reconfigurable Re^{4}PUF maintains strong resistance (Fig. 6 (a)).

Furthermore, even if attackers read ReRAM array conductance using probes [3] and train DNNs for a given INV

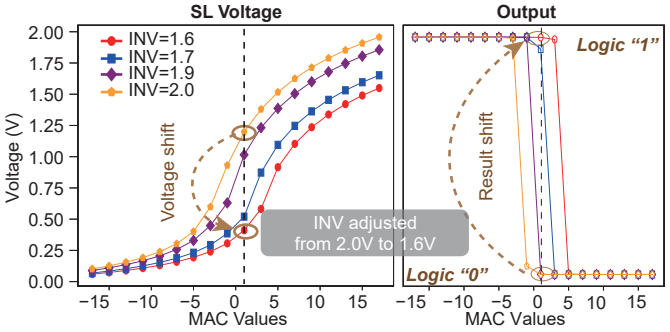


Fig. 7. INV voltage changes SL output, allow fast and low energy reconfigurability. When INV is adjusted from 2.0V to 1.6V, the output voltage of SL shifts from logic "0" to "1".

voltage vector, the real responses remain random guesses due to Re^{4}PUF's reconfigurable nature (Fig. 6 (b)).

B. Reconfiguration

Reconfiguration alters the mapping between challenges and responses, enabling the same input challenge to yield different output responses under different configuration. An optimal reconfiguration strategy should be both energy-efficient and low latency. Concealed ReRAM-based PUF necessitates the erasure of conductances to counter DNN modeling attacks and SCAs, a process that incurs significant costs due to erasing and reprogramming ReRAM cells [3]. To mitigate these drawbacks, our Re^{4}PUF leverages adjustments of inverter supply voltage INV for reconfiguration, avoiding the need to alter cell conductance values.

As illustrated in Fig. 7, modifying the INV voltage directly influences the SL output voltage, thereby changing MAC results. For instance, when the INV voltage is decreased from 2.0V to 1.6V, the SL voltage correspondingly drops from approximately 1.2V to 0.4V. This shift causes the SL inverter's output to transition from logic "0" to logic "1". This reconfiguration method is not only rapid but also get rid of ReRAM programming energy compared to Concealed ReRAM-based PUF [3].

C. Reliability

Reliability is assessed by evaluating the reproducibility of the PUF response under varying environmental conditions, such as temperature fluctuations. This reliability is quantified by the BER. An ideal PUF should produce the same response upon a given challenge across all conditions, or a BER of zero. However, the reliability of ReRAM-based PUFs is often compromised by read noise and conductance drift due to temperature changes.

Traditional memory-type PUFs in [8] (simulated here for comparison) can exhibit a peak BER of 2.5% in repeated CRP generation (Fig. 8 (a)) and up to 30% when more devices are selected for response generation (Fig. 8 (b)). In stark contrast, our Re^{4}PUF, utilizing a voltage divider in each cell, achieves a 0% BER, representing a significant enhancement. As temperature rises, the single ReRAM cell design used in memory-type PUF can lead to conductance changes and current variations, resulting in accumulated errors and higher BER. Our 3T2R

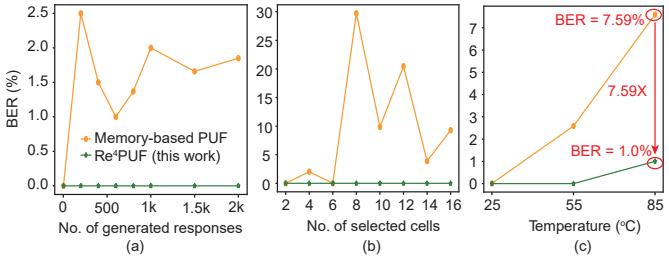


Fig. 8. The BER of Re^{4}PUF under 2k generation cycles (a), the population of selected cells (b), and temperature variations (c).

voltage division design integrates two ReRAMs in complementary states, an inverter, and a programming transistor. This configuration ensures that fluctuations in individual cell conductance do not impact the overall resistance ratio, thereby significantly enhancing reliability. Specifically, while the BER of traditional memory-type PUF in [8] escalates to 7.59% at 85°C, our Re^{4}PUF maintains a low BER of 1% under the same conditions (Fig. 8 (c)).

D. Security Metric

Uniformity (UF) measures the ratio between "0"s to "1"s in PUF's responses. The uniformity of a K-bit-long binary response is defined as the normalized hamming weight [8],

$$UF(B_i) \equiv \frac{1}{K}\sum_{k=1}^{K} b_{ki}, \tag{1}$$

where b_{ki} denotes the k-th bit of the i-th response B_i. The average uniformity is given by,

$$\langle UF \rangle = \frac{1}{C}\sum_{i=1}^{C} UF(B_i), \tag{2}$$

where C is the total number of CRPs. An ideal PUF should yield responses with 50% "0"s and "1"s, as any deviation from this balance could compromise security.

Uniqueness (UQ) measures the dissimilarity between response vectors from different PUF instances under the same input challenge. The uniqueness between two response vectors to the same i-th challenge, from the l-th and p-th PUF instances, is defined as the inter-PUF normalized hamming distance [8], where d measures the hamming distance between two binary vectors,

$$UQ(B_i^p, B_i^l) \equiv \frac{1}{K} d(B_i^p, B_i^l). \tag{3}$$

The uniqueness for the i-th challenge, averaged across all possible pairwise combination of PUF instances, is,

$$\langle UQ(B_i) \rangle \equiv \frac{2}{P(P-1)}\sum_{p=1}^{P}\sum_{l=p+1}^{P} UQ(B_i^p, B_i^l), \tag{4}$$

where P is the total number of PUF instances. Then the final averaged uniqueness is,

$$\langle UQ \rangle \equiv \frac{1}{C}\sum_{i=1}^{C} \langle UQ(B_i) \rangle. \tag{5}$$

The ideal value for uniqueness is 50%, indicating maximum dissimilarity.

TABLE I
COMPARISON OF THIS WORK WITH OTHER RERAM-BASED PUFS.

	VLSI'18 [7]	NE'18 [8]	IEDM'19 [2]	IEDM'20 [6]	Sci.Adv.'22 [3]	DATE'24 [12]	Re^{4}PUF (This Work)
Technology (nm)	200	55	N/A	N/A	130	90	180
NO. of CRPs #	$\binom{M}{m}\binom{N}{n}$	$\binom{M}{m}\binom{N}{n}$	$\binom{M}{m}\binom{N}{n}$	$\binom{M}{m}\binom{N}{n}$	$\binom{M}{m} \times N$	N/A	$\binom{M}{m}\binom{N}{n}$
Working mechanism*	WCS	WCS	WCS	WCS	WCS	VD	VD
Uniformity (%)	50.01	49.5	50.07	N/A	50.13	49.55	48.84
Diffuseness (%)	N/A	50.0	49.93	N/A	49.903	N/A	49.97
Uniqueness (%)	N/A	50.0	49.95	50.004	50.516	49.94	49.99
BER @ 85°C	4.00%	N/A	11.50%	0.01% (reprogramming)	about 4.00%	about 2% (TMV & Masking)	1.00%
Reconfigration	No	No	No	No	SET/RESET	SET/RESET	adaptive INV
Modeling accuracy	0.53	0.5	0.5	about 0.5	about 0.7	N/A	about 0.5
SCA accuracy	N/A	N/A	N/A	N/A	about 0.7	N/A	less than 0.7
Efficiency (pJ/bit)	0.21	0.02	0.004	N/A	12.03	2.225	1.108

* WCS: weighted-current summation. VD: voltage-divider.
# The number of CRPs in each $M \times N$ ReRAM array (block), where m and n represent the selected rows and columns, respectively

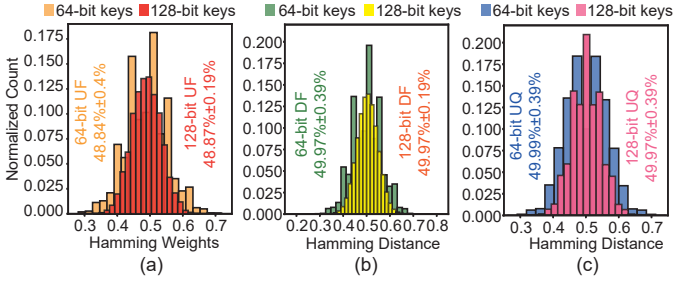


Fig. 9. The UF (a), DF (b) and UQ (c) distributions are computed based on 10k randomly generated 64-bit and 128-bit keys at 0.2V and 25°C.

Diffuseness (DF) quantifies the unique information in a PUF instance. It is defined as the intra-PUF normalized hamming distance d between the i-th and j-th responses [8],

$$DF(B_i, B_j) = \frac{1}{K} d(B_i, B_j). \tag{6}$$

The average diffuseness for all pairwise combination of challenges is,

$$\langle DF \rangle \equiv \frac{2}{C(C-1)} \sum_{i=1}^{C} \sum_{j=i+1}^{C} d(B_i, B_j), \tag{7}$$

with an ideal value of 50%.

The average uniformity, diffuseness, and uniqueness (across 10 dies) of the 64-bit and 128-bit keys generated by Re^{4}PUF, which are composed of 64-bit and 128-bit responses respectively, are close to the ideal value of 50%, as shown in Fig. 9.

E. Comparisons

Table I provides a comparative analysis of the proposed Re^{4}RAM PUF against other state-of-the-art ReRAM-based PUFs. The PUFs discussed in Refs. [2], [3], [6]–[8] utilize challenges to select different ReRAM cells within a 2D or 3D array, performing WCS and current comparison, featuring a large space of CRPs. In contrast, Ref. [12] employs a voltage division scheme with two adjacent device cells, yielding fewer CRPs. To increase the CRP space, we develop 2D ReRAM array using a 3T2R cell design that inherently supports voltage division. The CRP space is calculated as $\binom{M}{m}\binom{N}{n}$ per block.

At 85°C, existing ReRAM PUFs [6], [12] require reprogramming or techniques like TMV and Masking [13] to reduce the BER. In contrast, our 3T2R cell achieves a 1% BER without TMV or Masking.

Current ReRAM-based PUFs typically rely on SET/RESET operations for reconfigurability, whereas we achieve this with simple voltage adjustments. For modeling attacks and SCAs, previous ReRAM-based PUFs are tested against SVM and shallow MLP. We verify our design on DNNs including deep MLP and Transformer models, showing approximately 50% DNN modeling accuracy and and less than 70% SCAs accuracy, which confirms the strong resistance.

V. CONCLUSION

ReRAM-based PUFs have historically faced significant challenges in both reliability and security. Current ReRAM-based PUFs typically operate in a weighted-current summation mode, but due to the inherent noise of the devices, their stability is relatively poor. Furthermore, post-tapeout chips are essentially fixed black boxes, unable to theoretically or practically resist DNN modeling attacks. Existing reconfigurable PUFs based on conductance programming methods require extensive programming operations in practice, which not only incurs significant overhead but also may fail to restore the original PUF functionality. Re^{4}PUF with a 3T2R cell design significantly enhances reliability, while the adjustment of the INV voltage streamlines the reconfiguration process and enhances security. We believe that Re^{4}PUF will provide the hardware security community with a lightweight, secure, and reliable PUF solution.

ACKNOWLEDGMENT

This research is supported by the National Key R&D Program of China (Grant No. 2020AAA0109005), National Natural Science Foundation of China under Grant 62488101, 62495104 and 62374181, Shenzhen Science and Technology Innovation Commission (Grant No. SGDX20220530111405040), Beijing Natural Science Foundation (Grant No. Z210006), Hong Kong Research Grant Council (Grant Nos. 27209621, 17205922, 17212923), Joint Laboratory of Microelectronics (JLFS/E-601/24). This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Sudhir Kudva, Mahmut Sinangil, Stephen Tell, Nikola Nedovic, Sanquan Song, Brian Zimmer, and C. Gray. 16.4 high-density and low-power puf designs in 5nm achieving 23× and 39× ber reduction after unstable bit detection and masking. pages 302–304, 02 2024.

[2] MR Mahmoodi, H Nili, Z Fahimi, S Larimian, H Kim, and D Strukov. Ultra-low power physical unclonable function with nonlinear fixed-resistance crossbar circuits. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 30–1. IEEE, 2019.

[3] Bin Gao, Bohan Lin, Yachuan Pang, Feng Xu, Yuyao Lu, Yen-Cheng Chiu, Zhengwu Liu, Jianshi Tang, Meng-Fan Chang, He Qian, et al. Concealable physically unclonable function chip with a memristor array. *Science advances*, 8(24):eabn7753, 2022.

[4] Urbi Chatterjee, Rajat Subhra Chakraborty, Jimson Mathew, and Dhiraj K Pradhan. Memristor based arbiter puf: Cryptanalysis threat and its mitigation. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pages 535–540. IEEE, 2016.

[5] Rekha Govindaraj, Swaroop Ghosh, and Srinivas Katkoori. Design, analysis and application of embedded resistive ram based strong arbiter puf. *IEEE Transactions on Dependable and Secure Computing*, 17(6):1232–1242, 2018.

[6] Jianguo Yang, Dengyun Lei, Deyang Chen, Jing Li, Haijun Jiang, Qingting Ding, Qing Luo, Xiaoyong Xue, Hangbing Lv, Xiaoyang Zeng, et al. A machine-learning-resistant 3d puf with 8-layer stacking vertical rram and 0.014% bit error rate using in-cell stabilization scheme for iot security applications. In *2020 IEEE International Electron Devices Meeting (IEDM)*, pages 28–6. IEEE, 2020.

[7] Mohammad Reza Mahmoodi, Hussein Nili, and Dmitri B Strukov. Rx-puf: Low power, dense, reliable, and resilient physically unclonable functions based on analog passive rram crossbar arrays. In *2018 IEEE Symposium on VLSI Technology*, pages 99–100. IEEE, 2018.

[8] Hussein Nili, Gina C Adam, Brian Hoskins, Mirko Prezioso, Jeeson Kim, M Reza Mahmoodi, Farnood Merrikh Bayat, Omid Kavehei, and Dmitri B Strukov. Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors. *Nature Electronics*, 1(3):197–202, 2018.

[9] Yansong Gao, Damith C Ranasinghe, Said F Al-Sarawi, Omid Kavehei, and Derek Abbott. mrpuf: A novel memristive device based physical unclonable function. In *Applied Cryptography and Network Security: 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers 13*, pages 595–615. Springer, 2015.

[10] Shaza Zeitouni, Emmanuel Stapf, Hossein Fereidooni, and Ahmad-Reza Sadeghi. On the security of strong memristor-based physically unclonable functions. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017.

[12] Jiang Li, Yijun Cui, Chenghua Wang, Weiqiang Liu, and Shahar Kvatinsky. A concealable rram physical unclonable function compatible with in-memory computing. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[13] Sudhir Satpathy, Sanu K Mathew, Vikram Suresh, Mark A Anders, Himanshu Kaul, Amit Agarwal, Steven K Hsu, Gregory Chen, Ram K Krishnamurthy, and Vivek K De. A 4-fj/b delay-hardened physically unclonable function circuit with selective bit destabilization in 14-nm trigate cmos. *IEEE Journal of Solid-State Circuits*, 52(4):940–949, 2017.

[14] Zhongrui Wang, Saumil Joshi, Sergey E Savel'ev, Hao Jiang, Rivu Midya, Peng Lin, Miao Hu, Ning Ge, John Paul Strachan, Zhiyong Li, et al. Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature materials*, 16(1):101–108, 2017.

[15] Miguel Angel Lastras-Montano and Kwang-Ting Cheng. Resistive random-access memory based on ratioed memristors. *Nature Electronics*, 1(8):466–472, 2018.

[16] Qi Liu, Bin Gao, Peng Yao, Dong Wu, Junren Chen, Yachuan Pang, Wenqiang Zhang, Yan Liao, Cheng-Xin Xue, Wei-Hao Chen, Jianshi Tang, Yu Wang, Meng-Fan Chang, He Qian, and Huaqiang Wu. 33.2 a fully integrated analog reram based 78.4tops/w compute-in-memory chip with fully parallel mac computing. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 500–502, 2020.

ReaLM: Reliable and Efficient Large Language Model Inference with Statistical Algorithm-Based Fault Tolerance

Tong Xie[21], Jiawang Zhao[1], Zishen Wan[5], Zuodong Zhang[24], Yuan Wang[23], Runsheng Wang[234], Ru Huang[234], and Meng Li[123*]
[1]*Institute for Artificial Intelligence &* [2]*School of Integrated Circuits, Peking University, Beijing, China*
[3]*Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China*
[4]*Institute of Electronic Design Automation, Peking University, Wuxi, China*
[5]*Georgia Institute of Technology, Atlanta, GA, USA*

***Abstract*—The demand for efficient large language model (LLM) inference has propelled the development of dedicated accelerators. As accelerators are vulnerable to hardware faults due to aging, variation, etc, existing accelerator designs often reserve a large voltage margin or leverage algorithm-based fault tolerance (ABFT) techniques to ensure LLM inference correctness. However, previous methods often overlook the inherent fault tolerance of LLMs, leading to high computation and energy overhead. To enable reliable yet efficient LLM inference, in this paper, we propose a novel algorithm/circuit co-design framework, dubbed ReaLM. For the first time, we systematically characterize the fault tolerance of LLMs by performing a large-scale error injection study of representative LLMs and natural language understanding tasks. Then, we propose a statistical ABFT algorithm that fully leverages the error robustness to minimize error recovery as much as possible. We also customize the error detection circuits to enable a low-cost online collection of error statistics. Extensive experiments show that with only 1.42% circuit area and 1.79% power overhead, our ReaLM can reduce perplexity degradation from 18.54 to 0.29. Compared to existing methods, ReaLM consistently reduces recovery costs across different operating voltages and improves energy efficiency by up to 35.83% without compromising LLM performance. Our error injection code is available at https://github.com/PKU-SEC-Lab/ReaLM_DAC25/.**

***Index Terms*—Large language model, algorithm-based fault tolerance, algorithm/circuit co-design**

I. Introduction

Large language models (LLMs) have demonstrated remarkable performance across various tasks such as natural language processing [1], text generation [2], etc. With trillions of operations, LLMs are widely deployed on customized accelerators such as TPU-like systolic arrays (SAs) [3], [4] to meet their enormous computational demands. However, due to aging, process variation, noise, etc [5]–[8], reliability issues, e.g., timing violations, may emerge and make it challenging to ensure the computational correctness on these accelerators. Therefore, how to enable reliable yet efficient LLM acceleration becomes an important question and attracts increasing attention.

Although a large voltage margin can be reserved to prevent timing errors, it is usually not preferred due to the drastic increase in hardware costs. Meanwhile, directly reducing the voltage margin may raise the computation bit error rate (BER) and result in severe performance degradation, as shown in Fig. 1(a). To enable reliable yet efficient computation, various fault mitigation techniques from the circuit to the algorithm level have been proposed but often remain costly in the context of LLM inference. At the circuit level, techniques such as redundancy [9], [10], and Razor flip-flops (FFs) [11]–[14] are developed for traditional logic circuits but usually lack scalability for deep learning accelerators. At the algorithm level, fault-aware fine-tuning has been suggested for conventional convolutional neural networks (CNNs) [15]–[17]. However, the associated costs for LLM fine-tuning are prohibitive. Algorithm-based fault tolerance (ABFT) [18], an algorithm and circuit co-design technique designed for general matrix-matrix multiplication (GEMM), emerges as a cost-effective solution for error detection. ABFT detects errors in the computation results and triggers error recovery mechanisms, e.g., overvolting in dynamic voltage frequency scaling (DVFS) [19]–[21], recomputation, leading to much better efficiency.

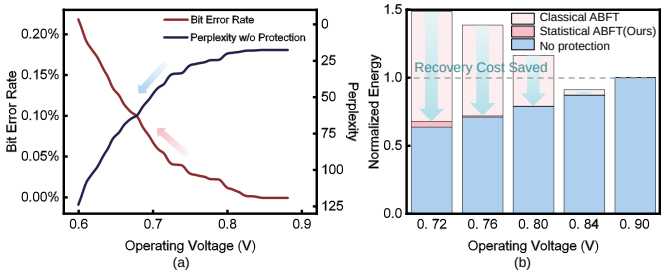


Fig. 1. (a) Lower operating voltages increase BER, leading to significant perplexity degradation without protection. (b) Leveraging model resilience can reduce recovery costs. BERs are synthesized on an SA with commercial 14nm PDK (nominal voltage: 0.9V), aligning with prior studies [11], [22], [23]. Perplexity is evaluated using `OPT-1.3B` on WikiText-2 dataset.

However, as shown in Fig. 1(b), we observe that ABFT still faces efficiency challenges due to the high recovery costs. This is because classical ABFT triggers recovery for each detected computation error, which may not be necessary. As in Fig. 1(a), LLM performance does not degrade immediately at low BER, indicating inherited error resilience. By leveraging the error resilience, unnecessary recovery can be minimized to boost the system's energy efficiency. Nevertheless, the error resilience of LLMs is still not yet well understood. While past works have studied CNNs extensively [24]–[27], the complex architecture, diversified computation patterns, and unique quantization strategies of LLMs can make it significantly different.

Therefore, to fully exploit this inherent error resilience, we propose ReaLM, an algorithm-circuit co-design framework that features reliable and efficient LLM inference. To the best of our knowledge, this is the first work that systematically and comprehensively characterizes the error resilience of LLMs. Our investigation reveals significant variations in error resilience across LLM network components due to the normalization operations. We also identify a trade-off between error magnitude and frequency in their impact on model performance. Building on these characterization insights, ReaLM further proposes a statistical ABFT approach to enable reliable and efficient LLM inference. ReaLM leverages the inherent resilience variations in LLMs to adaptively protect LLM network components in a cost-effective manner. We summarize our contributions as follows:

- We perform a large-scale error injection study on representative LLMs and systematically characterize how various hardware faults affect LLM performance. Our characterization reveals significant inherent resilience variations across LLM network components and identifies a trade-off between error magnitude

*Corresponding author: meng.li@pku.edu.cn

This work was supported in part by National Natural Science Foundation of China under Grant 62495102 and Grant 92464104, in part by Beijing Municipal Science and Technology Program under Grant Z241100004224015, and in part by 111 Project under Grant B18001.

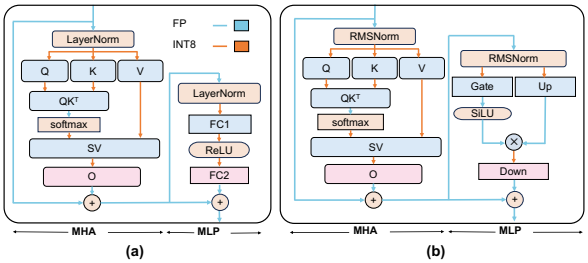


Fig. 2. Transformer blocks of (a) `OPT` and (b) `LLaMA`.

and frequency.

- Based on the characterization insights, we propose a statistical ABFT strategy featuring an adaptive error correction *algorithm* and a customized low-cost error detection *circuit*. The proposed strategy integrates seamlessly with SA, minimizing recovery costs while maintaining performance. It further enables energy savings by leveraging LLM's inherent resilience characteristics.
- Experimental results demonstrate that the proposed ReaLM significantly enhances error tolerance. ReaLM reduces perplexity degradation from 18.54 to 0.29 with only 1.42% circuit area and 1.79% power overhead, preserving LLM performance while significantly reducing recovery costs and delivering up to 35.83% energy savings compared to state-of-the-art methods.

II. Background

A. Quantized Inference of Generative LLMs

A typical LLM architecture primarily consists of stacked Transformer blocks, each containing several GEMM operations and various nonlinear functions. Fig. 2(a) and (b) illustrate the architectures of two popular Transformer variants, namely the `OPT` block [28] and the `LLaMA` block [29], respectively. Both variants consist of various **network components**, as labeled in Fig. 2, which will be referred to as `Q`, `Down`, and so on throughout this paper. Existing works [30]–[32] have enabled low bit-width integer quantization of these GEMM operations, while nonlinear functions are retained in floating-point (FP) for accuracy. The generative inference process of an LLM is divided into two stages: the prefill stage takes a prompt sequence as input, and the decode stage produce a single output token each time.

B. Fault-Tolerant Deep Learning

Error sources. Hardware accelerators are susceptible to various permanent and transient faults. Permanent faults often result from fabrication defects, while transient faults primarily arise from timing errors [7], [35]. As CMOS technology scales down to the nanoscale, effects such as aging, variation, and noise become increasingly pronounced [5]–[8]. while voltage underscaling further reduces the design margin intended to mitigate these issues. Together, these factors lead to unintended timing delays, ultimately causing timing errors, as shown in Fig. 1(a).

Model resilience. To effectively characterize the inherent resilience of neural networks, many works [22], [36]–[39] have employed a single-bit flip error model. [25] analyzed fault propagation in networks, and [26] assessed how quantization improves fault tolerance. [27] identified vulnerability differences of certain feature maps and suggested selective protection. [23], [40] further extended resilience studies to reinforcement learning applications. While [41] explored error resilience in LLMs, they did not quantify the relationship between errors and model performance. Considering the intricate model architecture, diversified network components, and unique quantization strategies, the error resilience of LLMs can be significantly different from traditional CNN models.

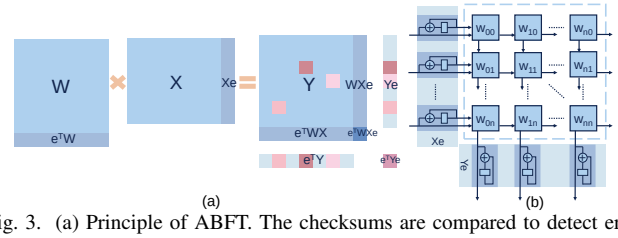


Fig. 3. (a) Principle of ABFT. The checksums are compared to detect errors and capture error statistics. (b) Implementation of ABFT on SA [44].

Error mitigation techniques. Error correction code (ECC) [42] can effectively address hardware faults in memory or on-chip buffers. Mitigating transient faults in computational datapaths is more challenging [38]–[40], [43], where various levels of protection can be leveraged, as summarized in Tab. I. However, these methods incur significant hardware or runtime overhead, particularly for LLMs. For example, redundancy methods [9], [10] such as double-modular redundancy (DMR), require multiple copies of computations to be performed concurrently. Circuit-level techniques [11]–[14] integrate shadow FFs to detect timing errors but lack scalability in modern accelerators. Algorithmic-level approaches [15]–[17] incorporate faults in fine-tuning to enhance error resilience. However, retraining LLMs is prohibitively computation-intensive.

C. Algorithm-Based Fault Tolerance (ABFT)

ABFT leverages checksums to enhance reliability in GEMM [18], as depicted in Fig. 3(a). In classical ABFT, matrices W and X are augmented with a checksum row $e^{\mathrm{T}}W$ and column Xe for error detection, where e is a unit column vector. Recent studies [33], [34], [44]–[46] also propose various lightweight detection algorithms which only compute one-side checksums, i.e., $e^{\mathrm{T}}WX$ or WXe, or matrix sum deviation (MSD), i.e., $e^{\mathrm{T}}WXe$, to lower ABFT hardware cost, as shown in Fig. 3(a). We also show an example of integrating ABFT with the SA following [44] in Fig. 3(b). While ABFT is suitable for LLMs that depend extensively on GEMM operations, it relies on precise checksum-based comparisons and overlooks the inherent resilience of neural networks. Recent work ApproxABFT [45] uses MSD to assess the error significance and allow small computational errors. However, this method ignores the impact of error frequency on the model accuracy and can lead to unnecessary recomputations.

Therefore, to address these challenges, we introduce ReaLM, a circuit-algorithm co-design framework designed to enhance the reliability and efficiency of LLM inference. We first inject computational errors into LLMs to study their resilience characteristics. Based on this characterization, we determine an adaptive error detection strategy to avoid unnecessary error recovery. Guided by this strategy, we customize a low-cost circuit to capture the computational error statistics.

III. Error Injection Framework

To study the resilience of LLMs, we performed statistical error injection experiments, the most widely used approach for assessing the effects of hardware faults on various models.

A. Error Model

Given that (i) permanent errors are relatively straightforward to detect [47] and do not require mitigation during runtime, and (ii) hardware faults in memory can be effectively addressed using ECC [27], [38]–[40], [43], in this work, we focus on transient computational errors occurring during inference and assume that data is correctly fetched from memory elements. To model hardware faults, we adopt the commonly used random bit-flip model [22], [36]–[39] as an abstraction. Specifically, timing errors are modeled by injecting

TABLE I
COMPARISON OF REPRESENTATIVE FAULT MITIGATION TECHNIQUES.

Method	Level	Detection Capability	Hardware Efficiency	Recovery Efficiency	Recovery Capability	Scalability	Compatibility w/ Accelerators
Redundancy [9], [10]	circuit	high	low	low	high	medium	medium
Razor FFs [11]–[14]	circuit	high	low	medium	low	low	low
Fault-aware Fine-tuning [15]–[17]	algorithm	-	-	prohibited	-	low	-
Classical ABFT [18], [33], [34]	circuit-algorithm	high	medium	low	high	high	high
Ours	circuit-algorithm	high	high	high	high	high	high

errors to higher bits, since they often affect more significant bits [7], [22], [46]. The severity of transient faults can be controlled by varying bit error rates.

B. Error Injection Method

We conduct our error injection study on two representative LLMs: `OPT-1.3B` [28] and `LLaMA-2-7B` [29]. To simulate computational errors, we developed a dynamic error injection simulation framework using PyTorch, wherein bit flips are emulated as tensor operations during runtime. Following [30], the inputs of GEMM are quantized to INT8, while the results are in INT32 format. Errors are then injected into the INT32 results of GEMM to realistically emulate compute errors in accelerators, which aligns with previous works [36]–[39].

Furthermore, to discern the different impacts of individual large errors versus multiple smaller errors that yield the identical MSD, we inject identical errors with defined error magnitude (mag) and error frequency ($freq$), adhering to the relationship $MSD = freq \times mag$. It is important to note that while this method diverges from the error models discussed in Sec. III-A, its effectiveness in assessing the resilience of LLMs is preserved.

C. Performance Benchmarks

To assess the effects of error injection, we also construct a comprehensive benchmark framework. Consistent with previous studies [30]–[32], we evaluate model performance using the LAMBADA commonsense QA benchmark [48] and the WikiText-2 language modeling task [49]. Performance on LAMBADA is measured by accuracy (↑), while WikiText-2 is evaluated using perplexity (↓) [50]. We further capture the autoregressive generation capabilities of LLMs using the X-Sum [51] text summarization task, measured by ROUGE-1 [52] scores (↑), and the GSM8K [53] arithmetic reasoning task, evaluated by accuracy (↑). Our framework can be extensible to other benchmarks, like Hellaswag [54] (measured by accuracy (↑)), which we examine in the evaluation section.

IV. LLM RESILIENCE CHARACTERIZATION

In this section, we present our LLM resilience characterization in detail, setting out to answer the following research questions:

- **Q1.1**: How does resilience vary across different layers of LLMs? (Sec. IV-A1)
- **Q1.2**: What is the bit-wise resilience of LLMs? (Sec. IV-A2)
- **Q1.3**: How does resilience vary among different components within LLMs during the prefill stage? (Sec. IV-A3)
- **Q1.4**: What is the correlation between error magnitude and frequency in impacting LLM performance? (Sec. IV-A4)
- **Q2.1**: How does the resilience of LLMs compare between the prefill stage and the decode stage? (Sec. IV-B1)
- **Q2.2**: How does resilience vary among different computational components within LLMs during the decode stage? (Sec. IV-B2)

To control for irrelevant variables, we implement the following protocols across research questions. (1) For Q1.4, we exclusively use `OPT-1.3B` model for fast assessment, whereas for Q2.1 and Q2.2, we employ only `LLaMA-2-7B` model to ensure reasonable performance. For all other questions, both models are utilized. (2) Error injections are applied to all layers for all questions, except for Q1.1. (3) In Q1.1, Q1.3, Q2.1 and Q2.2, we specifically target the 30th bit for flipping. (4) For Q1.1 and Q2.1, errors are injected across all network components within a single Transformer block simultaneously.

A. Resilience Study on Prefill Stage

In this subsection, we will directly give the conclusion of Q1.1 and Q1.2 since they are relatively straightforward, and we focus on discussing Q1.3 and Q1.4.

1) Layer-wise resilience: As shown in Fig. 4(a)(b), **while the layers exhibit comparable resilience behaviors, the earlier layers are more vulnerable to errors.**

2) Bit-wise resilience: By comparing error impacts in network components with FP outputs (`O`, Fig. 4(d)) and re-quantized INT8 outputs (`K`, Fig. 4(c)), we conclude that **while errors at lower bits have a negligible impact on model performance, errors at higher bits can reach a saturation point due to re-quantization.**

3) Resilience of different network components: As depicted in Fig. 4 (e)(f), our results reveal substantial differences in resilience behavior across the network components. Specifically, the `O` component in both models, `FC2` in `OPT-1.3B`, and `Down` in `LLaMA-2-7B` exhibit significantly higher sensitivity compared to other components. For ease of discussion, these components are referred to as ***sensitive components***, while the others are categorized as ***resilient components***.

Discussion: As demonstrated in Fig. 2, the sensitive components in both `OPT-1.3B` and `LLaMA-2-7B` are followed by normalization layers—LayerNorm and RMSNorm, respectively, unlike their resilient counterparts. We demonstrate in Fig. 5 that upon introducing a single error into the pre-normalization hidden states, a dramatic alteration in the subsequent elements is observed. This substantial variation is primarily due to the calculations of mean (μ) and standard deviation (σ) during normalization. In LLMs, hidden states inherently consist of a few outliers [30]–[32], while the majority of elements remain close to zero. These outliers predominantly influence the computation of μ and σ, shaping the overall statistics of the hidden states. Therefore, a larger error can manifest as an outlier and drastically skew these parameters, markedly altering the normalization outcome. In contrast, injecting a larger error into the resilient components affects only a localized set of elements. Thus, we conclude that **the network components followed by normalization exhibit much worse resilience than other components due to normalization.** Specifically, for $\mathtt{QK}^{\mathtt{T}}$ components followed by softmax, changes remain confined because softmax outputs between 0 and 1.

4) Tradeoff between error magnitude and frequency: While MSD can serve as a metric to depict the error severity, it cannot differentiate

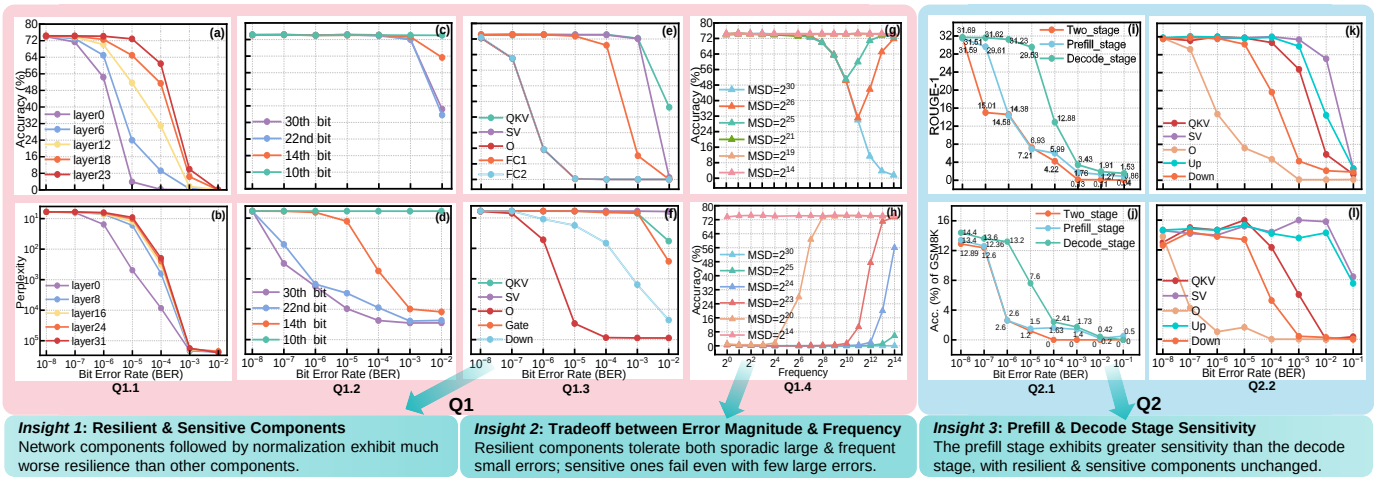


Fig. 4. **Q1.1**: (a)(b) Layer-wise resilience of different LLMs on different tasks. **Q1.2**: Bit-wise error resilience. (c) Error injection on `K`. (d) Error injection on `O`. **Q1.3**: (e)(f) Sensitivity to errors in different LLM components. **Q1.4**: Relationship between error frequency and magnitude. (g) Resilient components like `K`. (h) Sensitive components like `O`. Given MSD, the error magnitude decreases as the error frequency increases. **Q2.1**: (i)(j) Comparison between the prefill stage and decode stage. **Q2.2**: (k)(l) Impact of error injection across network components: `O`. and `Down` remain highly sensitive. (a)(c)(e)(g)(h) are evaluated with `OPT-1.3B` on LAMDABA; (b)(d)(f) with `LLaMA-2-7B` on WikiText-2; (i)(k) with `LLaMA-2-7B` on X-Sum; (j)(l) with `LLaMA-2-7B` on GSM8K.

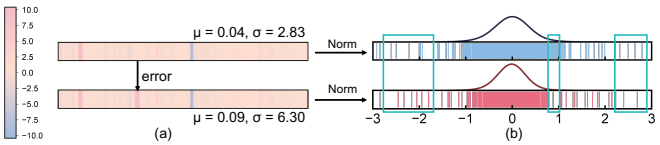


Fig. 5. (a) Data distribution of the pre-norm layer in LLMs, where outliers dominate μ and σ. Injecting larger errors can cause significant skew. (b) Data distribution after normalization is largely affected by the injected error.

between the impacts of a single large error and multiple smaller errors, despite them potentially yielding identical MSD values and having significantly different effects on model performance. Consequently, this section introduces additional metrics—error magnitude (mag) and error frequency ($freq$)—to more accurately represent error statistics. We define mag as $MSD/freq$ in Fig. 4(g)(h), indicating that an increase in $freq$ results in a decrease in mag. Our analysis reveals that resilient components exhibit non-monotonic resilience behavior: both sporadic large errors and frequent smaller errors minimally impact model performance. Conversely, a moderate frequency of medium-sized errors can severely impair performance. However, for sensitive components, acceptable performance is maintained only when numerous smaller errors are introduced.

Discussion. The non-monotonic resilience behavior observed in resilient components illustrates a trade-off between error magnitude and frequency. **Resilient components are tolerant to both sporadic large errors and frequent smaller ones, whereas sensitive components falter even with few large errors**. This finding aligns with earlier observations from Sec. IV-A3. Fig. 6 further delineates a critical region where significant performance degradation occurs.

B. Resilience Study on Decode Stage

LLMs are primarily different from CNNs due to their autoregressive decode stage. In this section, we explore the resilience characterization of the decode stage and differentiate it from the prefill stage.

1) Difference in prefill stage and decode stage: To differentiate resilience between the prefill and decode stages, we inject errors exclusively into the prefill stage, the decode stage, and both stages, respectively. As illustrated in Fig. 4(i)(j), our analysis indicates that performance degradation is consistently more severe during the prefill stage than during the decode stage across both tasks.

Discussion. **The prefill stage exhibits greater sensitivity to errors than the decode stage**. The reason lies in the KV-cache mechanism. During the decode stage, the newly generated query vector interacts with the KV-cache, which is primarily composed during the prefill stage. Consequently, errors during the prefill stage compromise most of the operations involved in generating the next token. In contrast, errors during the decode stage have a lesser impact, as the largely unaffected KV cache drives most of computations for the next token generation.

2) Resilience across compute units: Similar to the experiments described in Sec. IV-A3, we inject errors into various LLM network components and present the outcomes in Fig. 4(k)(l). Our findings confirm that **the sensitive components, specifically `O` and `Down`, continue to be highly vulnerable**, echoing observations from Sec. IV-A3.

V. Statistical Algorithm-Based Fault Tolerance

Building on the insights from Sec. IV, this section introduces our statistical ABFT framework. We integrate it with SAs, focusing on weight stationary (WS) and output stationary (OS) dataflows due to page limitations, though it is also compatible with input stationary (IS) dataflow.

A. Statistical-Based Error Detection Strategy

ReaLM utilizes MSD, error frequency ($freq$), and error magnitude (mag) as key statistics to characterize the error distributions. As shown in Fig. 6, an *acceptable* performance threshold can be empirically established to define the critical region where computational errors significantly impact LLM performance. ReaLM simply ignores errors outside the region.

In terms of the critical region, we find that for resilient components, the boundary of the critical error region is defined by a horizontal line and an inclined line with a slope greater than 1. The horizontal line suggests that resilient components are tolerable to errors below a frequency threshold θ_{freq}, i.e., the vertical coordinate of this line, regardless of their magnitude. For sensitive components, the critical region is bounded by the inclined line and horizontal axis. Note that the slope $a > 1$, the intercept $-b$ of the inclined line, and frequency threshold θ_{freq} are all fitted parameters.

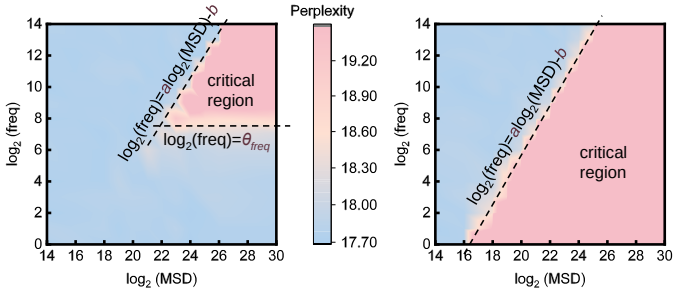


Fig. 6. Our statistical ABFT strategy only corrects errors falling inside the critical region.

We derive several key conclusions to guide our strategy: 1) the horizontal bound indicates that errors exceeding a certain MSD or magnitude all lead to unacceptable performance degradation at θ_{freq}. Therefore, errors larger than the threshold can be counted without distinction; 2) Errors positioned above the inclined line are deemed acceptable. This boundary delineates a clear threshold θ_{mag} beneath which errors, regardless of their frequency, do not significantly degrade the model performance; 3) Assuming that errors are equal, we can derive the upper bound of $\theta_{mag} = b-(a-1)\log_2 MSD$. Errors smaller than θ_{mag} have negligible impact on model performance.

Based on these insights, we assess the influence of errors with error statistics in real-time. Our methodology ignores errors smaller than θ_{mag} due to their minimal impact on performance. Instead, we only count significant errors to calculate the effective error frequency $freq_{eff} = \text{countif}(mag > \theta_{mag})$. We then compare $freq_{eff}$ to the threshold θ_{freq}; if $freq_{eff} > \theta_{freq}$, the recovery is triggered. Otherwise, the GEMM operation is considered safe.

B. Architecture Design of Statistical ABFT

We integrate our statistical ABFT into SA architecture and support both WS and OS dataflow (Fig. 7). We only need to add an extra "statistical unit", as shown in Fig. 7(c), to capture the error statistics and determine the need for recovery, which incurs a very small area overhead. For ease of explanation, we consider the example of computing a GEMM of matrices $W, X \in \mathbb{R}^{n\times n}$ on an $n \times n$ SA, resulting in $Y \in \mathbb{R}^{n\times n}$.

In the WS configuration of the SA (Fig. 7(a)), weights are stored within the processing elements (PEs), and inputs are streamed horizontally from left to right. The array computes partial sums that move vertically downward. A column of PEs is added on the right, storing $e^{\mathrm{T}}W$ for computing $e^{\mathrm{T}}WX$. Meanwhile, a row of adders at the bottom accumulates $e^{\mathrm{T}}Y$. This design minimizes the need for higher bit-width multipliers, as seen in [44], with only minimal area cost. The latency overhead of computing $e^{\mathrm{T}}WX$ and $e^{\mathrm{T}}Y$ is also negligible.

Similarly, for the OS dataflow (Fig. 7(b)), weights traverse horizontally from left to right, while inputs flow vertically from top to bottom. Outputs are generated directly at the intersection of weights and inputs within the PEs. We follow previous works [46] for our implementation. An additional column of adders is integrated on the left side of the array, dedicated to computing $e^{\mathrm{T}}W$. This checksum is then routed to an extra row of PEs positioned at the bottom and propagated rightward to calculate the product $e^{\mathrm{T}}WX$. The outputs y are vertically accumulated across the PEs to derive the output checksum $e^{\mathrm{T}}Y$. This configuration only requires the additional row of PEs to support higher-bit multiplications.

The proposed statistical unit comprises a subtractor, an accumulator, a Log2LinearFunction unit, and a sequence of n buffers

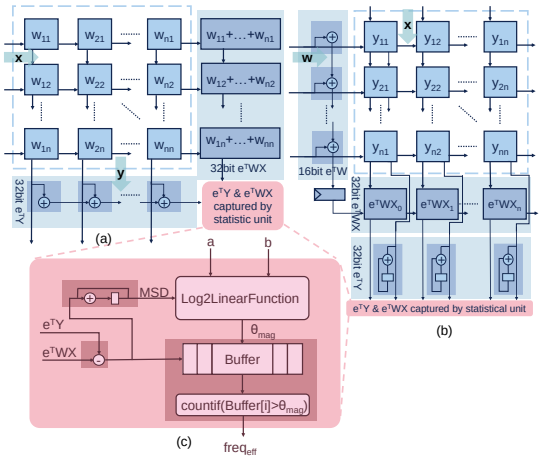


Fig. 7. Architecture design of statistical ABFT on SA: (a) ABFT implementation for WS dataflow; (b) ABFT implementation for OS dataflow; and (c) customized statistical units.

equipped with a comparator-based "countif" unit. The Log2LinearFunction unit is designed to calculate $\theta_{mag} = b-(a-1)\log_2 MSD$. Throughout the computation process, checksums are continuously fed into the statistical unit. They are first subtracted and the resultant differences are accumulated to compute MSD, which is then stored in the buffers. Once the accumulator completes the MSD computation, the data is forwarded to the Log2LinearFunction unit, where θ_{mag} is computed. Ultimately, the "countif" unit evaluates all buffer elements in parallel and counts the frequency of errors exceeding θ_{mag} to determine $freq_{eff}$.

In the decode stage, general matrix-vector multiplication (GEMV) dominates the computation. While batched GEMVs can also be treated as GEMMs, to which our method applies, GEMVs during non-batched inference are typically executed in specialized vector processing units, which fall outside the primary focus of SAs and ABFT techniques discussed in this paper.

Therefore, based on the insights in Fig. 4, the proposed statistical ABFT can enhance LLM resilience by adaptively leveraging captured error statistics. This, in turn, unlocks significant energy-saving opportunities, such as voltage underscaling, even in the presence of errors (Fig. 1), enabling both reliable and efficient LLM inference.

VI. Evaluation

A. Experiment Setup

We first use the lightweight `OPT-1.3B` measuring perplexity on WikiText-2 for rapid evaluation. To demonstrate generality, we also extend the evaluation to the recently released `LLaMA-3-8B` model and a different task, i.e., HellaSwag [54]. Timing errors are simulated by randomly flipping bits in the accumulation results (represented as 32-bit integers) following [7], [22]. The relation between the BER and the operating voltage follows Fig. 1. To determine the parameters for the proposed statistical ABFT, i.e., a, b, θ_{freq}, we inject errors into LLMs for performance evaluation. We empirically set the parameters to allow a 0.3 perplexity increase and a 0.5% accuracy decrease as the acceptable performance degradation.

The RTL for both our proposed and existing ABFT designs is implemented and integrated into SAs with 256 × 256 PEs supporting both WS and OS dataflows. These designs are synthesized using Synopsys Design Compiler with commercial 14nm PDK, where the nominal operating voltage is 0.9V. The clock cycle is set to 500 ps, with a critical path delay of 439 ps, and circuit area and power are obtained from synthesis results. We conduct timing

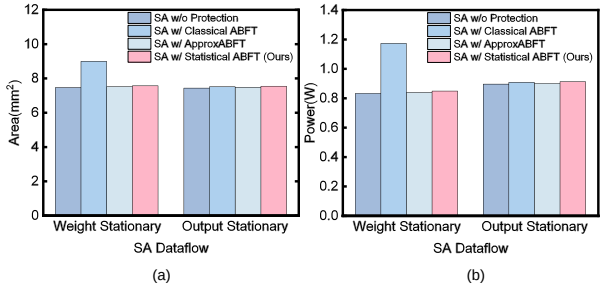


Fig. 8. (a) Area for WS and OS dataflow; (b) Power for WS and OS dataflow.

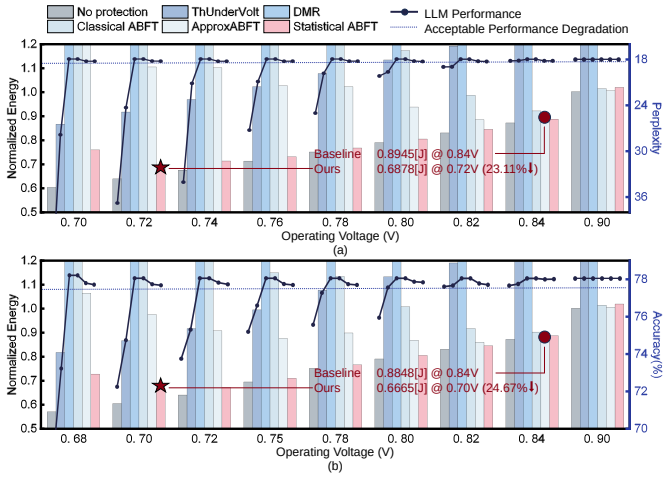


Fig. 9. LLM performance and total energy savings comparison. (a) `OPT-1.3B` on WikiText-2. (b) `LLaMA-3-8B` on HellaSwag. Our method preserves competitive performance with minimal protection overhead, achieving maximal energy savings at 0.72V and 0.70V, respectively.

analysis using Synopsys PrimeTime and HSPICE to capture BERs at reduced voltages, considering the toggle rate during real LLM inference, which is in line with previous chip measurement data [11], [22], [23]. Error recovery is assumed to be performed through recomputation at nominal voltage to ensure correctness, and the total energy consumption is evaluated accordingly.

B. Circuit Overhead Comparison

For both WS and OS dataflows, we analyze the circuit overhead in terms of area and power consumption across no protection, classical ABFT [46], ApproxABFT [45], and statistical ABFT, as depicted in Fig. 8. Compared to unprotected SAs, our design introduces minimal overhead, with only 1.43% area and 1.82% power overhead for WS dataflow, and 1.42% area and 1.79% power overhead for OS dataflow.

C. LLM Performance and Energy Savings

We apply our statistical ABFT to network component `K` in `OPT-1.3B` and `V` in `LLaMA-3-8B` as examples to demonstrate its effectiveness. Fig. 9 compares the total energy overhead and LLM performance for different methods: no protection, ThunderVolt [13], DMR, classical ABFT [46], ApproxABFT [45], and our statistical ABFT, across various operating voltages. Our statistical ABFT maintains comparable LLM performance while significantly reducing the recovery cost and therefore total energy. For `K` in `OPT-1.3B`, our approach achieves a sweet spot at 0.72V, reducing perplexity degradation from 18.54 to 0.29 and saving 23.11% energy compared to prior-art methods. Similarly, for `V` in `LLaMA-3-8B`, it finds a sweet spot at 0.70V, lowering accuracy degradation from 7.64% to 0.47% and saving 24.67% energy.

TABLE II
ENERGY SAVING ACROSS NETWORK COMPONENTS

`OPT-1.3B`			`LLaMA-3-8B`		
Network Component	Optimal Voltage (V)	Energy Saving	Network Component	Optimal Voltage (V)	Energy Saving
`Q`	0.70	28.68%	`Q`	0.71	24.26%
`K`	0.72	23.11%	`K`	0.72	24.17%
`V`	0.65	35.83%	`V`	0.70	24.67%
QK^T	0.67	33.54%	QK^T	0.73	19.46%
`SV`	0.75	17.44%	`SV`	0.66	34.46%
`O`	0.82	3.38%	`O`	0.83	2.40%
`FC1`	0.75	15.01%	`Gate`	0.78	10.21%
			`Up`	0.77	16.56%
`FC2`	0.83	3.14%	`Down`	0.83	3.12%

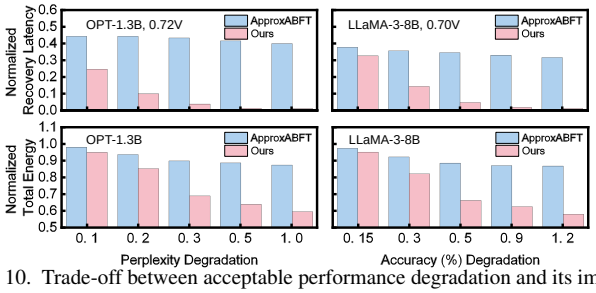


Fig. 10. Trade-off between acceptable performance degradation and its impact on recovery latency and total energy.

D. Energy Savings across LLM Components

We summarize the total energy savings for all network components in Tab. II. As shown, optimal operating voltages (i.e., sweet spots) and resulting benefits vary across components, reflecting the differences in resilience as discussed in Sec. IV. Our method achieves up to 35.83% energy savings on `V` in `OPT-1.3B` and 34.46% on `SV` in `LLaMA-3-8B`. While resilient components demonstrate benefits, sensitive components (`O`, `FC2`, and `Down`) show more limited savings due to their vulnerability to errors, which reduces the potential to exploit inherent resilience.

E. Tradeoff between Performance and Energy Savings

We explore the tradeoff between acceptable performance degradation and its impact on recovery latency and total energy over baseline [45] for `K` in `OPT-1.3B` and `V` in `LLaMA-3-8B`, as shown in Fig. 10. The recovery latency is evaluated at 0.72V and 0.70V, respectively, and the total energy is evaluated at the optimal voltages on each performance degradation constraint. This tradeoff demonstrates our design's flexibility in balancing computational efficiency with reliability requirements.

VII. Conclusion

In this paper, we introduce ReaLM, an algorithm/circuit co-design framework aimed at enhancing the reliability and efficiency of LLM inference. We conduct a systematic error injection experiment to characterize the resilience of LLMs, highlighting particular vulnerabilities in the normalization operation that significantly affect inference reliability. Building on these insights, we develop a statistical ABFT approach featuring a customized, low-cost error detection circuit and an adaptive error correction strategy, effectively reducing error recovery costs. Our evaluations reveal that ReaLM can significantly enhance LLM performance by reducing perplexity degradation from 18.54 to 0.29 with only 1.42% circuit area overhead and 1.79% power overhead. By fully leveraging the inherent resilience of LLMs, ReaLM can achieve up to 35.83% energy savings compared to existing methods while maintaining competitive model performance.

REFERENCES

[1] T. B. Brown *et al.*, "Language models are few-shot learners," in *Proc. NIPS*, pp. 1877–1901, 2020.

[2] W. X. Zhao *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[3] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, pp. 1–12, 2017.

[4] S. Markidis *et al.*, "Nvidia tensor core programmability, performance & precision," in *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pp. 522–531, IEEE, 2018.

[5] H. D. Dixit *et al.*, "Silent data corruptions at scale," *arXiv preprint arXiv:2102.11245*, 2021.

[6] I. Moghaddasi *et al.*, "Dependable dnn accelerator for safety-critical systems: A review on the aging perspective," *IEEE Access*, 2023.

[7] X. Jiao *et al.*, "Clim: A cross-level workload-aware timing error prediction model for functional units," *IEEE Transactions on Computers*, vol. 67, no. 6, pp. 771–783, 2017.

[8] R. Huang *et al.*, "Variability-and reliability-aware design for 16/14nm and beyond technology," in *Proc. IEDM*, pp. 12–4, IEEE, 2017.

[9] F. Libano *et al.*, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.

[10] N. Khoshavi *et al.*, "Shieldenn: Online accelerated framework for fault-tolerant deep neural network architectures," in *Proc. DAC*, pp. 1–6, IEEE, 2020.

[11] D. Ernst *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. MICRO*, pp. 7–18, IEEE, 2003.

[12] P. N. Whatmough *et al.*, "Dnn engine: A 28-nm timing-error tolerant sparse deep neural network processor for iot applications," *IEEE Journal Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.

[13] J. Zhang *et al.*, "Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Proc. DAC*, pp. 1–6, 2018.

[14] N. D. Gundi *et al.*, "Effort: Enhancing energy efficiency and error resilience of a near-threshold tensor processing unit," in *Proc. ASPDAC*, pp. 241–246, IEEE, 2020.

[15] C. Schorn *et al.*, "An efficient bit-flip resilience optimization method for deep neural networks," in *Proc. DATE*, pp. 1507–1512, IEEE, 2019.

[16] S. Kim *et al.*, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *Proc. DATE*, pp. 1–6, IEEE, 2018.

[17] Z. He *et al.*, "Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping," in *Proc. DAC*, pp. 1–6, 2019.

[18] K.-H. Huang *et al.*, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 518–528, 1984.

[19] H. Cherupalli *et al.*, "Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 671–681, 2016.

[20] J. Constantin *et al.*, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *Proc. DATE*, pp. 381–386, IEEE, 2015.

[21] T. Jia *et al.*, "19.4 an adaptive clock management scheme exploiting instruction-based dynamic timing slack for a general-purpose graphics processor unit with deep pipeline and out-of-order execution," in *Proc. ISSCC*, pp. 318–320, IEEE, 2019.

[22] Z. Zhang *et al.*, "Read: Reliability-enhanced accelerator dataflow optimization using critical input pattern reduction," in *Proc. ICCAD*, pp. 1–9, IEEE, 2023.

[23] Z. Wan *et al.*, "Mulberry: Enabling bit-error robustness for energy-efficient multi-agent autonomous systems," in *Proc. ASPLOS*, pp. 746–762, 2024.

[24] X. Jiao *et al.*, "An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations," in *Proc. ICCAD*, pp. 945–950, IEEE, 2017.

[25] G. Li *et al.*, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proc. SC*, pp. 1–12, 2017.

[26] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. DAC*, pp. 1–6, 2018.

[27] A. Mahmoud *et al.*, "Optimizing selective protection for cnn resilience.," pp. 127–138, 2021.

[28] S. Zhang *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[29] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[30] G. Xiao *et al.*, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *Proc. ICML*, pp. 38087–38099, PMLR, 2023.

[31] J. Lin *et al.*, "Awq: Activation-aware weight quantization for llm compression and acceleration," *arXiv preprint arXiv:2306.00978*, 2023.

[32] Z. Yuan *et al.*, "Rptq: Reorder-based post-training quantization for large language models," *arXiv preprint arXiv:2304.01089*, 2023.

[33] F. Libano *et al.*, "Efficient error detection for matrix multiplication with systolic arrays on fpgas," *IEEE Transactions on Computers*, 2023.

[34] T. Marty *et al.*, "Safe overclocking for cnn accelerators through algorithm-level error detection," *IEEE TCAD*, vol. 39, no. 12, pp. 4777–4790, 2020.

[35] B. Salami *et al.*, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 322–329, IEEE, 2018.

[36] B. Sangchoolie *et al.*, "One bit is (not) enough: An empirical study of the impact of single and multiple bit-flip errors," in *Proc. DSN*, pp. 97–108, IEEE, 2017.

[37] Y. He *et al.*, "Fidelity: Efficient resilience analysis framework for deep learning accelerators," in *Proc. MICRO*, pp. 270–281, IEEE, 2020.

[38] Y. S. Hsiao *et al.*, "Silent data corruption in robot operating system: A case for end-to-end system-level fault analysis using autonomous uavs," *IEEE TCAD*, 2023.

[39] G. Papadimitriou *et al.*, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *Proc. ISCA*, pp. 902–915, IEEE, 2021.

[40] Z. Wan *et al.*, "Analyzing and improving fault tolerance of learning-based navigation systems," in *Proc. DAC*, pp. 841–846, IEEE, 2021.

[41] U. K. Agarwal *et al.*, "Resilience assessment of large language models under transient hardware faults," pp. 659–670, IEEE, 2023.

[42] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[43] P. H. Hochschild *et al.*, "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, pp. 9–16, 2021.

[44] S. Bal *et al.*, "A novel fault-tolerant architecture for tiled matrix multiplication," in *Proc. DATE*, pp. 1–6, IEEE, 2023.

[45] X. Xue *et al.*, "Approxabft: Approximate algorithm-based fault tolerance for vision transformers," *arXiv preprint arXiv:2302.10469*, 2023.

[46] M. Safarpour *et al.*, "Algorithm level error detection in low voltage systolic array," *IEEE TCAS II*, vol. 69, no. 2, pp. 569–573, 2021.

[47] R. Isermann, "Process fault detection based on modeling and estimation methods—a survey," *automatica*, vol. 20, no. 4, pp. 387–404, 1984.

[48] D. Paperno *et al.*, "The lambada dataset: Word prediction requiring a broad discourse context," *arXiv preprint arXiv:1606.06031*, 2016.

[49] S. Merity *et al.*, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[50] F. Jelinek *et al.*, "Perplexity—a measure of the difficulty of speech recognition tasks," *The Journal of the Acoustical Society of America*, vol. 62, no. S1, pp. S63–S63, 1977.

[51] S. Narayan *et al.*, "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization," *arXiv preprint arXiv:1808.08745*, 2018.

[52] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, pp. 74–81, 2004.

[53] K. Cobbe *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[54] R. Zellers *et al.*, "Hellaswag: Can a machine really finish your sentence?," *arXiv preprint arXiv:1905.07830*, 2019.

PARO: Hardware-Software Co-design with Pattern-aware Reorder-based Attention Quantization in Video Generation Models

Xinhao Yang*[1,2], Tianchen Zhao*[1,2], Hongyi Wang[1,2], Wenheng Ma[1,2], Shulin Zeng[1,2], Zhenhua Zhu[1], Xuefei Ning[1], Huazhong Yang[1], Yu Wang[1†]
[1]Dept. of EE, BNRist, Tsinghua University [2]Infinigence-AI
[†]Corresponding author: yu-wang@tsinghua.edu.cn

***Abstract*—Transformer-based video generation models have demonstrated significant potential in content creation. However, the current state-of-the-art model employing "3D full attention" encounters substantial computation and storage challenges. For instance, the attention map size for CogVideoX-5B requires 56.50 GB, and generating a video of 49 frames takes approximately 1 minute on an NVIDIA A100 GPU under FP16. Although model quantization has proven effective in reducing both memory and computational costs, applying it to video generation models still faces challenges in preserving algorithm performance while ensuring efficient hardware processing. To address these issues, we introduce PARO, a video generation accelerator with pattern-aware reorder-based attention quantization. PARO investigates the diverse attention patterns of 3D full attention and proposes a novel reorder technique to unify these patterns into a unified "block diagonal" structure. Block-wise mixed precision quantization is further applied to achieve lossless compression under an average bitwidth of 4.80 bits. In terms of hardware, to overcome the limitation of existing mixed-precision computing units could not fully utilize the attention map bitwidth to accelerate QK multiplication, PARO designs an output-bitwidth aware mixed-precision processing element (PE) array through hardware-software co-design. This approach ensures that the mixed-precision characteristics are fully utilized to enhance hardware efficiency in the bottleneck attention computation. Experiments demonstrate that PARO delivers up to 2.71× improvement in end-to-end performance compared to an NVIDIA A100 GPU and achieves up to 6.38∼7.05× speedup over state-of-the-art ASIC-based accelerators on the CogVideoX-2B and 5B models.**

***Index Terms*—Video Generation Model, Mixed-precision Quantization, Hardware Accelerator.**

I. Introduction

Diffusion Transformers (DiTs) [1] and video generation models [2] have garnered significant research interest after the impressive generation quality of OpenAI's SORA [3] in 2024. Previous researches, like OpenSORA [2], utilize "spatial-temporal" attention, which performs attention separately along the spatial and temporal dimensions. More recent models such as CogVideoX [4] adopt "3D full attention", which processes all the spatial tokens for all frames together. Such an attention scheme further enhances algorithm performance, thus achieving state-of-the-art video generation quality. However, such an attention scheme leads to an order of magnitude increase in the token length for attention computation, resulting in excessive storage and computation costs. For example, the token length is 17.8k for the CogVideoX model, and the attention map takes up 56.50 GB for each transformer block. The attention computation accounts for 67.93% of the overall latency on an NVIDIA A100 GPU, which becomes the major bottleneck.

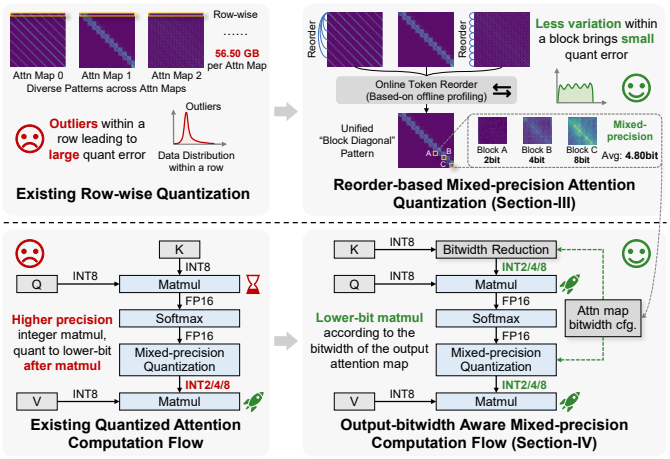


Fig. 1. Overview of the challenges and solutions in PARO.

To mitigate computation and memory consumption, model quantization is a highly effective technique that replaces high-precision floating-point weights and activations with low-bit integers. Low-bit integer multiplications are more hardware-efficient compared to high-precision floating-point operations. Although many existing researches have explored the quantization [5]–[10] of Vision Transformers (ViT) [11], directly applying existing methods to the models with 3D full attention causes notable degradation under INT8 and failure under INT4. Moreover, existing hardware accelerators can only benefit from the low-bit input tensors, failing to efficiently handle the scenario of computing the low-bit (e.g., 2bit) attention map from Q, K embeddings in higher-bits (e.g., 8bit), which is the bottleneck of 3D full attention video generation models.

To address these challenges, we delve into the data distribution characteristics of the attention maps. As illustrated in the upper left part of Fig. 1, we observe diverse "diagonal" patterns across different attention heads and transformer blocks. These patterns arise from the local information aggregation properties inherent in vision feature extraction. Traditional quantization methods assign the same set of quantization parameters to each row of the attention maps. However, the larger elements on the diagonal act as "outliers", leading to excessively large quantization scaling factors. In these cases, the majority of values are forcedly set to near-zero values,

*Both authors contributed equally to this work.

thus losing discriminative abilities, and resulting in large quantization errors. To mitigate this, we propose a reorder-based mixed-precision quantization method. By reordering the QK embeddings along the token dimension, we transform the diverse patterns into a unified "block diagonal" pattern and apply block-wise quantization. This reorder effectively clusters similar values into local blocks, reducing data variation within these blocks and thereby minimizing quantization errors. Moreover, different blocks show diverse attention values, contributing differently to the final attention outputs. Therefore, to compress for lower bits, we introduce mixed-precision quantization, which assigns higher bitwidths to blocks with larger attention values and quantization difficulty, thereby optimizing the overall quantization accuracy and efficiency.

In terms of hardware efficiency, existing mixed-precision computing units perform multiplication and accumulation (MAC) operations based on the bitwidths of the input tensors. These units are unable to leverage the mixed-precision of the output to reduce computational workload. Taking attention computation as a typical example, the attention map, as discussed earlier, can be quantized to mixed precision (e.g., 2/4 bits), while the QKV embeddings remain 8bit. For the $AttnV$ computation, the mixed-precision characteristics of the attention map can be effectively utilized by existing mixed-precision computing units. However, for $QK^\top$, despite the lower precision of the output attention map, existing computing units are constrained by the input bitwidths and must perform the matrix multiplication in 8bits. This limitation prevents them from exploiting the lower bitwidths of the output to further improve computational efficiency. To address this, with a pre-determined lower bitwidth configuration of the output tensor, the matrix multiplication can be performed in reduced bitwidths, aligned with the output precision. Unlike current methods that optimize only $AttnV$ [10], [12], this approach enables the acceleration of both $QK^\top$ and $AttnV$ computations, with negligible accuracy loss. By further exploiting the acceleration opportunities offered by mixed-precision attention maps, this method boosts the efficiency of attention processing, which is the performance bottleneck.

The contributions of PARO are as follows:

- We identify the key limitation of existing quantization methods by investigating the unique properties of "3D full attention" in video generation models, and design a reorder-based mixed-precision quantization method tailored for them.
- To fully harness the performance gain of mixed-precision quantization schemes, we design output-bitwidth aware mixed-precision processing elements (PEs). It fully exploits the capability of PEs to adapt to mixed-precision cases for both QK^T and $AttnV$ in attention computation.
- PARO achieves lossless quantization with an averaged 4.80bit and outperforms state-of-the-art ViT accelerator ViTCoD by 6.38∼7.05×. Evaluated under the same hardware resources, PARO achieves a 1.68∼2.71× speedup to an NVIDIA A100 GPU.

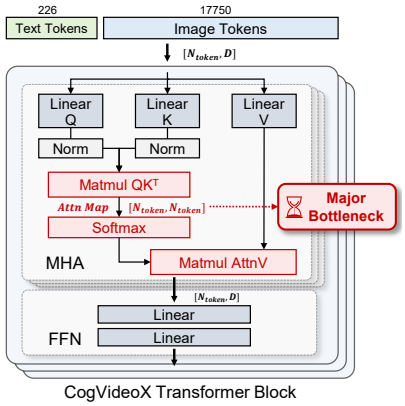


Fig. 2. Illustration of CogVideoX text-to-video generation model, in which the major bottleneck is the attention map related computations.

II. Background and Related Work

A. Video Generate DiTs

As discussed in Sec. I, our primary focus is to optimize the state-of-the-art video generation model, which utilizes a 3D full-attention mechanism, specifically the CogVideoX model. The model structure of CogVideoX is illustrated in Fig. 2, it comprises 42 transformer blocks. Each transformer block consists of two key components: multi-head self-attention (MHA) and feed-forward network (FFN). The computation of the MHA could be formulated as:

$$Q = XW_Q, K = XW_K, V = XW_V, O = \text{Softmax}(\frac{QK^T}{\sqrt{d}})V$$

Given an input token sequence with hidden dimension d, the attention projects it into query Q, key K, and value V matrices through linear layer with weights W_Q, W_K, W_V. Then, softmax is applied to the dot product of Q and K, divided by $\sqrt{d}$ to generate the attention map. Finally, the attention map is multiplied with V to generate the attention output. Subsequently, the FFN processes the attention output O with several full-connected linear layers. In CogVideoX, N_{token} is 17.8k, which is significantly larger than the hidden dimension (d), ranging from 1k to 4k. The computation of the attention map, which has a shape of $[N_{token}, N_{token}]$, becomes the major bottleneck (highlighted in red in Fig. 2).

B. DiT Quantization

Model quantization [13], [14] has been demonstrated as an effective method for model compression. By converting high bitwidth floating-point (FP) data into lower bitwidth integers, it significantly reduces both computational and memory costs. In this approach, weights and activations are quantized within each group G (e.g., tensor-wise or channel-wise). The quantization process approximates the FP value x using an integer representation x_{int} and quantization parameters (scaling factor s, zero point z):

$$x \approx \hat{x} = s(x_{\text{int}} - z)$$

For a group of size g, represented as the vector $x \in \mathbb{R}^g$, all elements share the same quantization parameters (s and z). The quantization operator Q with b-bit is described as:

$$x_{\text{int}} = Q(x; s, z, b) = \text{clamp}(\left\lfloor \frac{x}{s} \right\rceil + z, 0, 2^b - 1)$$

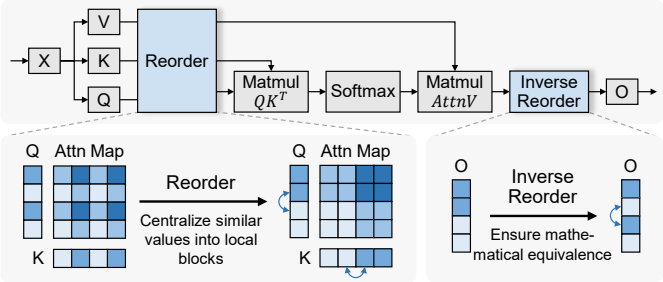


Fig. 3. Detailed description of the reorder process. Q, K, V are reordered along the token dimension to form a unified "block diagonal" pattern. The attention output O is inversely reordered to ensure mathematical equivalence.

In the quantization schemes used in recent work, weights are quantized offline, while activations are dynamically quantized online using a scaling factor $s = \frac{\max(x)-\min(x)}{2^b-1}$, where b is the bit width. The granularity for the quantization of weight and activation in linear layers is "per dimension" and "per token", respectively. For attention map quantization, the granularity is "per-row" for attention values and "per-dimension" for V.

C. Related Work

DiT Quantization. Existing methods [15] for DiT quantization focus on compressing the linear layers in video generation models with spatial-temporal attention. SageAttention and its follow-up work [16]–[18] further advance this by employing 8-bit quantization for QK in attention map computation to accelerate "3D full attention." In our method, we extend quantization to the linear layers, QKV, and attention maps, aiming to minimize expensive floating-point computations.

Hardware Accelerators. Extensive researches [19]–[24], have focused on designing customized accelerators for attention mechanisms. Sanger [25] introduces a locally structured sparse approach to reduce computational overhead. However, this method faces significant challenges when applied to video generation models. ViTCoD [26] addresses this issue with a threshold-based dynamic sparsification technique specifically designed for the attention mechanism in video generation models. It partitions the attention matrix into sparse and dense segments, processing them separately to improve efficiency. Other works [20], [24], [27] have explored strategies such as token pruning and joint video encoding to accelerate entire models. Despite these advancements, two major limitations persist for 3D attention mechanisms. First, existing methods fail to leverage the inherent similarities in 3D attention patterns, limiting opportunities for further optimization and compression. Second, their coarse-grained compression techniques struggle to achieve a balance between maintaining high accuracy and delivering significant acceleration, particularly without requiring additional model retraining.

III. Reorder-based Attention Quantization

A. Reorder-based Block-wise Quantization

In the popular 3D full attention models (e.g., CogVideoX), the token length (N_{token}) is significantly larger than the hidden dimension (d), leading to attention computation dominating the overall computational cost due to its quadratic complexity with respect to token length. As such, quantizing the attention map is critical for resource efficiency. However, attention quantization presents unique challenges. SageAttention, for instance, only quantizes the Q and K tensors, which accelerates only half of the attention computation.

In this paper, we extend this approach by quantizing Q, K, V, and the attention map. We begin by adopting a naive quantization scheme, adopting dynamic min-max quantization, using row-wise grouping for the attention map and dimension-wise grouping for V. However, we observe severe quality degradation under INT8 when solely quantizing the attention map. Additionally, the INT4 results in unreadable noise outputs. To investigate the causes of these issues, we visualize the attention map. As shown in the upper left part Fig. 1, the data within each quantization group (a row in the attention map) exhibits significant variation. A small subset of values, or "outliers", are significantly larger than the rest. The scaling factor (s) is determined based on these outliers, resulting in an excessively large scaling factor for most elements and introducing significant quantization errors. Based on these findings, we aim to reduce quantization errors by minimizing data variation within each quantization group.

We further visualize and analyze the structure of attention maps and observe distinct patterns across different blocks and heads, as illustrated in the upper left part of Fig. 1. An intuitive approach is to leverage these patterns to design specialized quantization groupings that reduce data variation within groups. However, the diversity of these patterns introduces challenges in designing distinct groupings for each case. To address this, we propose transforming these diverse patterns into a unified, hardware-friendly format to enhance hardware efficiency. Through detailed analysis, we find that all patterns essentially represent local aggregation across different dimensions (e.g., the same token across frames or neighboring spatial tokens). By reordering tokens, we can reorganize these patterns into local blocks, as shown in the upper right part of Fig. 1(b). Using block-wise grouping for quantization minimizes variation within each block, thereby reducing quantization error and improving overall performance.

Given an input token of size $N_{\text{frame}} \times N_{\text{width}} \times N_{\text{height}}$, we achieve local block-wise patterns by permuting these dimensions for the QK embeddings through token-level reorder. There are a total of 6 possible reorder plans for each attention head. Notably, the observed patterns remain consistent across different timesteps and input noise or prompts. We select the reorder plan that minimizes quantization error for each head and block offline. The reorder itself is performed online. Due to the compute-bound nature of the model, the overhead introduced by reorder is negligible.

B. Importance-guided Mixed Precision

As shown in the upper right part of Fig. 1, after reordering, similar values are grouped into localized blocks, however, different blocks still exhibit varying value distributions and contribute differently to the final results. These blocks have diverse "quantization sensitivities". Consequently, applying the

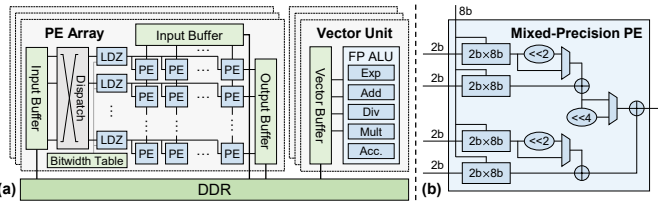


Fig. 4. (a) Overall architecture of PARO. (b) Each PE supports three multiplication mode: 2bit×8bit, 4bit×8bit, 8bit×8bit.

same bitwidth to all blocks results in a suboptimal trade-off between algorithm quality and hardware efficiency. A natural solution is to adopt mixed-precision quantization, allocating higher bitwidths to blocks with higher sensitivity.

Accurately estimating the "quantization sensitivity" is crucial for effective mixed-precision bitwidth allocation. We identify two key characteristics of data distribution that influence quantization sensitivity: (1) **"Block Importance"**: The average absolute values of each block vary, reflecting their relative significance for attention values. (2) **"Quantization Difficulty"**: Within each block, the degree of data variation differs, representing the block's "quantization difficulty". Both highly important blocks and those with larger quantization difficulty should be assigned higher bitwidths. Considering these factors, we propose the following sensitivity metric:

$$\mathcal{S} = (\sum_{i}^{G} x_i)^{\alpha} * (||x_i - x_q||)^{(1-\alpha)}, x \in R^G$$

For each block containing G values, we employ the average attention value as the "importance" and the quantization error to represent the "quantization difficulty". The hyper-parameter α serves to balance the relative emphasis between these two factors. Then, we formulate mixed precision bitwidth allocation as an integer programming problem as follows:

$$\begin{aligned} &\underset{c_{i,b}}{\text{argmin}} \quad \sum_{i=1}^{N} \sum_{b=0,2,4,8} c_{i,b} \cdot \mathcal{S}_{i,b} \\ &\text{s.t.} \quad \sum_{b=0,2,4,8} c_{i,b} = 1, \quad \sum_{i=1}^{N} \sum_{b=0,2,4,8} c_{i,b} \cdot b \leq \mathcal{B} \cdot N, \\ &\quad c_{i,b} \in \{0,1\}, \quad \forall i \in \{1, \cdots, N\}, \forall b \in \{0,2,4,8\} \end{aligned} \tag{1}$$

where N is the number of blocks in the model; $c_{i,b} = 1$ indicates that the i-th block will be quantized to b-bit, and $\mathcal{S}_{i,b}$ is the corresponding sensitivity score, $\mathcal{B}$ indicates the average bitwidth budget.

IV. PARO Architecture

A. Architecture Overview

The overall architecture of PARO is shown in Fig. 4(a), comprising multiple PE arrays, vector units, DDR, and a controller (not shown in the figure). Among these components, the fixed-point PE arrays serve as the primary fixed-point computation unit of the architecture, executing all matrix multiplications since PARO applies quantization to all linear and attention layers. The vector unit features a floating-point arithmetic and logic unit (ALU) and handles floating-point computations outside matrix multiplications (e.g., softmax).

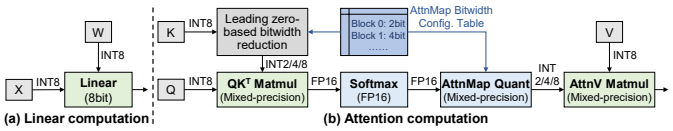


Fig. 5. (a) For linear computations, matrix multiplication operates in 8bit precision. (b) For attention computations, the bitwidth is dynamically reduced based on the attention map's bitwidth configuration.

As the quantization scales for linear and attention layers are in FP16 format, fixed-point accumulation results computed by the fixed-point PE arrays are forwarded to the vector unit. The vector unit converts these results to FP16 format and performs floating-point accumulation, yielding the final FP16 output for matrix multiplication.

B. Mixed-precision PE Array

The structure of each PE in the mixed-precision PE array is shown in Fig. 4(b). Each PE consists of four 2bit×8bit fixed-point multipliers. By adjusting the control signals of the multiplexers, each PE can execute one 8bit×8bit, two 4bit×8bit, or four 2bit×8bit multiplications per cycle.

For linear layers with W8A8 quantization, the PE array operates in 8bit×8bit mode, as shown in Fig. 5(a). In contrast, for attention layers, the bitwidth varies across different blocks in the attention map, while the QKV matrices remain 8bit. Since $QK^\top$ and $AttnV$ each account for half of the computations in attention, the PE array must efficiently handle two scenarios:

- $AttnV$: One input matrix (attention map) is mixed-precision while the other (V) is 8bit.
- $QK^\top$: Both input matrices (Q, K) are 8bits, but the output matrix after softmax is quantized to mixed-precision.

For $AttnV$, computations can be directly mapped to the PE array by dynamically configuring the PE mode based on the bitwidth table of each block in the attention map. For $QK^\top$, performing matrix multiplication at the bitwidth of the inputs is often computationally expensive because the output may be quantized to extremely low bitwidths (e.g., 2bit) after softmax. In extreme cases, such as when the attention map block has a bitwidth of 0, the computation for that block can be skipped entirely. To reduce computational costs, the matrix multiplication can be performed at a lower precision that approximates the same result after softmax quantization.

Inspired by this, PARO adopts an output-bitwidth-aware mixed-precision computation flow to leverage the mixed-precision characteristics of the output attention map. As illustrated in Fig. 5(b), this approach dynamically reduces the bitwidth of K to match the bitwidth of the corresponding output attention map block with little error. Experiments show that this optimization produced no perceptible differences in the quality of the generated video. By performing matrix multiplication at lower precision, the mixed-precision PE Array effectively accelerates $QK^\top$ computation.

Bitwidth reduction of K is implemented using a leading zero (LDZ) unit, integrated beside each PE row, as shown in Fig. 4(a). The LDZ unit identifies the most significant valid bit (MSVB) of the data and outputs the MSVB along with the following $K-1$ bits. The MSVB is the first 1 for positive

TABLE I
ALGORITHM PERFORMANCE OF TEXT-TO-VIDEO GENERATION ON COGVIDEOX PROMPT SET. THE DESCRIPTION OF METRICS IS PROVIDED IN SEC. V-A. WE COMPARE PARO QUANTIZATION METHOD WITH BASELINE METHODS, AND ABLATES EACH TECHNIQUE.

Method	Block-wise	Reorder	Mixed Precision	Bitwidth	FVD-FP16 (↓)	CLIPSIM (↑)	CLIP-Temp (↑)	VQA (↑)	Flicker. (↑)
FP16	-	-	-	16	0.0	0.201	0.997	52.86	97.1
SageAttention [16]	-	-	-	8 (QK-only)	0.08	0.200	0.997	51.25	97.1
Sanger [25]	-	-	-	-	0.22	0.195	0.991	50.84	97.0
Naive INT8	-	-	-	8	0.44	0.201	0.998	49.80	97.0
Block-wise INT8	✓	-	-	8	0.21	0.203	0.997	52.42	97.3
PARO INT8	✓	✓	-	8	0.19	0.203	0.997	50.92	97.2
Naive INT4	-	-	-	4	1.40	0.187	0.997	16.79	96.4
Block-wise INT4	✓	-	-	4	0.40	0.201	0.998	46.53	96.9
PARO INT4	✓	✓	-	4	0.28	0.202	0.998	50.12	96.9
PARO MP	✓	✓	✓	4.80	0.15	0.205	0.998	52.61	96.9

values and the first 0 for negative values. For example, if the LDZ unit is configured as 2bit, the 8bit value `8b00011010` is compressed to `2b11`. After multiplication, the result is restored by left-shifting based on the bit index of MSVB.

The varying bitwidths (0/2/4/8) across different blocks result in differing throughputs for processing these blocks. Therefore, a dispatcher is integrated into each PE array to balance the workloads across blocks. Notably, PARO supports 0bit quantization blocks. The dispatcher bypasses computation for 0bit blocks and maps the next block to the appropriate PE row.

V. EVALUATIONS

A. Evaluation Setup

Software Implementation. We apply the PARO quantization method, referred to as "PARO MP," to the CogVideoX-5B model. In PARO MP, the weights and activations of all linear layers are quantized to INT8. For attention computation, $QKVO$ are quantized to INT8, while the attention map after softmax is quantized using mixed precision (0, 2, 4, 8 bits). Notably, "0 bit" signifies skipping the computation for the corresponding block. Following the official implementation, we generate the 640×480, 49 frames videos, using CogVideoX example prompt set, DDIM [28] 50 steps. We evaluate the generation quality from various aspects following prior literature [15]. The "FVD-FP16" [29] estimate the fidelity of generated videos through measuring the feature space difference between quantized and FP16 outputs. The "CLIPSIM" [30] and "CLIP-Temp" [31] measures the text-video alignment and consistency of clip features across frames. The "VQA" [32] assesses the video quality from the aesthetic and technical perspective. The "Flicker." measures the temporal flickering. higher value denotes less flickering. We present both the statistical and qualitative results in Tab. I and Fig. 7.

Platforms for Comparison. We evaluate our architecture against two state-of-the-art ASIC-based accelerators, Sanger [25] and ViTCoD [26]. The attention map is pruned following the methods described in their papers, ensuring that the generation quality and accuracy remain consistent with PARO. Additionally, we measure the performance of the NVIDIA A100 using CUDA Events for execution time and `nvidia-smi` for power consumption.

TABLE II
AREA AND POWER BREAKDOWN OF PARO.

Component	Config	Area (mm^2)	Power (W)
PE Array	32×32×32 PEs	2.52 (30.8%)	3.60 (32.2%)
	Leading Zero Unit	0.65 (8.0%)	0.78 (7.0%)
	Others	0.39 (4.8%)	0.54 (4.8%)
Vector Unit	Exp/Div/Add/Mult/Acc.	2.79 (34.1%)	4.55 (40.6%)
Buffer	1.5 MB SRAM	1.82 (22.3%)	1.73 (15.4%)
Total	TSMC 12nm	8.17 (100%)	11.20 (100%)

Hardware Implementation. PARO is implemented using RTL for its hardware components and synthesized under TSMC 12nm process at 1 GHz using Synopsys Design Compiler to evaluate area and power. The hardware components are detailed in Tab. II. The DDR bandwidth of PARO is 51.2 GB/s. We use CACTI 7 [33] to assess the on-chip buffer in PARO. To ensure a fair comparison, we develop a cycle-accurate simulator to model the behavior and performance of both PARO and the baseline accelerators under the same hardware resource constraints. Furthermore, we align PARO's hardware resources (e.g., peak computing performance, memory bandwidth, frequency, on-chip buffer size, etc.) with those of the NVIDIA A100 GPU to evaluate our performance gains over modern GPUs, denoted as "PARO-align-A100".

B. Evaluation Results

Algorithm Performance. We evaluate the algorithm performance (generated video quality) against baseline methods, as detailed in Tab. I and Fig. 7. From both statistical and qualitative perspectives. "PARO-MP 4.80bit" demonstrates comparable or superior algorithm performance to baseline model compression techniques like "SageAttention" and "N:M sparse". It notably outperforms "PARO INT4" in all aspects and achieves similar algorithm performance to "PARO INT8". Compared to the FP16 baseline, "PARO-MP 4.80bit" incurs only a 0.25 VQA decrease and a 0.2 flickering score loss, which is hardly noticeable, as presented in Fig. 7.

Ablation Studies for Algorithm. We perform ablation studies for each component of the PARO quantization method, as shown in Tab. I. The "Naive" represents the naive round-to-nearest quantization scheme, and both "Naive INT8" and "Naive INT4" exhibit significant algorithm performance degra-

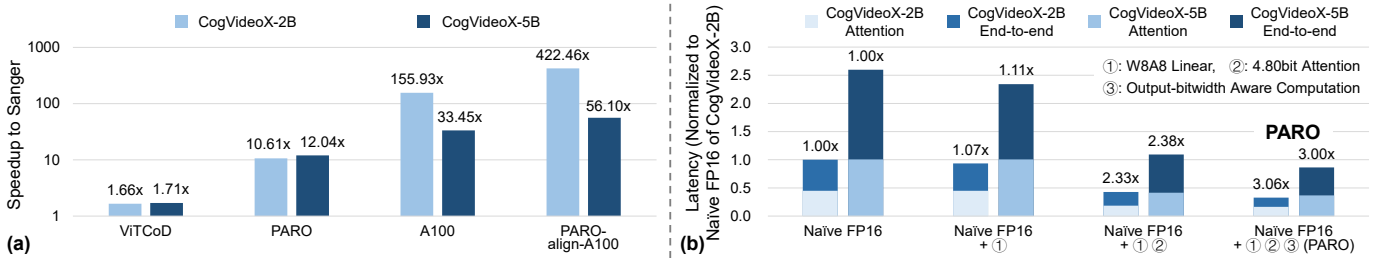


Fig. 6. (a) End-to-end speedup on CogVideoX-2B/5B, normalized to Sanger [25]. (b) The ablation study on different optimizations in PARO.

dation across multiple metrics. Incorporating block-wise quantization notably enhances visual quality under INT4, improving from 16.79 to 46.53. Further integrating reorder to balance the data distribution within blocks further elevates the generation quality. Ultimately, by adopting mixed precision quantization, "PARO-MP" achieves algorithm performance on par with INT8 and FP16, averaging 4.80bit.

Analysis of Attention Patterns. We visualize the attention maps before and after reorder in Fig. 8. The reorder effectively unifies diverse patterns into quantization- and hardware-friendly "block diagonal" pattern. It highlights that different attention heads conduct local aggregation along various dimensions, such as "frame" and "height" in the examples shown.

End-to-end Speedup. As shown in Fig. 6(a), we normalize the performance of PARO, other accelerators and GPUs to Sanger. Compared with Sanger and ViTCoD, PARO obtains performance improvements of 10.61/12.04× and 6.38/7.05× on CogVideoX-2B/5B, respectively. Although the A100 achieves higher end-to-end performance than PARO, this is mainly due to the higher peak performance and memory bandwidth of A100. When PARO uses the same hardware parameters as the A100, it achieves a performance improvement of 1.68/2.71× compared to the A100. This is primarily due to the mixed-precision attention quantization and our output-bitwidth aware mixed-precision computation flow.

Ablation Study for Performance. The breakdown of PARO's performance gains are illustrated in Fig. 6(b). The naive FP16 version includes no optimizations in PARO. Building on this baseline, we introduced W8A8 linear layer quantization, 4.80bit attention layer quantization, and output-bitwidth aware mixed-precision computing unit. For the 2B/5B models, linear layer quantization achieved a 1.07/1.11× speedup. Adding attention mixed-precision quantization further improved the speedup to 2.33/2.38×. Finally, incorporating the output-bitwidth aware mixed-precision optimization increased the speedup to 3.06/3.00×. Among these optimizations, 4.80bit attention quantization contributed the most significant improvement, as 3D full attention is the performance bottleneck.

Reorder Overhead. We evaluate the overhead of introducing the reorder for $QKVO$ during inference in PARO. Experimental results show that for the CogVideoX-2B/5B models, the reorder operation accounts for only 1.26% and 1.07% of the end-to-end latency, respectively. This is because the data size of the $QKVO$ matrices is only 0.36% of the attention map, making the overhead negligible in the highly compute-bound attention computations.

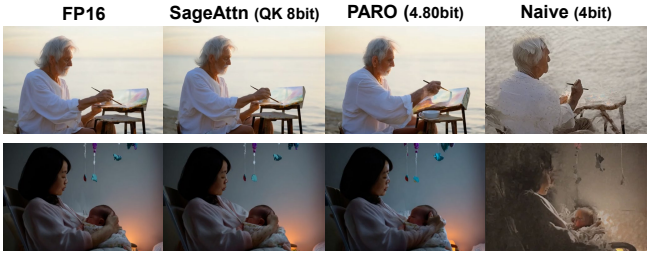


Fig. 7. Comparison of generated videos for different quantization methods. The PARO quantization method could generate videos without visual difference with FP16 generated videos with an averaged of 4.80bit.

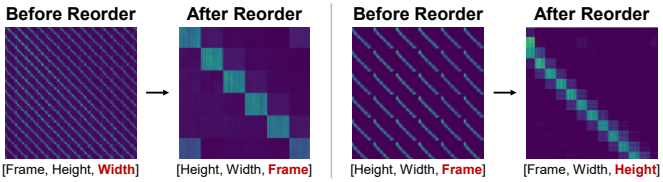


Fig. 8. Visualization of attention pattern before and after reorder. The reorder unifies different patterns into "block diagonal" pattern.

Energy Efficiency. Thanks to our hardware-friendly attention quantization and efficiently mixed-precision PE array, PARO achieves energy efficiency of 3.46/3.61 TOPS/W on the CogVideoX-2B/5B models, respectively, which are 4.86/6.43× higher compared to NVIDIA A100 GPU.

VI. Conclusions

We present PARO, a video generation accelerator that leverages hardware-software co-design with pattern-aware reorder-based attention quantization. PARO examines 3D full attention patterns and introduces a reorder technique to consolidate them into a block diagonal structure. Block-wise mixed precision quantization achieves lossless compression with an average bitwidth of 4.80. The hardware integrates a dynamically reconfigurable mixed-precision PE array, ensuring full utilization of mixed-precision characteristics. Experimental results demonstrate PARO delivers up to 2.71× improvement in end-to-end performance compared to an NVIDIA A100 GPU and achieves up to 6.38∼7.05× speedup over state-of-the-art ASIC-based accelerators on the CogVideoX-2B and 5B models.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 62325405, 62104128, U21B2031, 62204164), the National Key R&D Program of China (2023YFB4502200), Tsinghua EE Xilinx AI Research Fund, Tsinghua-Meituan Joint Institute for Digital Life, and Beijing National Research Center for Information Science and Technology (BNRist).

REFERENCES

[1] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4195–4205.

[2] HPC-AI, "Open-Sora," https://github.com/hpcaitech/Open-Sora, 2024.

[3] OpenAI, "Video generation models as world simulators," https://openai.com/index/video-generation-models-as-world-simulators/, 2024.

[4] Z. Yang, J. Teng, W. Zheng, M. Ding, S. Huang, J. Xu, Y. Yang, W. Hong, X. Zhang, G. Feng *et al.*, "Cogvideox: Text-to-video diffusion models with an expert transformer," *arXiv preprint arXiv:2408.06072*, 2024.

[5] X. Huang, Z. Shen, and K.-T. Cheng, "Variation-aware vision transformer quantization," *arXiv e-prints*, pp. arXiv–2307, 2023.

[6] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization," in *European conference on computer vision*. Springer, 2022, pp. 191–207.

[7] Z. Li and Q. Gu, "I-vit: Integer-only quantization for efficient vision transformer inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 065–17 075.

[8] Y. Li, S. Xu, B. Zhang, X. Cao, P. Gao, and G. Guo, "Q-vit: Accurate and fully quantized low-bit vision transformer," *Advances in neural information processing systems*, vol. 35, pp. 34 451–34 463, 2022.

[9] Z. Li, J. Xiao, L. Yang, and Q. Gu, "Repq-vit: Scale reparameterization for post-training quantization of vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 227–17 236.

[10] Y. Lin, T. Zhang, P. Sun, Z. Li, and S. Zhou, "Fq-vit: Post-training quantization for fully quantized vision transformer," *arXiv preprint arXiv:2111.13824*, 2021.

[11] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[12] P. Dong, L. Lu, C. Wu, C. Lyu, G. Yuan, H. Tang, and Y. Wang, "Packqvit: Faster sub-8-bit vision transformers via full and packed quantization on the mobile," *Advances in Neural Information Processing Systems*, vol. 36, pp. 9015–9028, 2023.

[13] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[14] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[15] T. Zhao, T. Fang, E. Liu, R. Wan, W. Soedarmadji, S. Li, Z. Lin, G. Dai, S. Yan, H. Yang *et al.*, "Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation," *arXiv preprint arXiv:2406.02540*, 2024.

[16] J. Zhang, J. Wei, P. Zhang, J. Zhu, and J. Chen, "Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration," in *International Conference on Learning Representations (ICLR)*, 2025.

[17] J. Zhang, H. Huang, P. Zhang, J. Wei, J. Zhu, and J. Chen, "Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization," 2024. [Online]. Available: https://arxiv.org/abs/2411.10958

[18] J. Zhang, C. Xiang, H. Huang, H. Xi, J. Wei, J. Zhu, and J. Chen, "Spargeattn: Accurate sparse attention accelerating any model inference," 2025.

[19] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "A^ 3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 328–341.

[20] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.

[21] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 692–705.

[22] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 14–26.

[23] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 599–615.

[24] P. Dong, M. Sun, A. Lu, Y. Xie, K. Liu, Z. Kong, X. Meng, Z. Li, X. Lin, Z. Fang *et al.*, "Heatvit: Hardware-efficient adaptive token pruning for vision transformers," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 442–455.

[25] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[26] H. You, Z. Sun, H. Shi, Z. Yu, Y. Zhao, Y. Zhang, C. Li, B. Li, and Y. Lin, "Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 273–286.

[27] Z. Song, C. Qi, F. Liu, N. Jing, and X. Liang, "Cmc: Video transformer acceleration via codec assisted matrix condensing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 201–215.

[28] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[29] T. Unterthiner, S. Van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly, "Fvd: A new metric for video generation," 2019.

[30] C. Wu, L. Huang, Q. Zhang, B. Li, L. Ji, F. Yang, G. Sapiro, and N. Duan, "Godiva: Generating open-domain videos from natural descriptions," *arXiv preprint arXiv:2104.14806*, 2021.

[31] P. Esser, J. Chiu, P. Atighehchian, J. Granskog, and A. Germanidis, "Structure and content-guided video synthesis with diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 7346–7356.

[32] H. Wu, E. Zhang, L. Liao, C. Chen, J. Hou, A. Wang, W. Sun, Q. Yan, and W. Lin, "Exploring video quality assessment on user generated contents from aesthetic and technical perspectives," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 20 144–20 154.

[33] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

SynGPU: Synergizing CUDA and Bit-Serial Tensor Cores for Vision Transformer Acceleration on GPU

Yuanzheng Yao[1], Chen Zhang[1]*, Chunyu Qi[1], Ruiyang Chen[1], Jun Wang[2], Zhihui Fu[2],
Naifeng Jing[1], Xiaoyao Liang[1], and Zhuoran Song[1]*
[1]Shanghai Jiao Tong University, Shanghai, China
[2]OPPO Research Institute
[1]{yzh_yao, songzhuoran}@sjtu.edu.cn

Abstract*—Vision Transformers (ViTs) have demonstrated remarkable performance in computer vision tasks by effectively extracting global features. However, their self-attention mechanism suffers from quadratic time and memory complexity as image resolution or video duration increases, leading to inefficiency on GPUs. To accelerate ViTs, existing works mainly focus on pruning tokens based on value-level sparsity. However, they miss the chance to achieve peak performance as they overlook the bit-level sparsity. Instead, we propose Inter-token Bit-sparsity Awareness (IBA) algorithm to accelerate ViTs by exploring bit-sparsity from similar tokens. Next, we implement IBA on GPUs that synergize CUDA and Tensor Cores by addressing two issues: firstly, the bandwidth congestion of the Register File hinders the parallel ability of CUDA and Tensor Cores. Secondly, due to the varying exponent of floating-point vectors, it is hard to accelerate bit-sparse matrix multiplication and accumulation (MMA) in Tensor Core through fixed-point-based bit-level circuits. Therefore, we present SynGPU, an algorithm-hardware co-design framework, to accelerate ViTs. SynGPU enhances data reuse by a novel data mapping to enable full parallelism of CUDA and Tensor Cores. Moreover, it introduces Bit-Serial Tensor Core (BSTC) that supports fixed- and floating-point MMA by combining the fixed-point Bit-Serial Dot Product (BSDP) and exponent alignment techniques. Extensive experiments show that SynGPU achieves an average of 2.15× ∼ 3.95× speedup and 2.49× ∼ 3.81× compute density over A100 GPU.

I. Introduction

Recently, attention-based models in the form of Vision Transformers (ViTs) [1] have achieved exceptional performance in the field of computer vision (CV) tasks such as image classification [2], object detection [3], and video recognition [4]. Compared to traditional convolutional neural network (CNN) models, ViTs leverage their ability to extract global features effectively, offering both enhanced performance and greater flexibility in diverse applications.

However, the achievements of ViTs are at the cost of inefficiency. As is well known, the self-attention mechanism in ViTs exhibits quadratic time and memory complexity with respect to the number of input tokens. Consequently, as image resolution or video duration increases, the computational performance of ViTs on GPUs deteriorates significantly. To address this limitation, various approaches have been proposed [5]–[8], focusing on reducing value-level redundancy by exploiting token similarity across image patches or video frames. Beyond value sparsity, our analysis reveals that bit-level redundancy offers an opportunity to further enhance the computational efficiency of ViTs.

As illustrated in Fig. 1, adjacent image pixels often manifest similar values, leading to many 0-bits in their differences. Building on this observation, we propose an Inter-token Bit-sparsity Awareness (IBA) algorithm that calculates the distances between input tokens and computes token differences by subtracting similar pairs. These token differences demonstrate increased bit-level sparsity, which can be effectively leveraged to accelerate computations. To implement this strategy on GPUs, we utilize the two inherently separate computing cores within the GPU: CUDA Cores and Tensor Cores, ***utilizing their complementary functions to enhance the efficiency of ViTs by harnessing bit-level sparsity***.

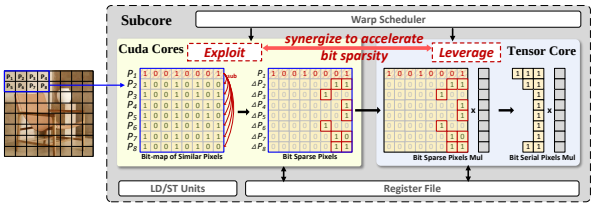
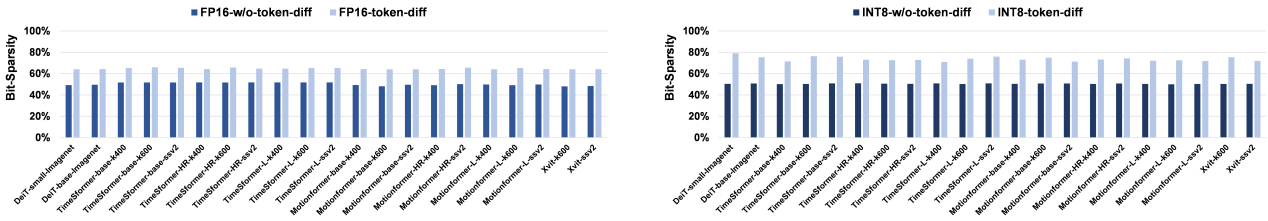


Fig. 1. Synergy mechanism of SynGPU.

The key insights of this scheme lie in two points: *first, leveraging the high-precision capabilities of CUDA Cores to exploit bit-level sparsity among similar tokens without compromising accuracy; second, utilizing Tensor Cores, optimized for high-throughput matrix multiplication and accumulation (MMA) operations, to enable bit-sparse acceleration.* However, achieving efficient synergy between CUDA Cores and Tensor Cores presents two critical challenges. Firstly, CUDA Cores and Tensor Cores compete for Register File (RF), which can result in RF bandwidth congestion, hindering their ability to operate in full parallelism. Secondly, Tensor cores implement the floating-point MMA by executing dot product (DP) operations in parallel. It is non-trivial to design bit-level circuits to support both fixed-point and floating-point DP due to the variability in the exponents of floating-point values within the vectors.

* Zhuoran Song and Chen Zhang are corresponding authors.
This work is partly supported by the National Natural Science Foundation of China (Grant No.62202288)



(a) Bit-level sparsity in FP16

(b) Bit-level sparsity in INT8

Fig. 2. Bit-level sparsity in FP16/INT8 between token differences and token without differences

In this paper, we propose SynGPU, an algorithm-hardware collaborative framework, that synergizes CUDA Cores and Tensor Cores for accelerating ViTs on the GPU. Specifically, to address the first challenge, SynGPU includes a novel data mapping scheme that enhances data reuse in Tensor Cores. This approach significantly reduces the frequency of RF accesses by Tensor Cores, thereby freeing up RF resources for CUDA Cores. Moreover, to tackle the second challenge, we develop a Bit-Serial Dot Product (BSDP) algorithm along with the Bit-Serial Tensor Core (BSTC), which aligns the exponents of floating-point values within each vector and repackages the results of DP operations. This design enables SynGPU to support both floating-point and fixed-point bit-sparse operations effectively.

In summary, our contributions are as follows:

- We propose SynGPU, an algorithm-hardware co-design framework that leverages the synergy between CUDA Cores and Tensor Cores on GPUs to accelerate bit-sparse ViTs. By utilizing CUDA Cores to exploit bit-level sparsity derived from token similarity and coordinating with Tensor Cores to perform bit-sparse MMA, SynGPU achieves substantial performance improvements for ViTs. To the best of our knowledge, SynGPU is the first framework specifically designed for bit-level ViTs acceleration on GPUs.
- At the algorithmic level, we propose two algorithms: IBA and BSDP. IBA is designed to extract bit-level sparsity among similar tokens, and BSDP focuses on combining both fixed-point and floating-point MMA by aligning the exponents of floating-point vectors to enable efficient bit-level sparse floating-point MMA.
- At the hardware level, we make two contributions. Firstly, we design a new data mapping that resolves the issue of bandwidth congestion, enabling full parallelism between CUDA cores and tensor cores. Secondly, we develop BSTC that is capable of accelerating bit-sparse MMA operations in fixed- and floating-point formats.

II. Background and Motivation

A. Vision Transformer

Inspired by the remarkable performance of Transformer [9] on NLP tasks, numerous efforts have been made to adapt Transformers for visual applications. Among them, ViTs stands out with its concise design and powerful capabilities in image and video tasks. ViTs divide images or video frames into patches, which are then converted into tokens through a Patch-Embedding Layer, enabling them to be directly fed into the Transformer.

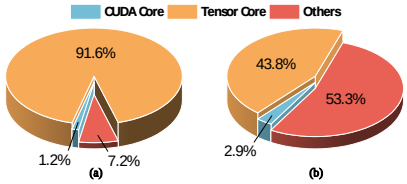


Fig. 3. The occupancy of RF bandwidth (a) and activation time (b) of CUDA Cores and Tensor Cores on A100 when executing ViTs.

Several accelerator architectures [5]–[7] have been proposed to optimize ViTs. For instance, DynamicViT [6] designs a dedicated dynamic prediction module to predict and prune redundant tokens. ViTCoD [5] employs an autoencoder module to reorder attention maps, creating denser or sparser fixed patterns to balance ViTs' workload. However, these advancements primarily focus on optimizing value-level sparsity and neglect the potential bit-level sparsity among similar tokens.

B. Ampere GPU

In recent years, the rapid advancement of neural networks and their substantial computational requirements have driven significant evolution in GPU structure and performance. The Nvidia A100 [10] serves as a prime example, featuring multiple Streaming Multiprocessors (SMs). Each Subcore of SM comprises several CUDA cores and a Tensor Core, sharing a common RF. Specifically, CUDA cores are utilized for high-precision general-purpose computations, whereas Tensor Cores are designed for high-throughput MMA operations, incorporating numerous DP units that support various fixed- and floating-point formats.

C. Motivation

While current research on ViT accelerators mainly focuses on value-level sparsity, our experimental results display a more promising form of bit-level sparsity among similar tokens. As demonstrated in Fig. 2, we run the implementation of various representative ViT models and subtract similar tokens within the attention map. As a result, we observe a significantly higher number of 0-bits for both FP16 and INT8, with bit sparsity increasing from 50.19% to 65.98% and from 50.48% to 75.82%, respectively.

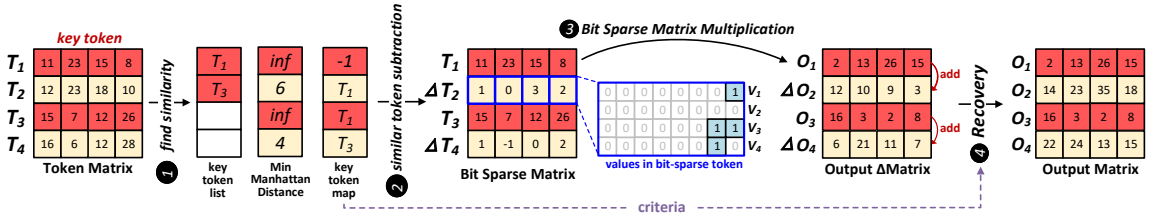


Fig. 4. Overview of the Inter-token Bit-sparsity Awareness algorithm.

Based on such observations, we intend to extract and leverage the bit-level sparsity to accelerate ViTs by collaborating with CUDA Cores and Tensor Cores. However, as shown in Fig. 3, when Tensor Core operates at full capacity with the RF bandwidth occupancy of 91.6%, CUDA Cores are mostly idle due to the competitive access to RF. Furthermore, different exponents of floating-point vectors impede accelerating MMA in Tensor Cores with fixed-point bit-level designs [11], [12]. Despite the effort made by Bitlet [13] to support bit-level floating-point formats, it leads to insufficient hardware utilization in lower-bit fixed-point computations. To address such inefficiencies, we propose SynGPU, an algorithm-hardware co-design framework that facilitates concurrent utilization of CUDA Cores and Tensor Cores to accelerate ViTs.

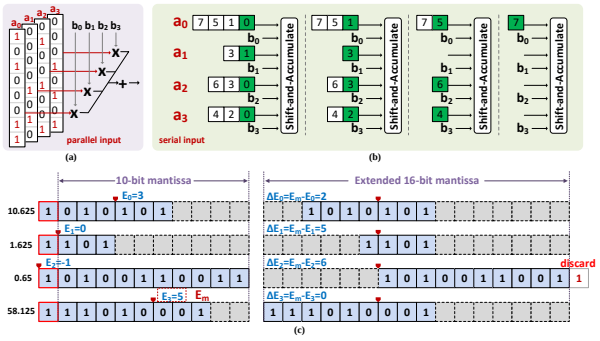


Fig. 5. Details of: (a) Parallel Dot Product. (b)Bit-Serial Dot Product. (c) Exponent alignment algorithm

III. Algorithm

A. Inter-token Bit-sparsity Awareness

In this section, we introduce Inter-token Bit-sparsity Awareness (IBA), an algorithm aimed at exploiting bit-level sparsity within ViTs. As illustrated in Fig. 4, in **step ❶**, we select a subset of key tokens at a regular interval, while the remaining tokens are non-key ones. Next, we compute the minimum Manhattan Distance as the criteria to locate the most similar key token to each of the non-key tokens and record them in a key token map. In the given example, we select key tokens at an interval of 2 and find that the most similar tokens to T_2 and T_4 are T_1 and T_3. **Step ❷** then subtracts the similar token pairs to acquire the Bit Sparse Matrix composed of numerous bit-sparse difference tokens. The blue-highlighted part displays the binary expressions of the values in the difference token $\Delta T_2 = T_2 - T_1$, exhibiting that the similarity between tokens leads to more redundant 0-bits. Subsequently, **step ❸** executes bit-sparse MMA operations to obtain the Output ΔMatrix efficiently. Finally, based on the linearity of MMA operations, **step ❹** utilizes the information in the key token map to add the outputs of O_1, O_3 to ΔO_2 and ΔO_4, respectively, thereby recovering the Output Matrix losslessly.

B. Bit-Serial Dot Product

The main challenge of performing bit-sparse MMA is to support both floating-point and fixed-point formats, particularly given the varying exponents of floating-point vectors. Since floating-point multiplication consists of two main operations: exponent addition and mantissa multiplication, with the latter following the same mechanism as fixed-point multiplication, we propose Bit-Serial Dot Product (BSDP), an algorithm that combines fixed-point bit-serial multiplication with floating-point exponent alignment to address the challenge.

Fixed-point BSDP. The conventional DP operation depicted in Fig. 5(a) computes all bits in parallel. To avoid unnecessary 0-bits, as illustrated in Fig. 5(b), we need to transform the multiplication process into a bit-serial input mode by encoding the matrix elements into bit index format. This approach converts multiplication into a series of shift-and-accumulate operations, effectively skipping the 0-bits.

Floating-point BSDP. A floating-point operand is expressed as $X = 2^E \times M$, where M is the mantissa, and E is the exponent (we omit the sign bit for simplicity). The floating-point BSDP consists of three steps: exponent alignment, mantissa multiplication, and repackage. As illustrated in Fig. 5(c), we first align the exponents of all elements in the vector to the largest E_m by right-shifting the mantissa. Additionally, since the right-shifting may cause overflow and subsequent loss of precision, we extend the 10-bit mantissa to 16 bits:

$$X_i = 2^{E_i} \times M_i = 2^{E_m} \times (M_i \gg \Delta E_i) \tag{1}$$

where $\Delta E_i = E_m - E_i$ are the right-shifting bits for mantissa. As for floating-point BSDP $\vec{a} \times \vec{b}$, we separately align both exponents and then multiply the mantissa following the same way as fixed-point BSDP. Subsequently, we repackage the result into the standard floating-point format by adding the unified exponents by Eqn. (2):

$$\begin{aligned} dp &= \sum_{i=0}^{m} [2^{E_i^a} \times M_i^a] \times [2^{E_i^b} \times M_i^b] \\ &= \sum_{i=0}^{k} 2^{E_m^a + E_m^b} \times (M_i^a \gg \Delta E^a) \times (M_i^b \gg \Delta E^b) \end{aligned} \tag{2}$$

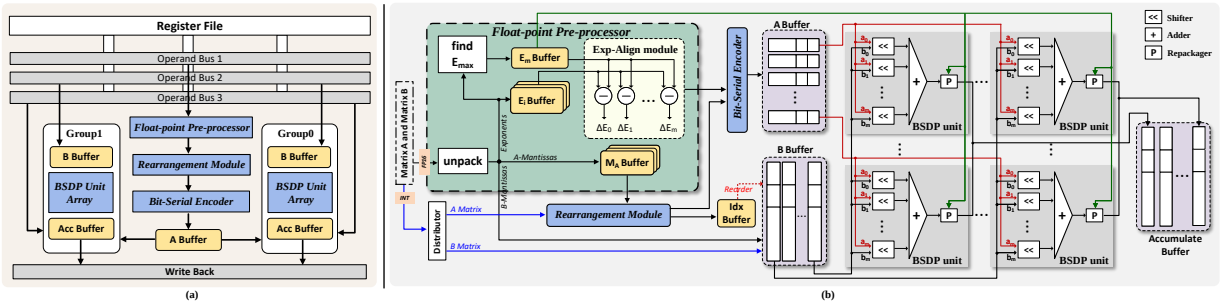


Fig. 6. Architecture of Bit-Serial Tensor Core: (a) Overview of BSTC. (b) Details of BSTC.

To reduce redundant computations, we further combine the two right shifts in alignment operations and the left shift in fixed-point BSDP to obtain the final result of floating-point BSDP operations, as shown in Eqn. (3):

$$dp = \sum_{i=0}^{k} \sum_{j=0}^{l} 2^{E_m^a + E_m^b} \times [M_i^b \ll (I_j - \Delta E^a - \Delta E^b)] \quad (3)$$

where I_j denotes the index of the 1-bit positions in the elements of $\vec{a}$.

IV. Architecture

A. Overview

The SynGPU architecture comprises two key designs. First, we introduce a data mapping scheme to solve the RF bandwidth congestion issue between CUDA Cores and Tensor Cores, enabling parallel processing capabilities for both cores. Second, we propose the Bit-Serial Tensor Core (BSTC), which is capable of computing both fixed- and floating-point MMA. Additionally, to maximize the efficiency of accelerating bit sparse MMA in BSTC, we design a rearrangement module to balance the matrix bit-sparsity.

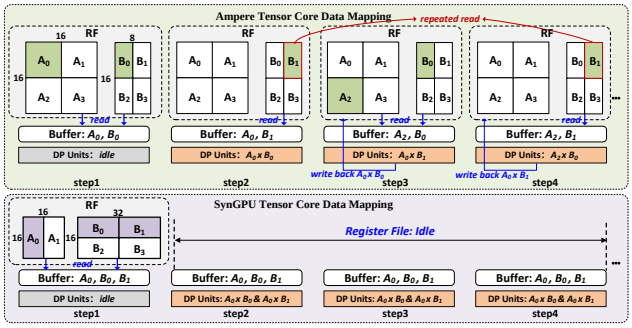


Fig. 7. Details of data mapping.

B. Data Mapping

When performing a 16 × 16 × 8 matrix operation in the original Tensor Core, it is separated into 8 steps of 8 × 8 × 4 MMA process (only 4 steps are given in Fig. 7), operating in a pipelined manner, with each process consisting of 3 parts: reading from RF, calculation, and writing back to RF. As depicted in the figure, in step 1, the Buffer reads A_0 and B_0, while the calculation and writing back correspond to step 2 and step 3. Unfortunately, the above execution consumes most of the RF's memory bandwidth, causing the CUDA cores to remain idle during Tensor Core operations. Additionally, this data flow of Tensor Core also leads to repeated data read-write operations, such as B_1 in steps 2 and 4, resulting in unnecessary overhead.

To address this issue, we propose to increase the capacity of the Tensor Core's data buffer, enabling it to read all required data for the current computation at once during step 1. By converting the bits of elements from parallel to serial processing in BSDP operations, the RF becomes idle during subsequent DP computation steps, allowing it to be utilized by CUDA cores.

Moreover, as the serial process decreases parallelism in the Tensor Core, we expand the dimension of the B matrix to ensure that the throughput does not decrease. Specifically, due to the 16-bit extended mantissa detailed in III-B, the worst-case scenario requires up to 16× cycles to compute MMA than Ampere Tensor Core. Thus, the expansion of the B matrix is set as 16 × 32. Additionally, to handle the repeated read-write operations, we split the A matrix into two groups, each with a size of 16 × 8, to compute $A_0 \times B_0$ and $A_0 \times B_1$ simultaneously.

C. Bit-Serial Tensor Core Design

As mentioned in III-B, to achieve bit-sparse MMA, floating-point numbers are required to be pre-processed for exponent alignment, while fixed-point numbers are able to directly perform BSDP operations. Therefore, as illustrated in Fig. 6 the architecture of BSTC mainly consists of two modules: the BSDP Array and the Floating-point Pre-processor.

BSDP Array. To implement bit-sparse matrix A × matrix B, we replace the DP Units within the Tensor Core with BSDP Units and arrange them in a grid array to concurrently work on performing a number of BSDP operations, where the (x, y)-th Unit executes BSDP between the x-th row of A and the y-th column of B. Each BSDP Unit consists of several shifters, an adder, and a repackage module to accomplish the fixed-point BSDP described in III-B. Within each cycle, a vector from the B Buffer and a serial of bit index code of matrix A from the A Buffer are fetched to perform shift-and-accumulate operations, which are repeated until the entire vector DP is finished.

Floating-point Pre-processor. The primary function of the Floating-point Pre-processor is to unpack floating-point elements in matrices and align the exponents in terms of

vectors. Initially, the unpacking process splits and extracts the exponents and mantissa from the floating-point data, and then finds the largest exponents as E_m, respectively storing all the segments into E_i Buffer, M_A Buffer, and E_m Buffer. Notably, since the B-Mantissas do not necessitate bit-serial encoding, they can be directly deposited in the B Buffer for subsequent DP computations. Next, the Exp-Align module works on subtractions $\Delta E_i = E_m - E_i$ as mentioned in Eqn. (3). Subsequently, based on the principles discussed in III-B, the Bit-Serial Encoder combines ΔE_i and M_A for encoding into bit-serial forms and stores them in the A Buffer for BSDP operation with B-mantissas. Finally, the repackage module that interfaces with the E_m Buffer is added at the end of each Unit to normalize the DP results to standard floating-point format, merge $E_m^a + E_m^b$ into the final exponent, and store them into the Accumulate Buffer before writing back.

D. Rearrangement Module Design

Despite the capability of BSTC in IV-C to facilitate the acceleration of fixed- and floating-point bit-sparse MMA operations, we discover that a primary bottleneck limiting the compute performance is the bit imbalance in the elements of matrix A. Apparently, the speed of BSDP operation executed in a BSDP Unit shown in Fig. 5(b) is constrained by the element a_0 including the most 1-bits in the vector. Likewise, in the BSDP Array, there is the same barrier over the entire matrix A. Moreover, due to the random matrix distribution, there is usually at least one bit-dense element in the matrix, significantly decreasing the speed of the BSDP Array.

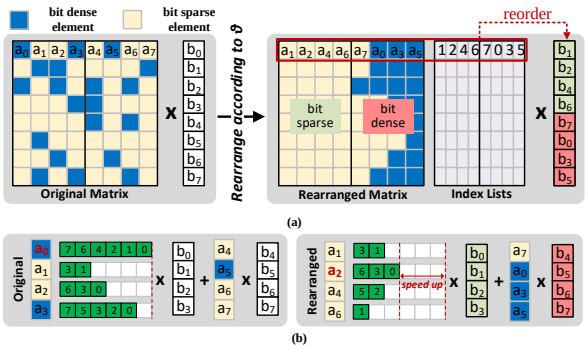


Fig. 8. Details of: (a) The Rearrangement scheme. (b) Speedup.

To settle the imbalance problem, as described in Fig. 8, we take twice the length of the BSDP Unit's vector as a unit to organize the rows of the matrix into bit-sparse and bit-dense sections, which respectively correspond to the mapping size stated in IV-B. Specifically, we sample a few elements from matrix A to roughly predict the sparsity of the matrix, which serves as a threshold θ for data rearrangement. Next, we compare the sparsity of each value against θ, thus creating a bit-sparse balanced matrix. In the given example, we rearrange the elements from a_0, a_1, a_2, a_3 into a_1, a_2, a_4, a_6. As a result, the rearrangement module reduces the cycle from 6 to 3. Moreover, to correctly realize MMA based on the rearranged A matrix, we maintain an index list, stored in Idx Buffer, as the reference to reorder the B matrix.

V. Evaluation

A. Methodology

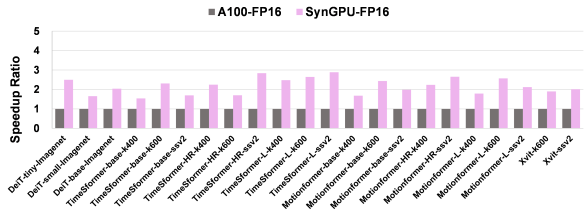
Software implementation. To verify the effectiveness of the SynGPU framework, we select DeiT [2] with ImageNet2021 [14] dataset as the representative of ViTs in image tasks and select TimeSformer [4], Motionformer [15], and Xvit [16] with Kinetics-400 (K400), Kinetics-600 (K600) [17], and SSv2 [18] datasets as the representative of ViTs in video tasks. All models are implemented using the official source code and pre-trained models without fine-tuning. Moreover, the interval used in the IBA algorithm is 80 and the batch size is set as 8 in the following experiments.

Hardware implementation. For kernel-level evaluation, we set the NVIDIA A100 GPU as the baseline. We employ Accel-Sim [19], an open-source cycle-accurate simulator, with integrating SynGPU model to simulate the computing performance in FP16 and INT8 format of A100 and SynGPU. We implement the proposed SynGPU architecture in Verilog and synthesize it by Synopsys Design Compiler to get the area and power consumption under 28nm technology. Given the absence of public data on the A100 Tensor Core area, and considering that the A100 utilizes a 7nm process while our study is based on a 28nm process, we follow work [20] to estimate A100's area and power under 28nm technology and 1.41GHz frequency. For SRAM-based on-chip buffers, we use CACTI 7 [21] to model their area and power consumption under 32nm technology and scale them to 28nm using DeepScaleTool [22].

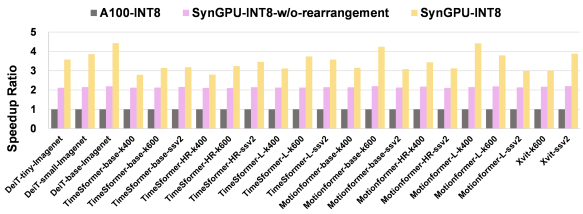
B. Experiment

Speedup. We compare the performance between A100 Tensor Core and SynGPU BSTC on various models and datasets in FP16 and INT8 formats respectively, as shown in Fig. 9(a) and Fig. 9(b). Due to the fact that the A100 Tensor Core does not skip any 0-bits, it consumes the same number of cycles when processing tensors of the same data dimension. On the other hand, SynGPU BSTC, equipped with the capability to accelerate bit-sparse MMA, achieves varying speedup compared to A100 depending on the bit sparsity of the attention map for different input data. The average speedup achieved by SynGPU-FP16 and SynGPU-INT8-w/o-rearrangement is 2.2× and 2.15× respectively, with peak speedup ratios of 2.89× and 2.20×, which do not reach the ideal speedup according to the bit-sparsity ratio indicated in Fig 2. The reason is that BSTC cannot utilize bit-sparsity with maximum efficiency without rearrangement. Therefore, based on the INT8 format, we conduct ablation experiments to demonstrate that rearrangement effectively leverages the bit sparsity acceleration capability of BSTC, providing a 3.38× speedup on average.

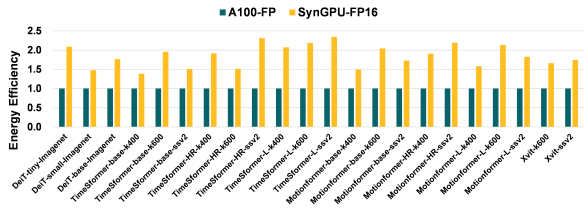
Energy Efficiency. Fig. 9(c) and Fig. 9(d) illustrate the energy efficiency results. Compared to the A100 GPU, SynGPU delivers an average of 1.84×, 1.85×, and 2.68× energy efficiency improvements in FP16, INT8-w/o-rearrangement, and INT8 situations. The optimization in FP16 and INT8 without rearrangement (INT8-w/o-rearrangement) attributed to skipping unnecessary 0-bit calculations using SynGPU,



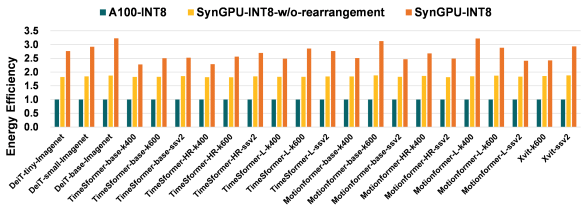
(a) Speedup in FP16



(b) Speedup in INT8



(c) Energy efficiency in FP16



(d) Energy efficiency in INT8

Fig. 9. Speedup and energy efficiency of SynGPU over A100 in FP16 and INT8 formats

thereby saving the dynamic on-chip energy. For INT8 with rearrangement, the enhancement is achieved by utilizing a rearrangement module to accelerate computation cycles, which in turn reduces the static on-chip energy.

TABLE I
AREA AND POWER OF SYNGPU ARCHITECTURE.

Parameter	Area(mm^2)	Area Ratio	Power(W)
16x16 Bit-Serial DP Units	0.9306		0.3000
1.25 KB A Buffer	0.0516		0.0358
1KB B Buffer	0.0233	**96.45%**	0.0269
2KB Acc Buffer	0.0857		0.1835
BSDP Array	**1.0913**		**0.5462**
FindExpMax	0.0006		0.0013
Exp-align	0.0003		0.0008
0.3KB Ma Buffer	0.0138	**1.91%**	0.1813
0.15KB Exp Buffer	0.0069		0.1813
Floating-point Pre-processor	**0.0216**		**0.3647**
Bit-Serial Encoder	**0.0019**	**0.16%**	**0.0430**
Comparators and selectors	0.0098		0.0225
0.15KB IdxList Buffer	0.0069	**1.48%**	0.1813
Rearrangement Module	**0.0167**		**0.2039**

Hardware Overhead and Area. Table I provides an overview of the hardware configurations and the area and power of the BSTC in each SM of SynGPU. The primary overhead of the BSTC is the BSDP Array, with an area of 1.0913mm^2. The other modules, the Floating-point Pre-processor, the Bit-Serial Encoder, and the Rearrangement Module with areas of 0.0216mm^2, 0.0019mm^2, and 0.0167mm^2, respectively, introduce about 1.91%, 0.16%, and 1.48% overhead of the entire Tensor Core, which are negligible compared to the area of the BSDP Array that occupies 96.45% of the chip area. Notably, the Rearrangement Module brings considerable speedup effects with little hardware overhead.

Compute Performance. Table II presents the compute performance of A100 Tensor Core and SynGPU BSTC in FP16 and INT8 formats. Despite the area of BSTC in SynGPU introduces a slight area overhead compared to the Tensor Core in A100 (1.13 mm^2 v.s. 0.98 mm^2), the peak computation performance of SynGPU has a significant advantage, reaching 901 TFLOPs in FP16 and 2761 TOPs in INT8, compared to 312 TFLOPs in FP16 and 624 TOPs in INT8 on A100. Thus, SynGPU achieves 2.49× and 3.81× improvements of compute density in FP16 and INT8 over A100, respectively under the same area budget. Moreover, according to the power simulation of computation modules given in Table I, we obtain the power efficiency of SynGPU for the FP16 and INT8 formats, which reaches 7.69 TFLOPs/W and 23.57 TOPs/W, respectively, delivering 2.58× and 3.95× increase over A100. Due to the acceleration and energy reduction arising from bit-level sparsity, SynGPU achieves substantial advancements in both compute density and compute power efficiency.

TABLE II
COMPUTE PERFORMANCE OF A100 AND SYNGPU.

Devices	Peak Perf.	Area Per SM	Compute Density	Power Efficiency
A100-FP	312 TFLOPs	0.98 mm^2	2.96 TFLOPs/mm^2	2.98 TFLOPs/W
A100-INT8	624 TOPs	0.98 mm^2	5.93 TOPs/mm^2	5.96 TOPs/W
SynGPU-FP	901 TFLOPs	1.13 mm^2	7.37 TFLOPs/mm^2	7.69 TFLOPs/W
SynGPU-INT8	2761 TOPs	1.13 mm^2	22.60 TOPs/mm^2	23.57 TOPs/W

VI. CONCLUSION

In this paper, we introduce SynGPU, an innovative algorithm-hardware co-design framework that synergizes CUDA Cores and Tensor Cores on GPUs to exploit and harness the bit-level sparsity among similar tokens, achieving significant performance enhancement of ViTs. SynGPU tackles two crucial challenges: firstly, the low parallelism caused by competitive access to the RF by CUDA Cores and Tensor Cores; secondly, the inability of tensor cores to accelerate bit-level sparse MMA due to differing exponents in floating-point vectors. Our innovative scheme fully exploits the bit sparsity of ViTs for acceleration, yielding remarkable outcomes without incurring accuracy loss.

References

[1] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[2] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[3] D. Wodajo, "Deepfake video detection using convolutional vision transformer," *arXiv: Computer Vision and Pattern Recognition,arXiv: Computer Vision and Pattern Recognition*, Nov 2020.

[4] G. Bertasius, H. Wang, and L. Torresani, "Is space-time attention all you need for video understanding?" in *ICML*, vol. 2, no. 3, 2021, p. 4.

[5] H. You, Z. Sun, H. Shi, Z. Yu, Y. Zhao, Y. Zhang, C. Li, B. Li, and Y. Lin, "Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 273–286.

[6] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, "Dynamicvit: Efficient vision transformers with dynamic token sparsification," *Advances in neural information processing systems*, vol. 34, pp. 13 937–13 949, 2021.

[7] P. Dong, M. Sun, A. Lu, Y. Xie, K. Liu, Z. Kong, X. Meng, Z. Li, X. Lin, Z. Fang *et al.*, "Heatvit: Hardware-efficient adaptive token pruning for vision transformers," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 442–455.

[8] Z. Li, M. Sun, A. Lu, H. Ma, G. Yuan, Y. Xie, H. Tang, Y. Li, M. Leeser, Z. Wang *et al.*, "Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 109–116.

[9] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[10] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[11] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture*, 2017, pp. 382–394.

[12] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 749–763.

[13] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 963–976.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] M. Patrick, D. Campbell, Y. Asano, I. Misra, F. Metze, C. Feichtenhofer, A. Vedaldi, and J. F. Henriques, "Keeping your eye on the ball: Trajectory attention in video transformers," *Advances in neural information processing systems*, vol. 34, pp. 12 493–12 506, 2021.

[16] A. Bulat, J. M. Perez Rua, S. Sudhakaran, B. Martinez, and G. Tzimiropoulos, "Space-time mixing attention for video transformer," *Advances in neural information processing systems*, vol. 34, pp. 19 594–19 607, 2021.

[17] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev *et al.*, "The kinetics human action video dataset," *arXiv preprint arXiv:1705.06950*, 2017.

[18] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag *et al.*, "The" something something" video database for learning and evaluating visual common sense," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5842–5850.

[19] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486.

[20] Z. Mo, L. Wang, J. Wei, Z. Zeng, S. Cao, L. Ma, N. Jing, T. Cao, J. Xue, F. Yang *et al.*, "Lut tensor core: Lookup table enables efficient low-bit llm inference acceleration," *arXiv preprint arXiv:2408.06003*, 2024.

[21] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[22] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2021, p. 1–5. [Online]. Available: http://dx.doi.org/10.1109/iscas51556.2021.9401196

Maximizing Energy Efficiency in Spiking Neural Networks: A Dynamic Joint Pruning Framework

Shuo Chen*†‡, Zeshi Liu*, Haihang You*‡
*Institute of Computing Technology, Chinese Academy of Sciences, Bejing, China
†School of Computer Science and Technology, University of Chinese Academy of Sciences, Bejing, China
‡Zhongguancun Laboratory, Bejing, China
{chenshuo20s, liuzeshi, youhaihang}@ict.ac.cn

***Abstract*—Spiking Neural Networks (SNNs) face increasing challenges for efficient deployment as architectures grow in complexity, necessitating network pruning to improve energy and computational efficiency. Existing pruning methods primarily focus on a single form of sparsity, overlooking the importance of joint pruning, which is critical for minimizing synaptic operations (SOPs) and enhancing energy efficiency. This paper presents a novel dynamic joint pruning framework that leverages both spatiotemporal spike sparsity and weight sparsity to minimize SOPs in SNN inference. Based on a comprehensive analysis of the SOPs model, we introduce an integrated solution that combines a multi-stage masking mechanism for fine-grained neuron firing threshold control, a temporal attention batch normalization (TABN) module with learnable time scaling factors, and a dynamic sparse strategy that adjusts importance coefficients based on real-time computational impact. Experimental results on CIFAR-10, CIFAR-100, and ImageNet validate the effectiveness of proposed framework. Our method achieves up to 126.38× compression ratio of SOPs on CIFAR-10 with minimal accuracy loss, establishing a new state-of-the-art in energy-efficient SNN pruning.**

***Index Terms*—spiking neural network, network pruning, low power computing**

I. Introduction

Spiking Neural Networks (SNNs), regarded as the third generation of neural networks [1], have attracted significant attention for their potential in energy-efficient machine intelligence [2], [3]. By leveraging sparse and event-driven spike signals, SNNs enable low-power and multiplication-free inference on neuromorphic hardware compared to Artificial Neural Networks (ANNs) [4]–[6]. However, to tackle increasingly complex tasks, modern SNN architectures have grown in depth and scale [7], [8], which requires greater computational resources and compromises their low power advantages.

Network pruning has emerged as a promising approach to eliminate redundancy and reduce model size in SNNs. Common techniques mainly focus on pruning synaptic connections, including magnitude-based pruning [9], [10], optimization-driven methods [11], [12], and bio-inspired algorithms [13], [14]. Although these methods achieve high weight sparsity and reduce computational resource requirements, they are less effective at directly reducing the number of synaptic operations (SOPs). SOPs reflect both the computational cost and energy consumption of SNN inference on neuromorphic hardware. Therefore, many works [4]–[6], [11] use SOP-related metrics (e.g., energy consumption per SOP) as indicators for assessing hardware energy efficiency. This indicates that minimizing SOPs is crucial for achieving low-power SNN computing.

To minimize SOPs in SNN inference, we develop a fine-grained SOPs model for SNNs and explore comprehensive pruning strategies. According to the SOPs model, weight sparsity and spatiotemporal spike sparsity are two fundamental factors determining SOP reduction. Additionally, the evolution of importance coefficients during pruning serves as another critical factor in the sparse training process. Previous works have achieved spike number reduction by different approaches [15]–[18]. In particular, sparse regularization as a pruning method directly eliminates model redundancy and achieves high spike sparsity [19]–[21].

However, studies that combine spike and weight sparsity to maximize SOP reduction remain limited. UPSNN [22] and ADMM [23] have preliminarily explored joint pruning with sparse regularization. The former introduces neuron sparsity but not the finer-grained spike sparsity, which directly determines SOP reduction. The latter simply performs weight pruning and activity regularization simultaneously without optimization, making it ineffective in achieving high sparsity during joint pruning. Consequently, neither method achieves the minimization of SOPs. Moreover, even after addressing these issues, joint pruning with sparse regularization still faces challenges: (1) spikes and weights being interdependent during pruning, as spikes are model outputs influenced by weights; (2) the need to balance the sparse strengths of different regularization terms during sparse training.

In this paper, we propose a dynamic joint pruning framework[1] that fully exploits available redundancy to minimize SOPs in SNN inference. Novel techniques are introduced to address the challenges in joint pruning. First, we use a multi-stage masking mechanism to decouple spike sparsity from weights. These masks partition neuron membrane potentials into multiple stages, enabling neurons to adopt different firing thresholds during inference and achieve spatial spike sparsity. Second, to address temporal redundancy, we introduce temporal attention batch normalization (TABN). TABN incorporates a trainable time scaling factor into standard batch normalization, thereby pruning redundant spike activity across time steps. Finally, we employ a dynamic regularization strategy for joint sparse training. Coefficients that reflect the importance of pruning candidates are dynamically updated based on the SOPs model to adjust sparse strengths. Experimental results demonstrate that the proposed framework significantly reduces SOPs, lowers energy consumption, and maintains high model accuracy.

In summary, our key contributions are as follows:

- We develop an SOPs model that incorporates the temporal dimension. This model identifies spatiotemporal spike sparsity, weight sparsity, and the evolution of importance coefficients as key factors influencing SOP reduction.
- We propose novel methods to achieve spatiotemporal spike sparsity: a multi-stage masking mechanism for spatial spike sparsity, which allows neurons to adopt variable firing thresholds, and temporal attention batch normalization (TABN), which prunes redundant spike activity across time steps to realize temporal spike sparsity.

[1]Our code: https://github.com/MmPple/SDJP

- We develop a dynamic joint pruning framework that comprehensively exploits the available redundancy for maximal SOP reduction. An dynamic strategy is introduced to adjust sparse regularization coefficients based on real-time computational impact.
- Experiments on CIFAR-10, CIFAR-100, and ImageNet demonstrate that our proposed framework significantly reduces SOPs and energy consumption while maintaining high model accuracy. On CIFAR-10, our method achieves an SOP compression ratio of up to 126.38× with only a 2.15% accuracy drop, significantly outperforming state-of-the-art methods.

II. Related Work and Background

A. Pruning for SNNs

Pruning induces sparsity in neural networks by removing certain elements, enhancing computational efficiency and reducing power consumption. In SNNs, the main forms of sparsity explored are weight sparsity and spike sparsity.

Weight Sparsity. Weight pruning in SNNs is widely studied because it enables low-power operation on neuromorphic hardware like Loihi [5], which supports unstructured sparsity. Inspired by pruning techniques in artificial neural networks (ANNs), magnitude-based pruning removes small weights during learning processes such as event-driven contrastive divergence [24], spike-timing-dependent plasticity [2], and ANN-to-SNN conversion [25]. Kim et al. [9] applied the lottery ticket hypothesis to SNN pruning using iterative magnitude pruning. ADMM combined spatio-temporal backpropagation with the alternating direction method of multipliers (ADMMs) [23]. Some incorporate biologically inspired strategies like synapse regeneration, such as Grad R [14] and ESLSNN [11].

Spike Sparsity. Sparse signals are inherently advantageous in SNNs, as sparse spiking patterns contribute to low-power computations in neuromorphic systems utilizing spike-triggered mechanisms. Pruning methods to achieve spike sparsity include regularization techniques and attention mechanisms. Regularization methods like L_p regularization [19] and low-activity regularization [20], [21], [23] promote high spike sparsity by eliminating redundancy without compromising performance. Attention mechanisms focus computational resources on critical information, reducing the number of spikes required for effective processing [18], [26], [27].

Hybrid Pruning. Some hybrid pruning methods have begun to combine sparsity at multiple levels. ADMM [23] performed weight pruning based on the ADMMs and activity regularization simultaneously. However, they simply combine the two without optimization, making it ineffective for joint pruning. UPSNN [22] combines unstructured weight and neuron pruning to maximize sparsity and enhance energy efficiency. However, UPSNN introduces neuron sparsity rather than finer-grained spike sparsity, and does not fully exploit redundancy to maximize energy efficiency.

B. Spiking Neural Network

Spiking neural networks apply the spiking neuron as the fundamental unit to collect signals from other neurons or inputs during a time step, and modulate the membrane potential by these signals. An output spike is generated and delivered to post-synaptic neurons when the membrane potential exceeds the threshold. The commonly used neuron model is Leaky Integrate and Fire (LIF) model [28]. It is defined as:

$$h[t] = u[t-1] + \frac{1}{\tau}(\sum_i w_i s_i[t] - u[t-1]), \tag{1}$$

$$u[t] = h[t] \cdot (1 - s[t]) + V_{reset} \cdot s[t], \tag{2}$$

where Eq.1 and Eq.2 describe the charging and resetting processes of a LIF neuron. t represents the current time step. s_i represents the spike train from the ith pre-synaptic neuron and w_i is the corresponding synaptic weight. $u[t]$ denotes the membrane potential at the end of the current time step t. V_{reset} represents the reset voltage, which we set to 0. $s[t] \in \{0, 1\}$ represents the output of the neuron in the current time step. The fire process is described as:

$$s[t] = \Theta(h[t] - V_{threshold}), \tag{3a}$$

$$\Theta(x) = 0, x < 0 \; otherwise \; 1, \tag{3b}$$

where $V_{threshold}$ is the threshold voltage.

For the learning process, we use the spatio-temporal backpropagation (STBP) [29] to train SNNs. To overcome the problem that Eq.3b is not differentiable at zero, we follow pervious works and use the arc tangent function as a surrogate function to replace Eq.3b in the backward phase [29].

III. Methodology

In this section, we present a framework for dynamic joint pruning in deep SNNs. Figure 1 shows an overview of the proposed framework. This framework performs sparse regularization with joint optimization to leverage spatiotemporal spike sparsity and weight sparsity. It integrates a sparse strategy that dynamically adjusts importance coefficients during sparse training. Our objective is to minimize SOPs in SNN inference while maintaining high model accuracy.

A. SOPs Model

The energy consumption in SNNs primarily arises from synaptic operations (SOPs). Specifically, the energy consumption for SNNs is given by:

$$E = C_E \cdot \#\text{SOPs} = C_E \cdot \sum_i s_i \cdot c_i, \tag{4}$$

where E represents the total energy consumption, C_E is the average energy required for each synaptic operation, s_i denotes the number of spikes fired by neuron i, and c_i is the number of synaptic connections from neuron i.

From Eq.4, reducing energy consumption depends on minimizing SOPs, which are determined by the number of spikes fired by each neuron (spike sparsity) and the number of synaptic connections (weight sparsity). To better describe spike sparsity, we incorporate the temporal dynamics of SNNs into the SOPs model:

$$\#\text{SOPs} = \sum_t \sum_i s_i[t] \cdot c_i, \tag{5}$$

where $s_i[t]$ denotes the number of spikes fired by neuron i at time step t. By considering time steps, we can further decompose spike sparsity across both time and space.

Based on Eq. 5, we define spatiotemporal spike sparsity, which describes the frequency and number of spikes fired by neurons across time and space. This provides a refined view of neuron activity, indicating the precise sparse pattern of spikes rather than simply whether a neuron is active. Furthermore, the SOPs model indicates that achieving high spatiotemporal spike sparsity and weight sparsity is critical for minimizing SOPs in SNN inference. This need to develop efficient joint pruning methods.

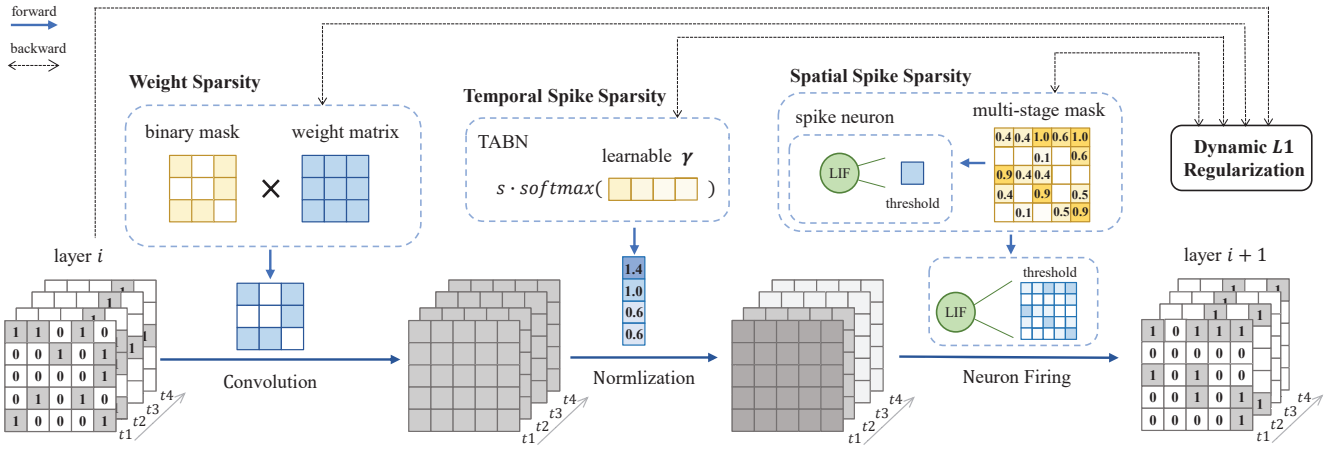


Fig. 1: The overview of proposed framework.

B. Spatial Spike Sparsity

To achieve high spike sparsity, L_1 regularization has proven effective [19]–[21]. However, this approach shows limited effectiveness when combined with weight pruning. Since L_1 regularization indirectly achieve spike sparsification through weight modification. This leads to a coupling between spike sparsity and weight sparsity.

To address this coupling issue, an intuitive solution is to independently reduce spikes through trainable gating parameters, such as neuron thresholds. However, the unbounded nature of threshold values, combined with the regularization constraint, poses significant challenges for training convergence. Thresholds may drift arbitrarily, destabilizing the optimization process. Moreover, completely pruning neurons would require manual threshold adjustments for each neuron, which is impractical and lacks scalability in large-scale networks.

We propose a novel multi-stage masking mechanism that equivalently substitutes variable thresholds in sparse training. Our approach applies a mask m_i^n to the neuron membrane potential $h_i[t]$:

$$h_i'[t] = m_i^n \cdot h_i[t] \tag{6}$$

where mask m_i^n is constrained to values from a set of K values uniformly distributed between 0 and 1. The multi-stage mask specifically addresses a key limitation of continuous masks, which typically converge toward a bimodal distribution of 0 and 1 during training, making it difficult to differentiate firing of neurons. This approach modulate membrane potentials through multiple stages, enabling fine-grained control over spike firing without complete neuron deactivation, thereby enhancing spatial spike sparsity.

To overcome the non-differentiability of step functions, we approximate the multi-stage mask by accumulating multiple sigmoid functions in practice:

$$m_i^n = \sum_{k=0}^{K-1} \frac{1}{K} \cdot \sigma\left(\beta_n \left(\theta_i - \frac{k}{K}\right)\right) \tag{7}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ denotes the sigmoid function, β_n controls the function steepness, and θ_i represents the learnable parameter.

During the training process, θ_i are initialized to 0, and β_n is gradually increased, resulting in steeper sigmoid functions and increased difficulty for neurons to transition between stages. This gradual constraint allows neurons to naturally settle into different stages with corresponding firing thresholds.

During training, θ_i are initialized to 0 and trained with L_1 regularization, and β_n is gradually increased. This results in steeper sigmoid functions and makes it increasingly difficult for neurons to transition between stages. This gradual constraint allows neurons to naturally settle into different stages with corresponding firing thresholds.

C. Temporal Spike Sparsity

Recent studies have revealed significant redundancy in outputs across different time steps in SNN models—a phenomenon known as temporal information concentration [30] or spatio-temporal invariance [26]. This presents an opportunity for temporal spike sparsification. While existing temporal compression methods focus on directly reducing the number of time steps to decrease latency and computation, they are often ineffective for modern STBP-based SNNs that already operate with few time steps. Further reduction can lead to significant performance degradation.

To adaptively achieve temporal spike sparsity, we propose a novel data-independent temporal attention batch normalization (TABN) that incurs no additional computational cost during inference. TABN replaces the linear transformation in standard batch normalization layers with a temporal attention mechanism, enabling differentiated spike firing across time steps. The computation of TABN is formulated as:

$$v_i'[t] = s \cdot \left(\frac{v_i[t] - \mu}{\sqrt{\sigma^2 + \epsilon}}\right) \cdot \alpha_t + \beta, \tag{8}$$

where $v[t]$ represents the accumulated input of the i-th neuron at time step t, μ and σ denote the batch mean and standard deviation, and ϵ ensures numerical stability. The time step scaling factor α_t is normalized through a softmax function:

$$\alpha_t = \frac{e^{\gamma_t}}{\sum_{k=1}^{T} e^{\gamma_k}}, \tag{9}$$

where T represents the total number of time steps, and γ_t is a learnable parameter. The global learnable scaling factor s prevents spike and gradient vanishing during training, while β serves as a learnable bias term.

TABN learns temporal importance without introducing additional computational overhead. Specifically, we implement temporal spike sparsity at channel level, aligning with the observation that temporal output similarity manifests at the channel granularity [30]. Spike outputs of a channel represent a complete unit of feature expression and transmission. The channel-wise application of TABN enables

precise optimization of temporal redundancy while preserving feature integrity.

D. Dynamic Joint Pruning Framework

Combining proposed spatiotemporal spike sparsification mechanisms with L1 regularization, we develop a joint pruning framework comprising sparse training and fine-tuning phases.

In the sparse training phase, we jointly optimizes weight sparsity and spatiotemporal spike sparsity through the binary weight mask, multi-stage neuron mask, and TABN. We control the sparsity level using L1 regularization:

$$L_{\text{sparse}} = \lambda \left(\sum_i \|\boldsymbol{m}_i^w\|_1 + \sum_j \|m_j^n\|_1 + \sum_c \|\boldsymbol{\gamma}_c\|_1 \right) \tag{10}$$

where $\boldsymbol{m}_i^w$, m_j^n, and $\boldsymbol{\gamma}_c$ denote the weight mask vector for presynaptic neuron i, the multi-stage mask for neuron j, and the temporal weight vector for channel c, respectively. λ is a hyperparameter controlling the sparsity level. In practice, we use the approximation in Eq. 7 with $K = 1$ to overcome the non-differentiability of binary weight mask $\boldsymbol{m}_i^w$.

In the fine-tuning phase, we fix the sparse masks and convert multi-stage masks to neuron thresholds. The weights and neurons corresponding to zero masks with $\beta_n < 0$ are completely pruned. The learned temporal importance distribution of TABN is preserved. We retrain the pruned model without the sparsification loss L_{sparse} to recover performance loss induced by pruning during the sparse training phase.

To improve the joint pruning and minimize SOPs reduction, we introduce an strategy that dynamically adjusts sparse regularization terms based on real-time computational impact. According to the SOPs model Eq. 5, each sparsity form's contribution to computational efficiency is weighted by respective coefficients, leading to the dynamic sparse loss function:

$$L_{\text{sparse}} = \lambda \left(\sum_i s_i \|\boldsymbol{m}_i^w\|_1 + \sum_j w_j \|\boldsymbol{m}_j^n\|_1 + \sum_c w_c \|\boldsymbol{\gamma}_c\|_1 \right). \tag{11}$$

Here, the coefficients represent each sparsity form's computational impact:s_i measures spike count of presynaptic neuron i for weight sparsity, w_j reflects the preserved next-layer connection count of neuron j for spatial spike sparsity, and w_c indicates the remaining channel-level weights for temporal spike sparsity. These coefficients are dynamically updated during sparse training to ensure pruning decisions align with their actual computational efficiency impact.

For convolutional layers, we estimate coefficients at the channel level to simplify computation. Specifically, $\boldsymbol{m}_i^w$ represents weight masks associated with input channel i, encompassing all output channels and kernel spatial dimensions (height and width). The scaling factor s_i measures the spike count of input channel i. Similarly, the mask $\boldsymbol{m}_j^n$ corresponds to the multi-stage neuron mask vector for input channel j, and w_j reflects the number of preserved convolutional kernel weights associated with input channel j.

TABLE I: Hyperparameter settings for proposed method.

Dataset	Time Steps	Optimizer	Learning Rate	Batch Size
CIFAR10	8	Adam	0.0001	16
CIFAR100	4	SGD	0.025	64
ImageNet	4	Adam	0.001	256

IV. Experimental Results

A. Experimental Setup

In this section, we evaluate the proposed method for the image classification task on the CIFAR-10 [32], CIFAR-100 [32], and ImageNet [33] datasets. Following previous works, we employ spike frequency encoding and take the first convolutional layer of the model as a learnable encoder to transform static image inputs into spiking patterns. Our implementation is based on PyTorch and the SpikingJelly package. All experiments are executed on an RTX 3090 GPU.

We follow the hyperparameter setting from UPSNN [22]. For CIFAR-10, sparse training is performed over 800 epochs, followed by 200 fine-tuning epochs, while CIFAR-100 and ImageNet use 280 sparse training epochs and 40 fine-tuning epochs each. During the sparse training, β_n is linearly increased from 5 to 1000, with stage count K is fixed to 20. In the experiment of performance comparison, we use $\lambda = \{5, 10, 40\} \times 10^{-12}$ for CIFAR-10, $\{1, 2, 3\} \times 10^{-9}$ for CIFAR-100, and $\{1, 6, 10\} \times 10^{-11}$ for ImageNet. The details of other hyperparameters is shown in Table I.

B. Comparison with the State-of-the-Art Works

1) Performance Comparison: We compare our method with state-of-the-art SNN pruning algorithms including ADMM [23], Grad R [14], ESLSNN [11], STDS [31], and UPSNN [22], as summarized in Table II. In the table, "Avg. SOPs" indicates average SOPs per instance in inference. "Ratio" is the compression ratio, indicating the energy efficiency. "Sparsity Type" specifies the achieved sparsity: "W" for weight sparsity, "W+N" for weight and neuron sparsity, and "W+S" for weight and spike sparsity.

On CIFAR-10, our method achieves the highest accuracy of 92.65% while reducing energy consumption to 21.55M SOPs and achieving a compression ratio of 36.24. This outperforms UPSNN, which requires 38.32M SOPs with a ratio of 20.25 for similar accuracy. At higher sparsity levels, our method maintains superior accuracy and further reduces energy consumption, demonstrating up to 126.38× improvement in energy efficiency. For CIFAR-100, our method attains an accuracy of 72.51%, using only 17.11M SOPs, surpassing both STDS and UPSNN in accuracy and energy savings. On the more challenging ImageNet dataset, our method achieves 61.96% accuracy, using only 227.45M SOPs, outperforming previous methods that either have lower accuracy or require more SOPs. These results demonstrate that by jointly pruning weights and spikes, our method effectively balances high accuracy with significant energy reduction across various datasets.

2) Sparsity Comparison: Table III presents the effect of joint pruning in achieving spike sparsity and weight sparsity at different accuracy levels. We compare proposed method with UPSNN [22] and ADMM [23] on SEW ResNet18 for CIFAR-100. In Table III, "#Spike" and "Avg. FR" denote the average number of spikes and the average spike firing rate per neuron across instances in test set. "Param." indicates the total number of remaining weights after pruning. Our method achieves the best performance among the compared methods. Compared to UPSNN, our method significantly reduces both the total number of spikes and the average firing rate. This reduction is attributed to our fine-grained control over neuron spike firing. These results demonstrate the efficiency of our method in joint pruning for spike and weight sparsity.

C. Ablation Study Results

1) Performance with different SOPs: We evaluate the robustness of proposed method across various average SOPs. As depicted in Fig-

TABLE II: Performance comparison between proposed method and the state-of-the-art works. We mark the best results in bold.

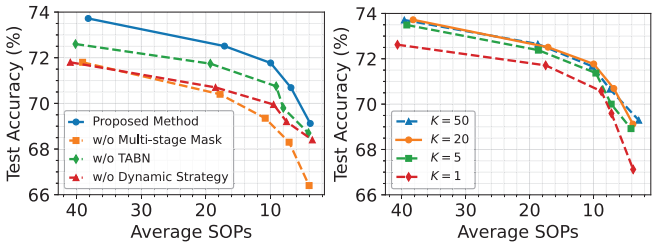
Dataset	Method	Arch.	Sparsity Type	Test Acc. (%)	Acc. Loss (%)	Avg. SOPs (M)	Ratio
CIFAR10	Grad R [14]	6Conv, 2FC	W	92.54	-0.30	371.05	2.09
				92.03	-0.81	143.69	5.40
				89.32	-3.52	41.89	18.53
	ESLSNN [11]	ResNet19	W	92.08	-0.63	298.24	2.11
				91.46	-1.25	178.10	3.54
				90.90	-1.81	108.89	5.79
	ADMM [23]	7Conv, 2FC	W+S	90.19	-0.13	107.97	2.91
				88.18	-2.14	49.72	6.32
				79.63	-10.69	28.65	10.96
	UPSNN [22]	6Conv, 2FC	W+N	92.63	-0.21	38.32	20.25
				92.05	-0.79	16.47	47.12
				90.65	-2.19	8.50	91.31
	This Work	6Conv, 2FC	W+S	**92.65**	**-0.21**	**21.55**	**36.24**
				92.18	**-0.68**	**9.61**	**81.27**
				90.71	**-2.15**	**6.18**	**126.38**
CIFAR100	STDS [31]	SEW ResNet18	W	72.69	-1.36	48.65	5.34
				71.05	-3.00	19.11	13.61
				70.13	-3.92	13.90	18.71
	UPSNN [22]	SEW ResNet18	W+N	72.34	-1.82	27.16	9.24
				71.43	-2.73	21.16	11.86
				70.45	-3.71	9.60	26.14
	This Work	SEW ResNet18	W+S	**72.51**	**-1.54**	**17.11**	**15.20**
				71.77	**-2.28**	**9.96**	**26.10**
				70.69	**-3.36**	**6.81**	**38.18**
ImageNet	STDS [31]	SEW ResNet18	W	61.51	-1.67	512.95	2.61
				59.93	-3.25	273.39	4.89
				58.06	-5.12	182.49	7.33
	UPSNN [22]	SEW ResNet18	W+N	61.89	-1.29	311.70	4.29
				60.00	-3.18	177.99	7.52
				58.99	-4.19	116.88	11.45
	This Work	SEW ResNet18	W+S	**61.96**	**-1.14**	**227.45**	**5.88**
				60.31	**-2.79**	**109.18**	**12.24**
				59.07	**-4.03**	**57.31**	**23.33**

TABLE III: Sparsity comparison between proposed method and the state-of-the-art works. We mark best results in bold.

Method	Test Acc.(%)	#Spike	Avg. FR	Param. (M)
ADMM [23]	72.50	250647	0.1141	5.75
	71.65	215339	0.0986	3.89
UPSNN [22]	72.47	261386	0.1630	3.93
	71.59	165418	0.1419	3.11
This Work	**72.51**	**79321**	**0.0470**	**3.19**
	71.77	**56491**	**0.0439**	**2.72**

ure 2a, our method consistently maintains high test accuracy on the CIFAR-100 dataset even as the average SOPs decrease significantly. When reducing the average SOPs from 38.23M to 3.82M, which is a reduction of over 10 times, the test accuracy only declines from 73.72% to 69.12%. These results highlight that our method remains effective across a wide range of SOP levels, especially at lower SOPs corresponding to lower energy costs.

2) Effect of Different Mechanisms: To understand the impact of each mechanism in our method, we perform an ablation study by individually removing the multi-stage mask, temporal attention batch normalization (TABN), and the dynamic coefficient strategy. As shown in Figure 2a, the absence of the multi-stage mask results in a sharper accuracy loss as SOPs decrease, indicating that spatial spike sparsity plays a crucial role in reducing SOPs. Removing TABN and the dynamic coefficient strategy also affects accuracy, especially at higher SOPs levels. This is because, at this point, there is more redundant spike activity across time steps and greater variance in the importance among different sparsity terms, offering more opportunity for effective sparsity.



(a) (b)

Fig. 2: (a) Results of pruned models with different mechanisms for CIFAR-100. (b) Results of pruned models fro CIFAR-100 with different stage count K in the multi-stage mask.

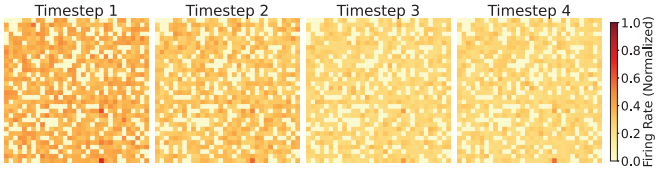


Fig. 3: Visualization of spatiotemporal spike sparsity in feature maps of the penultimate layer of pruned model on CIFAR-100.

3) Effect of Stage Count K: We evaluate the impact of different stage count settings K in the multi-stage mask on the performance of our method. As shown in Figure 2b, when $K = 1$, the multi-stage mask reduces to a binary mask, leading to suboptimal performance. As K increases, the performance improves, demonstrating the benefit of using a multi-stage mask with finer granularity. In particular, our method exhibits stable performance across a wide range of K values (from 5 to 50), indicating that it is robust to the choice of stage count.

4) Visualization of Spatiotemporal Spike Sparsity: To further illustrate the effect of proposed method, we visualize the spatiotemporal spike sparsity in the feature map outputs of the penultimate layer of pruned model on CIFAR-100, As shown in Figure 3, the color intensity indicates the average neuron firing rate. The visualization reveals both spatial and temporal spike sparsity achieved by our method. Spatially, at a single time step, neurons exhibit significant variations in firing rates, demonstrating that the multi-stage mask effectively promotes sparsity by selectively suppressing less important neurons. Temporally, we observe that spikes are mainly concentrated in the first two time steps. This is attributed to the TABN, which reduces redundant spike activity over time.

V. Conclusion

In this paper, we propose a dynamic joint pruning framework for SNNs that fully exploits weight sparsity and spatiotemporal spike sparsity to maximizing SOPs reduction and energy efficiency in inference. Our multi-stage masking mechanism and temporal attention batch normalization enable fine-grained control over spatiotemporal neuron activity without compromising model accuracy. Experimental results demonstrate that our framework significantly reduces SOPs and energy consumption, achieving superior performance compared to state-of-the-art methods. This work highlights the importance of jointly optimizing multiple forms of sparsity and provides a foundation for future research in low-power SNN computing.

Acknowledgments

This work was supported by the Postdoctoral Fellowship Program (Grade B) of China Postdoctoral Science Foundation under grant number GZB20240760.

REFERENCES

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] N. Rathi, P. Panda, and K. Roy, "Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 668–677, 2018.

[3] A. Basu, L. Deng, C. Frenkel, and X. Zhang, "Spiking neural network integrated circuits: A review of trends and future directions," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2022, pp. 1–8.

[4] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[5] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[6] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He, F. Chen, N. Deng, S. Wu, Y. Wang, Y. Wu, Z. Yang, C. Ma, G. Li, W. Han, H. Li, H. Wu, R. Zhao, Y. Xie, and L. Shi, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.

[7] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going deeper with directly-trained larger spiking neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 11 062–11 070.

[8] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian, "Deep residual learning in spiking neural networks," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 21 056–21 069.

[9] Y. Kim, Y. Li, H. Park, Y. Venkatesha, R. Yin, and P. Panda, "Exploring lottery ticket hypothesis in spiking neural networks," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XII*. Springer, 2022, pp. 102–120.

[10] S. Huang, H. Fang, K. Mahmood, B. Lei, N. Xu, B. Lei, Y. Sun, D. Xu, W. Wen, and C. Ding, "Neurogenesis dynamics-inspired spiking neural network training acceleration," in *60th ACM/IEEE Design Automation Conference, DAC*, 2023, pp. 1–6.

[11] J. Shen, Q. Xu, J. Liu, Y. Wang, G. Pan, and H. Tang, "Esl-snns: An evolutionary structure learning strategy for spiking neural networks," in *AAAI Conference on Artificial Intelligence*, 2023.

[12] J. Chen, H. Yuan, J. Tan, B. Chen, C. Song, and D. Zhang, "Resource constrained model compression via minimax optimization for spiking neural networks," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, p. 5204–5213.

[13] G. Bellec, D. Salaj, A. Subramoney, R. A. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," *Advances in neural information processing systems*, vol. 31, 2018.

[14] Y. Chen, Z. Yu, W. Fang, T. Huang, and Y. Tian, "Pruning of deep spiking neural networks through gradient rewiring," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2021, pp. 1713–1721, main Track.

[15] D. Wu, Y. Qi, K. Cai, G. Jin, X. Yi, and X. Huang, "Direct training needs regularisation: Anytime optimal inference spiking neural network," 2024. [Online]. Available: https://arxiv.org/abs/2405.00699

[16] Q. Meng, M. Xiao, S. Yan, Y. Wang, Z. Lin, and Z.-Q. Luo, "Training high-performance low-latency spiking neural networks by differentiation on spike representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 12 444–12 453.

[17] N. Rathi and K. Roy, "Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 6, pp. 3174–3182, 2023.

[18] M. Yao, G. Zhao, H. Zhang, Y. Hu, L. Deng, Y. Tian, B. Xu, and G. Li, "Attention spiking neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 8, pp. 9393–9410, 2023.

[19] S. Narduzzi, S. A. Bigdeli, S.-C. Liu, and L. A. Dunbar, "Optimizing the consumption of spiking neural networks with activity regularization," in *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 61–65.

[20] D. Neil, M. Pfeiffer, and S.-C. Liu, "Learning to be efficient: algorithms for training low-latency, low-compute deep spiking neural networks." New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2851613.2851724

[21] T. Pellegrini, R. Zimmer, and T. Masquelier, "Low-activity supervised convolutional spiking neural networks applied to speech commands recognition," in *2021 IEEE Spoken Language Technology Workshop (SLT)*, 2021, pp. 97–103.

[22] X. Shi, J. Ding, Z. Hao, and Z. Yu, "Towards energy efficient spiking neural networks: An unstructured pruning framework," in *The Twelfth International Conference on Learning Representations*, 2024.

[23] L. Deng, Y. Wu, Y. Hu, L. Liang, G. Li, X. Hu, Y. Ding, P. Li, and Y. Xie, "Comprehensive snn compression using admm optimization and activity regularization," *IEEE transactions on neural networks and learning systems*, 2021.

[24] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, "Stochastic synapses enable efficient brain-inspired learning machines," *Frontiers in neuroscience*, vol. 10, p. 241, 2016.

[25] Y. Liu, K. Qian, S. Hu, K. An, S. Xu, X. Zhan, J. J. Wang, R. Guo, Y. Wu, T.-P. Chen, Q. Yu, and Y. Liu, "Application of deep compression technique in spiking neural network chip," *IEEE transactions on biomedical circuits and systems*, vol. 14, no. 2, pp. 274–282, 2019.

[26] M. Yao, J. Hu, G. Zhao, Y. Wang, Z. Zhang, B. Xu, and G. Li, "Inherent redundancy in spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 16 924–16 934.

[27] S. Kundu, G. Datta, M. Pedram, and P. A. Beerel, "Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 3953–3962.

[28] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[29] Y. Wu and et.al., "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.

[30] Y. Kim, Y. Li, H. Park, Y. Venkatesha, A. Hambitzer, and P. Panda, "Exploring temporal information dynamics in spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 7, 2023, pp. 8308–8316.

[31] Y. Chen, Z. Yu, W. Fang, Z. Ma, T. Huang, and Y. Tian, "State transition of dendritic spines improves learning of sparse spiking neural networks," in *International Conference on Machine Learning*, 2022.

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

SAGA: A Memory-Efficient Accelerator for GANN Construction via Harnessing Vertex Similarity

Ruiyang Chen, Xueyuan Liu, Chunyu Qi, Yuanzheng Yao, Yanan Sun, Xiaoyao Liang, and Zhuoran Song*
Shanghai Jiao Tong University, Shanghai, China
{chenruiyang, songzhuoran}@sjtu.edu.cn

***Abstract*—Graph-traversal-based Approximate Nearest Neighbor (GANN) search and construction have become key retrieval techniques in various domains, such as recommendation systems and social networks. However, deploying GANN in real-world scenarios faces significant challenges, as high-dimensional vertices within the graph can lead to intensive memory demands. Although architectures like NDSearch have been proposed to accelerate GANN search, they are hard to deploy for GANN construction, as their pre-processing methods introduce massive overhead in dynamic graphs. In this paper, given the observation that neighboring vertices in a dynamic graph exhibit feature similarity, we propose SAGA, the first accelerator that alleviates memory bound in GANN construction. To capture this similarity, we directly leverage the first step of construction to gather vertices with the same starting point into a cluster to minimize the similarity detection overhead. Next, we decompose vertices into key and non-key ones, where their deltas fall in a narrow range, which is suitable to be quantized to lower bit widths. Building upon this approach, we design a specialized architecture, which efficiently implements the GANN construction by two-level scheduling and a mixed-precision supported bit-serial unit. Through comprehensive evaluation, we demonstrate that SAGA can achieve an average speedup of 9.30×, 4.87×, 4.15× and 35.46×, 7.60×, 5.15× energy savings over CPU, GPU and NDSearch, respectively, while retaining task accuracy.**

I. Introduction

Graph-traversal-based Approximate Nearest Neighbor (GANN) construction and search have become critical techniques in various domains, including recommendation systems [1], [2], computer vision [3], [4], and social networks [5], [6]. In these applications, GANN first performs construction to generate a graph that can guide searches, after which it searches the constructed graph to find the top-K nearest vertices to a given query. Unfortunately, considering that high-dimensional vertices consume substantial memory, the deployment of GANN in real-world scenarios is significantly constrained by memory bottlenecks [7]–[13]. Moreover, since the graph is **dynamically** updated during construction but remains **static** during searching, the construction stage is more complex. *Therefore, the goal of this work is to alleviate the memory bottleneck in GANN construction.*

* Zhuoran Song is the corresponding author.

This work is partly supported by the National Natural Science Foundation of China (Grant No. 62202288), the Natural Science Foundation of Shanghai (#23ZR1433200), and the Shanghai 2024 "Science and Technology Innovation Action Plan" Fundamental Research Program in Integrated Circuit (No. 24JD1400300).

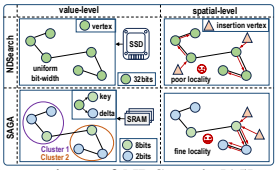


Fig. 1. Comparison of NDSearch [15] and SAGA.

To address the memory-bound problem of GANN, many existing works [14], [15] focus on accelerating GANN search. Taking NDSearch [15] as an example, it involves pre-processing the static graph by remapping the neighboring vertices on the same page of SSD to improve the search locality. However, since the graph is dynamically updated during GANN construction, deploying NDSearch in construction requires iteratively executing its pre-processing method at runtime, incurring significant latency and making GANN search accelerators ill-suited for construction. In contrast, our approach leverages the unique characteristics of GANN to accelerate the construction process. **Inspired by the *homophily theory* [16] in social networks, where users within the same community tend to share similar characteristics such as ages or interests, we discovered that neighboring vertices exhibit feature similarity. This similarity allows us to reduce precision in representing the differences between similar features, thus alleviating the memory bandwidth stress.** For example, given two vertices (20, 18, 10, 9) and (20, 17, 9, 8) and their difference (0, 1, 1, 1), naively storing each vertex takes 4 Bytes, while storing their difference only takes 1 Byte, saving 75% of memory storage.

In this paper, we propose **the first accelerator for GANN construction**, named SAGA. We observe that the first step in construction can naturally indicate vertices' similarity, so we carefully leveraged the intermediate results from this step to gather vertices into clusters and decompose vertices into one key vertex and multiple non-key vertices. To take advantage of value-level similarity within each cluster, we differentiate non-key vertices from key ones to construct deltas, which fall in a narrower range compared to the original vertices, making them more suitable for low-precision representation. Moreover, given that similar vertices are more likely to access the same vertex during graph traversal, we propose to exploit such a spatial-level similarity by rearranging the access order of vertices. To further illustrate the superiority of SAGA, Fig. 1 provides a comparison between NDSearch and our

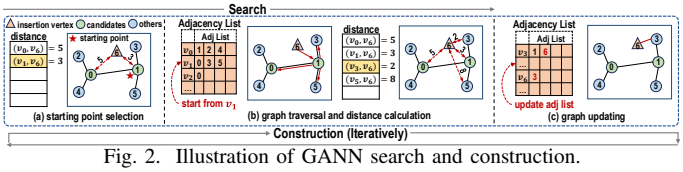


Fig. 2. Illustration of GANN search and construction.

approach. At the value level, NDSearch stores vertices in SSD with full precision, whereas we use a reduced bit width for delta storage, making GANN fit within on-chip SRAM. At the spatial level, NDSearch constructs vertices in a random order, while we reorder the execution sequence of similar vertices, thus achieving better locality.

Our contributions can be summarized as follows:

- From the algorithm perspective, we propose a similarity detection algorithm that gathers similar vertices into clusters. Then we introduce a quantization algorithm that leverages vertex similarity to reduce storage bit width.
- From the hardware perspective, we present an architecture design and develop a two-level scheduling that optimizes both the insertion and access order of vertices.

II. Background

A. GANN Construction and Search

GANN construction consists of three steps: starting point selection, distance calculation, and graph updating. First, starting point selection is used to reduce redundant calculations during construction. To simplify the search space, GANN samples certain vertices from the vertex set as candidates for the starting point, corresponding to vertices v_0 and v_1 in Fig. 2(a). Then, it calculates the distance from insertion vertex v_6 to these candidates and selects the closest candidate as the starting point. Typically, the distance is measured using either Inner Product (IP) or Euclidean (L2) distance, where the mathematical forms are $v_i \cdot v_j$ and $(v_i - v_j)^2$, respectively, with v_i and v_j being the vertices in the calculation. Second, distance calculation computes the distance between the insertion vertex and all neighbors of the starting point. In Fig. 2(b), v_6 sequentially computes distances to v_0, v_3, and v_5, which are 5, 2, and 8. Thus, the closest vertex to v_6 is identified as v_3. Notably, distance computation is iterative; if the neighbors of the starting point also have neighbors, they will also be calculated until the algorithm's pre-defined conditions are met. Finally, graph updating establishes a bi-directional link between the insertion vertex and its top-K closest neighbor. In Fig. 2(c), a bi-directional link is added between v_6 and v_3, and the adjacency list is then updated.

Given a constructed graph, GANN search uses the starting point selection and distance calculation steps to find the top-K nearest vertices to a given query, as shown in Fig. 2. Note that during the search process, the graph is completely static, while during construction, the graph is dynamic.

B. Related Works

There is a series of works on accelerating GANN [14], [15], but they all focus on searching rather than construction. SmartSSD [14] is the first architecture to accelerate GANN search. To make graph search algorithms more efficient on their hardware, SmartSSD pre-partitions the original static graph offline and performs searching separately on each sub-graph, narrowing the search scope. NDSearch [15] is the state-of-the-art accelerator for GANN search. NDSearch proposes remapping vertices on the SSD so that the neighboring vertices of each vertex are physically adjacent. To determine such remapping order, NDSearch conducts an expensive breath-first search (BFS) traversal of the entire graph during pre-processing. Unfortunately, both of their pre-processing approaches need to be iteratively executed whenever the graph is changed, which frequently happens during construction. As a result, their methods will introduce significant latency and may not be the proper option for GANN construction.

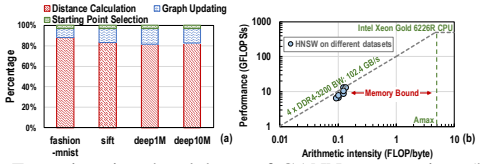


Fig. 3. (a) Execution time breakdown of GANN construction; (b) Roofline model in distance calculation.

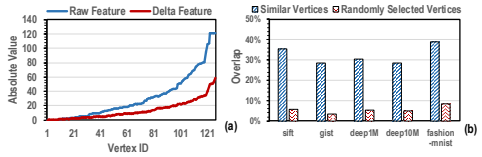


Fig. 4. (a) Value distribution in ascending order; (b) Graph traversal overlap.

III. Motivation

To find the performance bottlenecks in GANN construction, we conducted a breakdown analysis of the construction process on a CPU, which reveals that the majority of the execution time originates from distance computations, as shown in Fig. 3(a). Following this, we present our roofline model experiment for the distance calculation step in Fig. 3(b), demonstrating that all results fall into the left side of the roofline, which is in the DRAM bandwidth-constrained region. This sub-optimal performance is mainly due to the high dimension of vertices and their irregular memory accesses. This motivates us to find a solution that optimizes both storage and locality problems during distance calculations.

Observation 1: Vertex features contain redundancy at the value level. Fig. 4(a) visualizes the differences between neighboring vertices on sift [17] dataset. Compared to vertices' raw features, deltas have a narrower distribution, making them better suited for low-precision representation. Based on this observation, we propose quantizing deltas to a lower bit width, thus improving bandwidth utilization.

Observation2: Graph traversal exhibit reuse at the spatial level. As shown in Fig. 4(b), we compare the proportion of graph traversal overlap for similar vertices, and the proportion of graph traversal overlap for randomly selected vertices. The comparison between them shows that similar vertices are more likely to access the same vertex. This motivates us to bundle similar vertices together to enhance data locality.

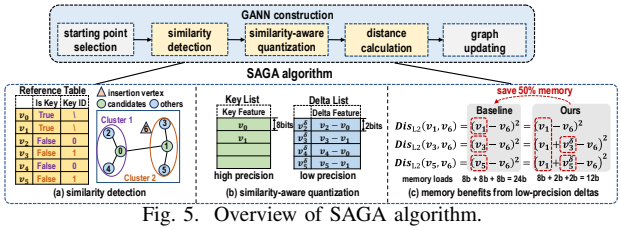


Fig. 5. Overview of SAGA algorithm.

IV. SAGA Algorithm

In this section, we introduce our SAGA algorithm, which comprises three essential operators: similarity detection, similarity-aware quantization and distance calculation.

A. Similarity Detection

To capture the similarity between vertices, we propose leveraging the intermediate results from the starting point selection step. This proposal builds on a well-known theory that users owning similar features in the community tend to connect through hubs [16]. Specifically, we find that candidates from construction's first step are naturally suited to serve as hubs, where vertices selecting the same candidate as a starting point are highly likely to be similar.

To implement similarity detection, we record the starting point of each insertion vertex and gather vertices with the same starting point into a single cluster. We designate this starting point as the **key** vertex, where other vertices are considered **non-key** vertices. To avoid duplicate storage of the same key vertex, each of the non-key vertices needs to record its key vertex's ID in a Reference Table. Fig. 5(a) shows an example of similarity detection: v_2 and v_4 both share v_0 as their starting point, so they are gathered into a cluster with v_0 as the key. In the Reference Table, we record whether each vertex is a key vertex (Is Key) and its corresponding key vertex's ID (Key ID). For instance, since v_2 is a non-key vertex, the Reference Table marks its Is Key identifier as False and records its Key ID as 0, allowing v_2 to directly reference its key vertex v_0.

B. Similarity-Aware Quantization

After identifying the similarities, we design a similarity-aware quantization algorithm to reduce storage bit width, which is a pre-processing step for distance calculations. Given the Reference Table and a non-key vertex v_i, we look up the key vertex v_j in the Reference Table and calculate the delta v_i^δ for v_i as follows: $v_i^\delta = v_i - v_j$. Next, we apply high-precision quantization to v_j with a wide range, while using low-precision quantization to v_i^δ with a narrow range. Fig. 5(b) illustrates our method: for key vertex v_0, we store its feature with 8-bit in the Key List; for non-key vertices, such as v_2, we compute its delta from v_0 and store it with 2-bit in the Delta List, recorded as v_2^δ. It is important to note that since there are numerous delta vertices but few key vertices, storing delta vertices with low precision effectively reduces the required bandwidth. Fig. 5(c) shows a specific example: with low-bit storage for deltas, the three distance calculations in our method require only 12 bits of memory loads, while the baseline approach requires 24 bits for the same calculations. Our method reduces memory overhead by 50%.

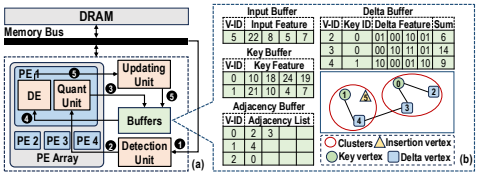


Fig. 6. Overview of (a) SAGA architecture; (b) Buffers (V-ID: Vertex ID).

C. Distance Calculation

Since we apply different bit widths for key and non-key vertices, the original distance calculation is no longer applicable, so we will discuss how to achieve distance calculation between insertion vertex v_{in} and both key and non-key vertices.

1. For key vertex v_j, we directly compute its distance to v_{in} by IP ($Dis_{IP}(v_{in}, v_j)$) or L2 ($Dis_{L2}(v_{in}, v_j)$). We store the distance in a partial sum table, indexed by v_j's ID, which will be used to avoid redundant computations for the same v_j.

2. For the non-key vertex v_i, the equations for the IP and L2 calculations are as follows:

$$Dis_{IP}(v_{in}, v_i) = Dis_{IP}(v_{in}, v_j) + Dis_{IP}(v_{in}, v_i^\delta) \quad (1)$$

$$\begin{aligned} Dis_{L2}(v_{in}, v_i) =& Dis_{L2}(v_{in}, v_j) - 2Dis_{IP}(v_{in} - v_j, v_i^\delta) \\ &+ Dis_{IP}(v_i^\delta, v_i^\delta) \end{aligned} \quad (2)$$

From the formulas, we see that the number of distance calculations increases from 1 to 2 (for the IP) and 3 (for the L2), which might seem to negate performance gains. Fortunately, we find that the distance calculations between v_{in} and v_i can be saved by reusing the distance between v_{in} and v_j ($Dis_{IP}(v_{in}, v_j)$ or $Dis_{L2}(v_{in}, v_j)$). Therefore, we retrieve $Dis_{IP}(v_{in}, v_j)$ or $Dis_{L2}(v_{in}, v_j)$ from the partial sum table and add it to the remaining terms in Eqn. 1 and Eqn. 2 to obtain the final result. Additionally, $Dis_{IP}(v_i^\delta, v_i^\delta)$ in the L2 distance calculation is also a constant value for each v_i^δ and can also be pre-computed during quantization.

D. Walk-Through Example

Fig. 5 also provides an example of how the SAGA algorithm integrates with the original construction process. Steps that remain unchanged are marked in grey, while steps improved by SAGA are highlighted in yellow. First, we perform the starting point selection, a step that remains unmodified. Next, SAGA leverages the results of the starting point selection for similarity detection, which in turn guides the similarity-aware quantization. Given the quantized vertices, SAGA calculates the distance based on whether a vertex is a key or a non-key vertex. Finally, the graph is updated based on the calculated distances. The above steps are repeated until all insertion vertices have been fully processed.

V. Proposed Hardware Design

A. Overview

To support our SAGA algorithm, we co-design a hardware architecture. Fig. 6(a) illustrates the overall architecture of SAGA, which includes several on-chip buffers, a Detection

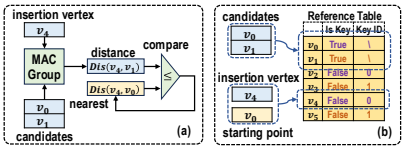


Fig. 7. Detection Unit for (a) starting point selection; (b) similarity detection.

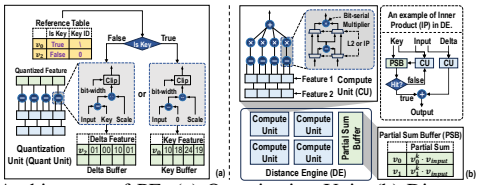


Fig. 8. Architecture of PE: (a) Quantization Unit; (b) Distance Engine.

Unit, an array of Processing Elements (PEs), and an Updating Unit. Each PE comprises a Quantization Unit (Quant Unit) and a Distance Engine (DE).

Dataflow. The dataflow in SAGA is also shown in Fig. 6(a), while the detailed buffer entries are illustrated in Fig. 6(b). ❶ To select the starting point, a batch of insertion vertices is transferred from DRAM to the Detection Unit via the memory bus. The Detection Unit then searches for the nearest starting point for each insertion vertex. ❷ In order to capture the similarity between vertices, the Detection Unit leverages the starting point information to generate a Reference Table, which stores whether each vertex is a key vertex and its Key ID. ❸ The Quant Unit performs quantization separately for key and non-key vertices and stores the features in Key Buffer and Delta Buffer, respectively. ❹ The DE retrieves the quantized features from Input Buffer, Key Buffer and Delta Buffer to compute distances. ❺ To update the constructed graph, the Updating Unit reads the distances computed by the DE, selects the top-K nearest vertices to the insertion vertex, and establishes bi-directional links in the Adjacency Buffer. The loop ❶ to ❺ repeats until the construction terminates.

B. Detection Unit

The Detection Unit is responsible for executing the starting point selection and similarity detection steps during the construction process. Fig. 7(a) illustrates how the Detection Unit selects starting points: given an insertion vertex, the Detection Unit uses a MAC group to compute its distance to all candidates and applies a comparator to select the nearest candidate as the insertion vertex's starting point. Fig. 7(b) depicts the similarity detection process. For candidates, the Detection Unit marks their Is Key identifiers as True. For insertion vertices, the Detection Unit sets their Is Key identifiers to False and records their starting point's ID as Key ID.

C. Processing Engine Array

The PE array of SAGA contains four PEs, each PE consists of a Quantization Unit and a Distance Engine.

Quantization Unit. The Quantization Unit (Quant Unit) is responsible for determining whether a vertex is a key vertex and performing its quantization. Fig. 8(a) depicts the architecture of the Quant Unit. The Quant Unit first determines whether a vertex is a non-key or key vertex based on the Is Key identifier recorded in the Reference Table. The Quant Unit then performs quantization and writes the quantized features into the corresponding buffer.

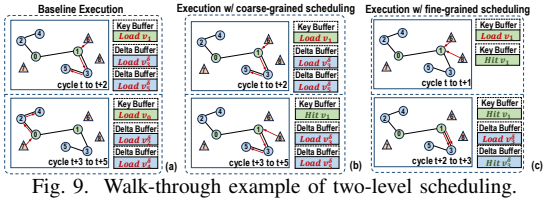


Fig. 9. Walk-through example of two-level scheduling.

Distance Engine. The Distance Engine (DE) is the most important component within the PE, responsible for computing the IP/L2 distances. As depicted in Fig. 8(b), each DE includes four Compute Units (CUs) dedicated to distance calculations and one Partial Sum Buffer (PSB). Each CU accepts two feature vectors as input and performs distance calculations based on the chosen metric (IP or L2); the PSB functions as a multi-way set-associative cache for storing reusable partial sums. Generally, a DE can simultaneously compute the distances for one insertion vertex and four neighboring vertices.

Based on the quantization algorithm outlined in Section IV-C, we will detail the distance calculations for both key and non-key vertices.

Key Vertex. When the insertion vertex requires a distance calculation with a key vertex, we retrieve v_{in} from the Input Buffer and v_j from the Key Buffer, then send them to the CU for the computation of the partial sum. Since vertices within the same cluster can reuse the partial sum, we store its value in the PSB and index it by the Key ID. If there is insufficient space in the PSB to accommodate the new partial sum, we identify an entry to evict according to the Least Recently Used (LRU) policy and replace it with the new partial sum.

Non-Key Vertex. According to Eqn. 1 and Eqn. 2, we trigger DE to compute $Dis_{IP}(v_{in}, v_i^\delta)$ or $Dis_{IP}(v_{in} - v_j, v_i^\delta)$. As for partial sum ($Dis_{IP}(v_{in}, v_j)$ or $Dis_{L2}(v_{in}, v_j)$), we first look it up in the PSB using the Key ID as an index. If it hits in PSB, we can retrieve the corresponding value; if it misses in PSB, we will compute the partial sum and store it in the PSB. At last, regarding the $Dis_{IP}(v_i^\delta, v_i^\delta)$ in Eqn. 2, we directly index it in the Sum column of Delta Buffer with V-ID.

Since key and delta features differ in bit width, we employ a bit-serial method in the CU to leverage the computational advantages of low bit width representation.

D. Updating Unit

The Updating Unit is used for sorting the distances and updating the graph accordingly. The Updating Unit receives the output from the DE and produces the top-K vertices with the smallest distances. To achieve this, Updating Unit employs a hardware implementation similar to that in [18], maintaining a minimum heap for up to one hundred input distances. After obtaining the top-K nearest vertices, the Updating Unit updates the corresponding entries in the Adjacency Buffer, completing the graph construction process.

E. Two-level Scheduling

In our previous design, we reduced storage and computational costs by lowering the bit width. However, due to the irregular memory access in construction, Key/Delta Buffers still encounter a high miss rate. Therefore, we propose a two-level scheduling that reorders vertex from both coarse-grained and fine-grained perspectives.

The coarse-grained scheduling rearranges the insertion order of the vertices, ensuring that vertices within the same cluster are executed sequentially. This is justified by the overlapping graph traversal paths of vertices in the same cluster. After coarse-grained scheduling, the fine-grained scheduling continues to reorder the access order of vertices. Considering that the reusable entries of insertion vertices in the Key/Delta Buffers may be evicted due to capacity constraints, we propose interleaving the distance computation for insertion vertices.

We then provide a walk-through example to illustrate the benefits of applying two-level scheduling. Fig. 9(a) shows the situation without scheduling, revealing poor locality in the Key/Delta Buffers. Fig. 9(b) illustrates the result of coarse-grained scheduling, where we reordered v_7 and v_8. This allows similar vertices v_6 and v_8 to be executed sequentially. It is evident that the coarse-grained scheduling strategy transforms v_8's access to v_1 from a miss to a hit, thereby improving the locality of the Key Buffer. However, due to the constrained capacity of the Delta Buffer, reusable entries (v_3^δ and v_5^δ) still get evicted, leading to a high miss rate in the Delta Buffer. Fig. 9(c) demonstrates the effect of applying both coarse-grained and fine-grained scheduling, where we interleave the distance calculations for v_6 and v_8, allowing for reuse of results in the Delta Buffer. With fine-grained scheduling, we make v_1 hit in Key Buffer and v_3^δ hit in Delta Buffer, demonstrating fine locality in both the Delta and Key Buffers.

VI. Evaluation

A. Experiment Methodology

Benchmark and datasets. We evaluated SAGA on HNSW [12], the most representative and widely cited GANN algorithm. The evaluation was conducted on five datasets from the ANN-benchmark [19]: sift [17], deep1M [20], deep10M [20], fashion-mnist [21], and gist [22]. For each dataset, we use the recommended parameter settings and distance metrics from the ANN-benchmark during construction.

Experiment Setup. To model the behavior of SAGA, we build a cycle-accurate simulator. We implement and synthesize the digital logic circuits at the 28nm technology node with 500MHz using Synopsys Design Compiler. The on-chip buffers are evaluated using CACTI 7.0 [23] with a 32nm technology node, and we use the DeepScaleTool [24] to scale the results down to 28nm. For CPU and GPU, we selected FAISS [25] and cuhnsw [26] libraries as implementations of HNSW. We use an Intel Xeon Gold 6226 CPU running at 2.0GHz as the baseline for the CPU, and an A100 GPU with 80GB VRAM as the baseline for the GPU. To compare SAGA with previous GANN accelerators, we selected the state-of-the-art NDSearch [15] as a baseline. As GANN iteratively updates the graph during construction, the BFS introduced by NDSearch must be re-executed after each update. Since NDSearch does not incorporate additional hardware to support BFS, we implemented BFS on the CPU using the high-performance kernel from GAP [27]. To ensure a fair comparison, we scaled NDSearch's compute units and buffers under the same area budget as SAGA.

TABLE I
Comparisons with NDSearch (FP32) and SAGA (CR: Compression Ratio).

Dataset	Config	Accuracy	Average Bits	CR
sift [17]	NDSearch	96.0±2.0%	32	1×
	SAGA	94.2±2.1%	2.19	14.63×
gist [22]	NDSearch	94.7±2.0%	32	1×
	SAGA	93.4±2.4%	2.07	15.48×
deep1M [20]	NDSearch	95.3±2.0%	32	1×
	SAGA	94.2±2.4%	2.36	13.52×
deep10M [20]	NDSearch	95.9±1.6%	32	1×
	SAGA	94.6±2.1%	2.43	13.18×
fashion-mnist [21]	NDSearch	97.8±0.9%	32	1×
	SAGA	96.1±1.4%	2.29	13.97×

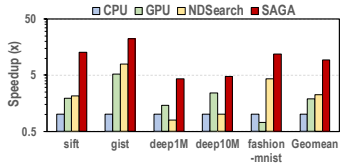


Fig. 10. The speedup of SAGA over CPU, GPU and NDsearch.

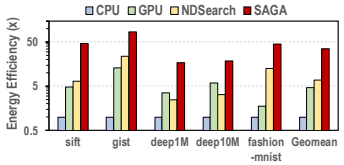


Fig. 11. The energy efficiency of SAGA over CPU, GPU and NDsearch.

B. Accuracy

In TABLE I, we present a comparison of accuracy and compression ratio between our method and NDSearch across various datasets. Extensive experiments demonstrate that our approach achieves a high compression ratio while maintaining accuracy. We leveraged the value-level similarity between vertices by using high-bitwidth representations for critical key vertices, while employing low-bitwidth representations for the majority of non-key vertices. This approach achieves a well-balanced trade-off between accuracy and compression ratio.

C. Overall Results

Speedup Fig. 10 illustrates the speedup of SAGA. Generally, SAGA achieves an average speedup of 9.30×, 4.87× and 4.14× over CPU, GPU and NDSearch. In comparison with CPU, SAGA manifests the benefits of customized architecture, which offers higher bandwidth utilization and better on-chip buffer locality. Compared to GPU, although they offer high throughput, GANN construction is not a compute-centric application, which leads to low utilization of the GPU cores. For example, in the fashion-mnist dataset, we observe that the GPU performance is even lower than the CPU. Additionally, SAGA outperforms the state-of-the-art GANN accelerator NDSearch because our quantization method achieves a higher

TABLE II
AREA AND POWER COMPARISON BETWEEN SAGA AND NDSEARCH.

-	SAGA		
	Config	Area (mm^2)	Power (mW)
Detection Unit	1×(32-bit)	0.19	5.51
Distance Engine	4×(bit-serial)	0.35	97.73
Quant Unit	4×(8-bit)	0.07	21.72
Updating Unit	1×(8-bit)	0.03	5.43
On-chip Buffers	320KB	0.62	197.34
Overall	-	**1.26**	**327.73**
-	NDSearch		
	Config	Area (mm^2)	Power (mW)
Detection Unit	1×(32-bit)	0.19	5.51
Distance Engine	512 MACs(32-bit)	0.58	173.75
Quant Unit	-	-	-
Updating Unit	1×(32-bit)	0.12	21.71
On-chip Buffers	320KB	0.64	202.40
Overall	-	**1.53**	**403.37**

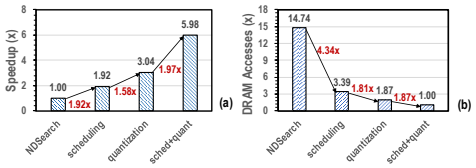


Fig. 12. The contribution of each method to (a) the improvement of performance and (b) the DRAM access reduction on the sift dataset.

compression ratio (see Table I). Moreover, we observe that the BFS operations performed by NDSearch introduce significant overhead. In the deep1M experiment, NDSearch even fails to outperform the CPU baseline, with BFS accounting for 77% of its execution time. Therefore, NDSearch is not well-suited for dynamic construction processes.

Energy Efficiency. Fig. 11 shows the energy efficiency of SAGA. For energy efficiency, SAGA offers an average energy efficiency of 35.48×, 7.60× and 5.15× over CPU, GPU and NDSearch. Thanks to SAGA's quantization algorithm, it achieves higher bandwidth utilization, significantly reducing the overall DRAM access time. Additionally, SAGA's scheduling algorithm enhances the locality of on-chip buffers and further reduces the volume of DRAM read and write, leading to substantial energy savings in memory access.

Area and Power. The area and power consumption of SAGA are summarized in Table II. SAGA utilizes only a small number of computational engines and units, as GANN construction is not a compute-intensive task. Compared to NDSearch, SAGA maintains the same buffer capacity but can store more data due to its lower bit width representations, significantly enhancing locality. Note that SAGA achieves a 17.65% reduction in area and an 18.75% reduction in power compared to NDSearch. This is attributed to SAGA's low-bitwidth hardware design, whereas NDSearch is designed for full precision.

D. Detailed Analysis

Ablation study. We conduct the ablation study on the proposed methods, including two-level scheduling (*scheduling*), similarity-aware quantization (*quantization*) and the combination of scheduling and quantization (*sched+quant*), which is the final version of SAGA. Fig. 12(a) illustrates the performance improvements achieved by SAGA's methods. As shown in the figure, SAGA achieves a 1.92× speedup due to enhanced buffer locality enabled by *scheduling*. SAGA also achieves a 3.04× speedup by leveraging lower bit width for storage using *quantization*. Finally, when combing *scheduling* and *quantization*, SAGA achieves a remarkable 5.98× speedup. Fig. 12(b) further demonstrates SAGA's improvement in DRAM accesses, our proposed methods achieve DRAM reductions of 4.34×, 7.88× and 14.74×, respectively.

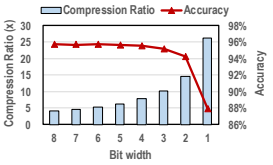


Fig. 13. Compression ratio (higher is better) and accuracy (higher is better) compared with NDSearch under different bit widths on the sift dataset.

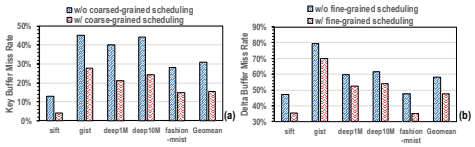


Fig. 14. The effect of scheduling in (a) Key Buffer miss rate (lower is better) and (b) Delta Buffer miss rate (lower is better).

Parameter Trade-off. In SAGA, we need to determine the bit width for deltas to balance the compression ratio and accuracy. Note that the key vertex is represented using 8-bit. As shown in Fig. 13, when the bit width decreases, SAGA achieves higher compression ratio, reaching up to 26.26× when the bit width is 1. However, the accuracy in this case drops significantly to 87.9%, which is unacceptable. Therefore, we select 2-bit for deltas, achieving a compression ratio of 14.63× while maintaining a high accuracy of 94.2%.

Scheduling. To evaluate the two-level scheduling, we explore the impact of coarse/fine-grained scheduling on the Key/Delta Buffers, respectively. Fig. 14 verifies that both coarse/fine-grained scheduling can effectively reduce miss rates of the two buffers. This improvement is due to the enhanced temporal locality of vertex accesses.

VII. CONCLUSION

In this paper, we propose SAGA, the first accelerator to leverage similarity to address the memory-bound challenges in GANN construction. To capture similarity, we first introduce a similarity detection algorithm that clusters vertices based on their starting points. Next, to reduce storage overhead, we propose a similarity-aware quantization algorithm that quantizes the deltas of similar vertices to lower bit width. We design a hardware architecture and develop a two-level scheduling algorithm to optimize locality in buffers. Experimental results demonstrate that SAGA achieves an average 4.15× speedup and 5.15× energy efficiency compared to the state-of-the-art NDSearch, with negligible accuracy loss.

REFERENCES

[1] Y. Park, S. Park, W. Jung, and S.-g. Lee, "Reversed cf: A fast collaborative filtering algorithm using a k-nearest neighbor graph," *Expert Systems with Applications*, vol. 42, no. 8, pp. 4022–4028, 2015.

[2] Y. Meng, X. Dai, X. Yan, J. Cheng, W. Liu, J. Guo, B. Liao, and G. Chen, "Pmd: An optimal transportation-based user distance for recommender systems," in *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42*. Springer, 2020, pp. 272–280.

[3] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li, "Scalable k-nn graph construction for visual descriptors," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 1106–1113.

[4] A. Kosuge and T. Oshima, "An object-pose estimation acceleration technique for picking robot applications by using graph-reusing k-nn search," in *2019 First International Conference on Graph Computing (GC)*. IEEE, 2019, pp. 68–74.

[5] N. Chen, "On the approximability of influence in social networks," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 3, pp. 1400–1415, 2009.

[6] S. Kumar and P. Kumar, "Upper approximation based privacy preserving in online social networks," *Expert Systems with Applications*, vol. 88, pp. 276–289, 2017.

[7] C. Fu, C. Wang, and D. Cai, "High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4139–4150, 2021.

[8] A. Singh, S. J. Subramanya, R. Krishnaswamy, and H. V. Simhadri, "Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search," *arXiv preprint arXiv:2105.09613*, 2021.

[9] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *arXiv preprint arXiv:1707.00143*, 2017.

[10] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[11] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th international conference on World wide web*, 2011, pp. 577–586.

[12] Y. A. Malkov and D. A. Yashunin, "Hnsw: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[13] M. Wang, X. Xu, Q. Yue, and Y. Wang, "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 1964–1978, Jul. 2021. [Online]. Available: https://doi.org/10.14778/3476249.3476255

[14] J.-H. Kim, Y.-R. Park, J. Do, S.-Y. Ji, and J.-Y. Kim, "Accelerating large-scale graph-based nearest neighbor search on a computational storage platform," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 278–290, 2022.

[15] Y. Wang, S. Li, Q. Zheng, L. Song, Z. Li, A. Chang, Y. Chen *et al.*, "Ndsearch: Accelerating graph-traversal-based approximate nearest neighbor search through near data processing," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 368–381.

[16] M. Mcpherson, L. S. Lovin, and J. Cook, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, 2001.

[17] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 1815–1822.

[18] R. Bhagwan and B. Lin, "Fast and scalable priority queue architecture for high-speed network switches," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 2. IEEE, 2000, pp. 538–547.

[19] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Information Systems*, vol. 87, p. 101374, 2020.

[20] A. Babenko and V. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2055–2063.

[21] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[22] H. Sandhawalia and H. Jégou, "Searching with expectations," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2010, pp. 1242–1245.

[23] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[24] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[25] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," 2024.

[26] "Cuda implementation of hierarchical navigable small world graph algorithm." [Online]. Available: https://github.com/js1010/cuhnsw

[27] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

Process-Variation-Aware Design Optimization for Wavelength-Routed Optical Networks-on-Chip

Liaoyuan Cheng[1], Mengchu Li[1], Tsun-Ming Tseng[1], Martin Schottenloher[2], Ulf Schlichtmann[1]
[1]{liaoyuan.cheng, mengchu.li, tsun-ming.tseng, ulf.schlichtmann}@tum.de, [2]martin@schottenloher.de
[1]Technical University of Munich, Germany, [2]Ludwig Maximilian University of Munich, Germany

***Abstract*—With low latency and collision-free communication, wavelength-routed optical networks-on-chip (WRONoC) become an effective solution to the growing demands for multi-core communications. Microring resonators (MRRs), the primary optical components in WRONoC, are susceptible to process variation. Under process variation, the MRR's transmission spectrum shifts, results reduced signal power and increased crosstalk. However, the impacts caused by process variation have not yet been considered in WRONoC designs. In this work, we propose a methodology to optimize the MRR radii and signal wavelengths to counter process variation. Specifically, we construct analytical models of expected signal power under MRR process variation and develop optimization methods to maximize the expected signal transmission power in WRONoC. Results show up to 7.51 dB improvement in worst-case expected signal transmission power compared to designs that do not consider process variation.**

***Index Terms*—Wavelength-routed optical networks-on-chip, design optimization, process-variation-aware**

I. Introduction

With the development of silicon photonics, optical networks-on-chip (ONoCs) become an effective solution to the growing demand for inter-core communications in multiprocessor systems-on-chip (MPSoCs) [1], [2]. With the wavelength-division multiplexing technology, a single waveguide can carry multiple optical signals on different wavelengths, providing high bandwidths [3], [4]. Wavelength-routed optical networks-on-chip (WRONoC) is a promising type of ONoC. On WRONoCs, all signal paths are fixed during the design phase, which eliminates the overhead for dynamic allocation paths and allows all initiators and targets to communicate simultaneously without data conflicts [5], [6].

The primary WRONoC components for signal routing are microring resonators (MRRs). An MRR consists of a looped waveguide and a coupling mechanism. Optical signals on different wavelengths can be coupled to different MRRs in the network and directed to different transmission paths. The coupling degree depends on the MRR radius and the signal wavelength [7], [8]. If the optical path circumference of an MRR is an integer multiple of the wavelength of a signal, the signal will be maximally coupled to the MRR, referred to as being *on-resonance* with the MRR. An MRR can resonate with multiple wavelengths, and these wavelengths are referred to as the *resonant wavelengths* of that MRR. On the other hand, signals on wavelengths other than the resonant wavelengths of an MRR will be hardly coupled to the MRR, referred to as being *off-resonance* with the MRR. Fig. 1(a) illustrates

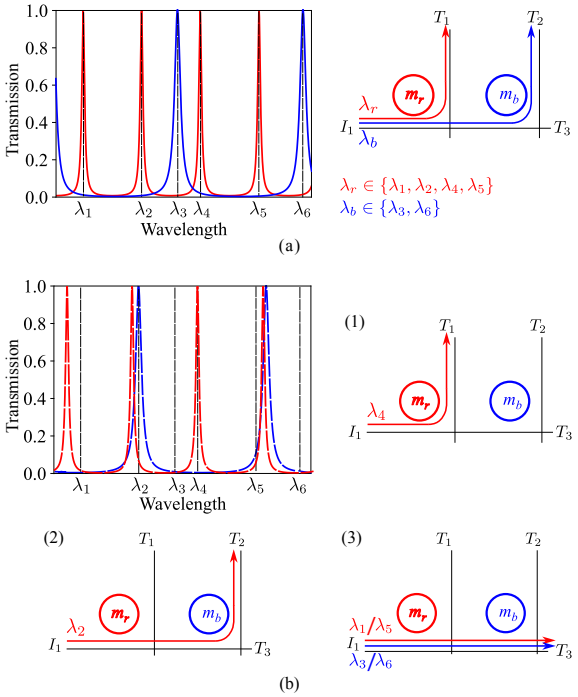


Fig. 1. (a) MRR coupling mechanism without process variation. Peaks of the red and blue spectra represent the resonant wavelengths of m_r and m_b, respectively. (b) MRR coupling mechanism considering process variation. (1) λ_4 is on-resonance with m_r despite the process variation. (2) λ_2 becomes on-resonance with m_b instead of m_r due to the process variation. (3) λ_1, λ_3, λ_5, and λ_6 become off-resonance with both m_r and m_b due to the process variation.

the coupling mechanism. The red and blue circles represent two MRRs of different radii, denoted as m_r and m_b, the transmission spectra of which are represented as red and blue curves. In particular, λ_1, λ_2, λ_4, λ_5 are on-resonance with m_r and off-resonance with m_b. λ_3, λ_6 are on-resonance with m_b and off-resonance with m_r. Thus, to send a signal from initiator I_1 to target T_1, we can modulate the signal on wavelength λ_1, λ_2, λ_4, or λ_5. To send a signal from I_1 to T_2, we can modulate the signal on wavelength λ_3 or λ_6.

However, MRRs are susceptible to process variation, which leads to changes in their radii and transmission spectra [9]. Fig. 1(b) illustrates an example of the shifts of the MRR transmission spectra due to the changes of the radii of m_r and m_b resulting from the process variation. Specifically, the influences of process variation on different wavelengths and

MRRs are different. For example, λ_4 remains on-resonance with m_r despite the process variation, as shown in Fig. 1(b)(1). However, λ_2, which was supposed to be on-resonance with m_r, becomes on-resonance with m_b instead, leading to the signal being misdirected to T_2 instead of T_1, resulting in crosstalk at this unintended destination, as shown in Fig. 1(b)(2). Also, λ_1, λ_3, λ_5, and λ_6, which were supposed to be on-resonance with m_r or m_b, are off-resonance with both m_r and m_b under process variation, and become crosstalk in the network, as shown in Fig. 1(b)(3). The example shows that different wavelengths and MRR radii react differently to process variation, and the wavelengths assigned to different signal paths and the radii assigned to different MRRs interact and thus need to be optimized to find robust radius and wavelength configurations under process variations.

Several approaches have been proposed to deal with the process variation in the MRR. [9] and [10] propose methods to adjust the physical design parameters of MRR to reduce the power loss as signals enter the MRR. Similarly, [11] introduces a sparse combined generalized polynomial chaos (gPC) model for analyzing the uncertainty and optimizing the design parameters of a five-ring coupled resonator filter under process variation, thereby enhancing the robustness. However, these studies are limited to adjusting the parameters of only one type of MRR for a specific signal wavelength range near 1550 nm. They cannot comprehensively optimize the parameters of all MRRs in a network. Besides, they do not consider common WRONoC communication scenarios in which an MRR may be passed by multiple signals, and a signal may pass through multiple MRRs along its path. Considering the process variation that may occur in WRONoC, [12] and [13] suggest adding backup MRRs to improve network reliability. However, this approach inevitably results in signals passing through additional MRRs, thereby increasing insertion loss.

In this work, we propose the first analytical model to characterize the expected signal transmission efficiency under process variation given different MRR radii and signal wavelengths. Based on the analytical model, we propose an integer linear programming (ILP) method to optimize the selection of MRR radii and signal wavelengths to maximize the worst-case transmission efficiency of all signals in the network. Since the complexity of the ILP problem is NP-hard, we further develop a WRONoC-tailored simulated annealing algorithm to efficiently approximate the global optimum. Experimental results show that our method can significantly improve the expected network performance and make WRONoCs more robust against process variation.

II. Background

There are two common optical switching elements: One is the parallel switching element (PSE), in which two waveguides are placed in parallel at the top and bottom of the ring. The other is the crossing switching element (CSE), where the waveguides are crossed orthogonally, and a ring is placed near the crossing waveguides. When a signal enters the switching element from the input port, a portion of its power will be coupled to the MRR and delivered to the drop port, while the remaining portion will exit the MRR via the through port. An MRR is designed to be a **drop MRR** if a signal is on-resonance and maximally transmitted to its drop port. Conversely, it is designed to be a **through MRR** if a signal is off-resonance and maximally transmitted to its through port. The portion of the signal power at the expected port is defined as the signal **transmission efficiency** at that MRR. For a given MRR radius r, and a given signal wavelength λ, the transmission efficiency of the signal at a drop or through MRR, denoted as $H_d(r,\lambda) \in [0,1]$ or $H_t(r,\lambda) \in [0,1]$, can be computed as follows [7]:

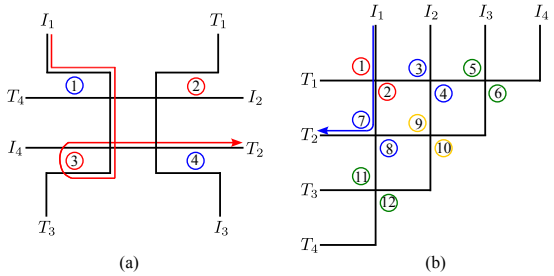


Fig. 2. WRONoC topologies with four ports. (a) Light Topology. The red line represents a signal path from initiator I_1 to T_2. (b) Snake Topology. The blue line highlights a signal path from initiator I_1 to T_2.

$$H_d(r,\lambda) = \frac{k^4}{1 - 2t^2 cos(\phi(r,\lambda)) + t^4} \tag{1}$$

$$H_t(r,\lambda) = 1 - H_d(r,\lambda) \tag{2}$$

where t and k denote the self-coupling and the cross-coupling coefficients with $t^2 + k^2 = 1$, and $\phi(r,\lambda)$ denotes the signal-pass phase shift [14], calculated as:

$$\phi(r,\lambda) = \beta(\lambda) 2\pi r \tag{3}$$

$$\beta(\lambda) = \frac{(2.57 - 0.85(\lambda \cdot 10^6 - 1.55))2\pi}{\lambda} \tag{4}$$

where $\beta(\lambda)$ is the propagation constant [14].

A WRONoC topology can be defined by a set $\mathcal{I}$ of initiators, a set $\mathcal{T}$ of targets, a set $\mathcal{P} \subseteq \mathcal{I} \times \mathcal{T}$ of signal paths, and a set $\mathcal{M}$ of MRRs. For a signal path $p \in \mathcal{P}$, we use $\mathcal{TM}_p \subseteq \mathcal{M}$, $\mathcal{DM}_p \subseteq \mathcal{M}$, and $\mathcal{M}_p = \mathcal{TM}_p \cup \mathcal{DM}_p$ to represent the set of through MRRs, drop MRRs and total MRRs along p, respectively. We also denote c_p as the number of waveguide crossings in p. Two representive topologies, Light [6] and Snake [15], are shown in Fig. 2.

Example 2.1: Fig. 2(a) shows a Light topology [6] with $\mathcal{I} = \{I_1, I_2, I_3, I_4\}$, $\mathcal{T} = \{T_1, T_2, T_3, T_4\}$, $\mathcal{P} = \{(I_1, T_2), (I_1, T_3), \cdots, (I_4, T_3)\}$, and $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$. The signal path $(I_1, T_2) \in \mathcal{P}$ can be described with $\mathcal{TM}_{(I_1,T_2)} = \{m_1, m_4\}$, $\mathcal{DM}_{(I_1,T_2)} = \{m_3\}$, $\mathcal{M}_p = \{m_1, m_3, m_4\}$ and $c_{(I_1,T_2)} = 4$.

For each signal path, the portion of the signal power received by the target is defined as the signal **transmission efficiency of the path**, denoted as H_p. For a given signal on wavelength λ, H_p can be calculated based on the transmission

efficiency of the signal at the MRRs and the waveguide crossings along the path:

$$H_p = (1-cl)^{c_p} \prod_{m\in\mathcal{DM}_p} H_d(r_m,\lambda) \prod_{m\in\mathcal{TM}_p} H_t(r_m,\lambda) \quad (5)$$

where cl denotes the crossing loss parameter, r_m denotes the radius of MRR m and c_p denotes the waveguide crossing number in p.

III. Analytical Model of the Expected Signal Transmission Efficiency

MRRs are sensitive to process variations, under which the radii may change, leading to shifts in the transmission spectra. Consequently, the transmission efficiency of the signal path will also be varied.

Based on the fact that photonic circuit fabrication is non-uniform and that the fabrication data can be fit to a normal distribution [16]–[18], we model the radius of an MRR as a normally distributed random variable $R \sim N(r, (\eta\cdot r)^2)$, where $r \in \mathbb{R}^+$ is the mean value serving as the nominal radius of the MRR, η is the relative error resulting from the process variation, and $\eta \cdot r$ is the standard deviation. The **expected signal transmission efficiency**, which is the expected value of the signal transmission efficiencies at various possible radii of an MRR, weighted by their respective probabilities in the normal distribution, can be then modeled. The expected signal transmission efficiency at the drop or through port of the MRR can be denoted as $\mathbb{E}[H_d(R,\lambda)]$ or $\mathbb{E}[H_t(R,\lambda)] \in [0,1]$. To derive these, we need the following theorem [19]:

Theorem 3.1: The expected value of a non-negative continuous random variable X can be computed as $\mathbb{E}(X) = \int_0^\infty (1-F_X(x))\,dx$ with cumulative distribution function (CDF) $F_X(x) = P(X \le x)$, where $P(X \le x)$ is the probability of $X \le x$.

We denote $H_d(R,\lambda)$ as X, and the CDF of $H_d(R,\lambda)$ as $F_X(x)$, which can be calculated based on equation (2) as:

$$\begin{aligned} F_X(x) &= P\left(H_d(R,\lambda) \le x\right) \\ &\overset{(4)}{=} P(\cos(2\pi R\beta(\lambda)) \le (1-k^4/x+t^4)/(2t^2)). \end{aligned} \quad (6)$$

For convenience, we denote $\frac{1-\frac{k^4}{x}+t^4}{2t^2}$ as $g(x)$. Thus:

$$\begin{aligned} F_X(x) &= \sum_{n\in\mathbb{Z}} P\left(2n\pi + \arccos(g(x)) \le 2\pi R\beta(\lambda) \le \right. \\ &\qquad \left. 2(n+1)\pi - \arccos(g(x))\right) \\ &= \sum_{n\in\mathbb{Z}} (P(R \le (2\pi(n+1) - \arccos(g(x)))/(2\pi\beta(\lambda))) \\ &\qquad - P(R \le (2n\pi + \arccos(g(x)))/(2\pi\beta(\lambda)))). \end{aligned} \quad (7)$$

According to [19], the CDF of $R \sim N(r,(\eta\cdot r)^2)$ can be denoted as $P(R \le y) = \Phi\left(\frac{y-r}{\eta\cdot r}\right)$, where $y \in \mathbb{R}$ and Φ denotes the CDF of the standard normal distribution $Z \sim N(0,1)$. Therefore, we obtain

$$\begin{aligned} &P(R \le (2\pi(n+1) - \arccos(g(x)))/(2\pi\beta(\lambda))) \\ &= \Phi(((2\pi(n+1) - \arccos(g(x)))/(2\pi\beta(\lambda)) - r)/(\eta\cdot r)), \end{aligned}$$

$$P(R \le \frac{2n\pi + \arccos(g(x))}{2\pi\beta(\lambda)}) = \Phi(\frac{\frac{2n\pi+\arccos(g(x))}{2\pi\beta(\lambda)} - r}{\eta\cdot r})$$

which implies that

$$\begin{aligned} F_X(x) = \sum_{n\in\mathbb{Z}} [&\Phi((\frac{2\pi(n+1)-\arccos(g(x))}{2\pi\beta(\lambda)} - r)/(\eta\cdot r)) \\ &- \Phi((\frac{2n\pi+\arccos(g(x))}{2\pi\beta(\lambda)} - r)/(\eta\cdot r))]. \end{aligned} \quad (8)$$

We know that $x \in [k^4/(2-k^2)^2, 1]$ from equation (2). Thus, according to Theorem 3.1, we can calculate $\mathbb{E}[H_d(R,\lambda)]$ as:

$$\begin{aligned} \int_{\frac{k^4}{(2-k^2)^2}}^1 (1 - \sum_{n\in\mathbb{Z}} [&\Phi((\frac{2(n+1)\pi-\arccos(g(x))}{2\pi\beta(\lambda)} - r)/(\eta\cdot r)) \\ &- \Phi((\frac{2n\pi+\arccos(g(x))}{2\pi\beta(\lambda)} - r)/(\eta\cdot r))])\,\mathrm{d}x. \end{aligned} \quad (9)$$

To determine the range of n in the equations above, we first reference equation (7), which establishes $2n\pi + \arccos(g(x)) \le 2\pi R\beta(\lambda) \le 2(n+1)\pi - \arccos(g(x)))$. Given that the range of function $arccos$ within its domain is $[0,\pi]$, it follows that the maximum possible range of $2\pi R\beta(\lambda)$ is $2n\pi \le 2\pi R\beta(\lambda) \le 2(n+1)\pi$. From this, we derive the bounds for n as $R\beta(\lambda) - 1 \le n \le R\beta(\lambda)$. Considering R as a random variable requires the full range of its possible values to be considered. The three-sigma rule of a normal distribution states that 99.7% of the values for R fall within $[r-3\eta r, r+3\eta r]$ [20]. To include a broader range of possibilities, we extend this range to $[r-6\eta r, r+6\eta r]$, thus estimating the range of n as $[\lfloor(r-6\eta r)\beta(\lambda)-1\rfloor, \lfloor(r+6\eta r)\beta(\lambda)\rfloor]$. We divide the integration domain into 1000 equal subintervals and use the Left Riemann Sum to approximate the expected value.

Next, we need another theorem from [19] to calculate $\mathbb{E}[H_t(R,\lambda)]$:

Theorem 3.2: Let X and Y be two random variables, then $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$.

According to Theorem 3.2 and equation (2):

$$\begin{aligned} &\mathbb{E}[H_t(R,\lambda)] + \mathbb{E}[H_d(R,\lambda)] = \mathbb{E}[H_t(R,\lambda) + H_d(R,\lambda)] = 1. \\ &\mathbb{E}[H_t(R,\lambda)] = 1 - \mathbb{E}[H_d(R,\lambda)]. \end{aligned} \quad (10)$$

Subsequently, the expected transmission efficiency of a signal path, denoted by $\mathbb{E}[H_p]$, can be calculated as

$$(1-cl)^{c_p} \prod_{m\in\mathcal{DM}_p} \mathbb{E}[H_d(R_m,\lambda)] \prod_{m\in\mathcal{TM}_p} \mathbb{E}[H_t(R_m,\lambda)] \quad (11)$$

where $R_m \sim N(r_m, (\eta\cdot r_m)^2)$.

Finally, we define the **worst-case expected transmission efficiency**, denoted as $\mathbb{E}[H_w]$, as follows:

$$\mathbb{E}[H_w] = \min\{\mathbb{E}[H_p] | p \in \mathcal{P}\}. \quad (12)$$

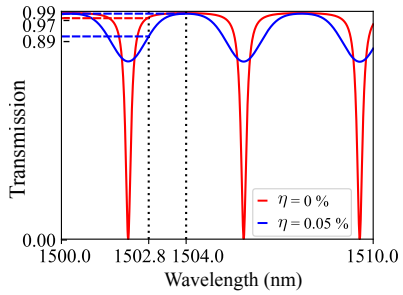


Fig. 3. Transmission efficiencies of signals on different wavelengths at the through port of an MRR.

IV. Problem Formulation

The expected signal transmission efficiency may differ given different combinations of MRR radii and signal wavelengths.

Example 4.1: As an example, Fig. 3 illustrates the transmission efficiency of signals in the wavelength range $\lambda \in [1500\,\text{nm},\ 1510\,\text{nm}]$ with a step size of $0.01\,\text{nm}$ at the through port of an MRR with a nominal radius of $25\,\mu\text{m}$, i.e. $r = 25 \cdot 10^{-6}$. The red curve represents the transmission efficiency calculated based on the nominal radius without considering process variation, i.e., $H_t(r, \lambda)$; and the blue curve represents the approximate expected transmission efficiency considering process variation, i.e., $\mathbb{E}[H_t(R, \lambda)]$, with $\eta = 0.05\%$, $R \sim N(r, (\eta r)^2)$ and $k = 0.4$ [10]. We highlight two signals on wavelengths $1502.8\,\text{nm}$ and $1504\,\text{nm}$. While both signals exhibit high transmission efficiency with $H_t(r, 1502.8 \cdot 10^{-9}) = 97\%$ and $H_t(r, 1504 \cdot 10^{-9}) = 99\%$ without considering process variation, the expected transmission efficiency $\mathbb{E}[H_t(R, 1502.8 \cdot 10^{-9})]$ drops to 89%, while $\mathbb{E}[H_t(R, 1504 \cdot 10^{-9})]$ remains at 99%. Thus, $1504\,\text{nm}$ is a more robust signal wavelength for a through MRR with $25\,\mu\text{m}$ nominal radius under process variation.

In this work, we propose to optimize the selections of MRR radii and signal wavelengths in a WRONoC topology to maximize the worst-case expected transmission efficiency, which determines the minimum laser power required to support the desired communications and is thus considered the most critical design factor. Therefore, we formulate the optimization problem as follows:

Input: (1) A WRONoC topology. (2) A range of MRR radii. (3) A range of signal wavelengths. (4) The relative error η. (5) The crossing loss coefficient cl. (6) The cross-coupling coefficient k.

Output: (1) The nominal radius of every MRR in the topology. (2) The wavelength used in every signal path.

Maximize: Worst-case expected transmission efficiency $\mathbb{E}[H_w]$.

V. Methods

To prepare for the optimization, we construct a set $\mathcal{R}$ of MRR nominal radii options and a set Λ of signal wavelengths options based on the input ranges and the step size between two options. Next, for each combination of the MRR nominal radius $r \in \mathcal{R}$ and for each signal wavelength $\lambda \in \Lambda$, we calculate the expected transmission efficiency of the signal at the drop or through port of the MRR in dB, denoted as $\mathbb{E}[H_d(R, \lambda)]_{\text{dB}}$ or $\mathbb{E}[H_t(R, \lambda)]_{\text{dB}}$, with $R \sim N(r, (\eta \cdot r)^2)$, based on our analytical model:

$$\mathbb{E}[H_d(R, \lambda)]_{\text{dB}} = 10 \log_{10} \mathbb{E}[H_d(R, \lambda)] \tag{13}$$

$$\mathbb{E}[H_t(R, \lambda)]_{\text{dB}} = 10 \log_{10} \mathbb{E}[H_t(R, \lambda)]. \tag{14}$$

A. Integer Linear Programming

We propose an integer linear programming method to formally model the optimization problem.

1) Radii and Wavelengths Assignment: We introduce a binary variable $b_{m,r}^{p,\lambda}$ for each signal path $p \in \mathcal{P}$ in the input topology, each MRR $m \in \mathcal{M}_p$ in path p, each nominal radius option $r \in \mathcal{R}$ and each wavelength option $\lambda \in \Lambda$ to represent whether the transmission efficiency of the signal in path p at MRR m should be calculated based on λ and r, with the following constraint:

$$\forall p \in \mathcal{P}\ \forall m \in \mathcal{M}_p : \sum_{r \in \mathcal{R}} \sum_{\lambda \in \Lambda} b_{m,r}^{p,\lambda} = 1. \tag{15}$$

2) Consistent Wavelength Selection: A signal path may involve multiple MRRs, and the transmission efficiency at different MRRs along the path should be calculated based on the same wavelength. In other words, if a wavelength is used for the calculation of the transmission efficiency at any MRR along the path, the same wavelength will also be used for the calculation of the transmission efficiency at other MRRs along the path. This can be modelled with the following constraint, where $m_p \in \mathcal{M}_p$ can be arbitrarily selected from $\mathcal{M}_p$:

$$\forall p \in \mathcal{P}\ \forall \lambda \in \Lambda\ \forall m \in \mathcal{M}_p \setminus \{m_p\} : \sum_{r \in \mathcal{R}} b_{m,r}^{p,\lambda} = \sum_{r' \in \mathcal{R}} b_{m_p,r'}^{p,\lambda}. \tag{16}$$

3) Consistent Radii Selection: An MRR m can appear in multiple signal paths, and the transmission efficiency at the same MRR in different signal paths should be calculated based on the same radius. Similar to Section V-A2, we introduce the following constraint, where $\mathcal{P}_m \subseteq \mathcal{P}$ denotes the set of signal paths that involve m, and p_m can be arbitrarily selected from $\mathcal{P}_m$:

$$\forall m \in \mathcal{M}\ \forall r \in \mathcal{R}\ \forall p \in \mathcal{P}_m \setminus \{p_m\} : \sum_{\lambda \in \Lambda} b_{m,r}^{p,\lambda} = \sum_{\lambda' \in \Lambda} b_{m,r}^{p_m,\lambda'}. \tag{17}$$

4) Expected Transmission Efficiency: For each $p \in \mathcal{P}$ and for each $m \in \mathcal{M}_p$, we introduce a continuous variables $v_{p,m} \in [-100, 0]$ to represent the expected transmission efficiency in dB at MRR m in path p with the following constraints:

$$\forall p \in \mathcal{P}\ \forall m \in \mathcal{DM}_p : v_{p,m} = \sum_{r \in \mathcal{R}} \sum_{\lambda \in \Lambda} (b_{m,r}^{p,\lambda} \mathbb{E}[H_d(R, \lambda)]_{\text{dB}}) \tag{18}$$

$$\forall p \in \mathcal{P}\ \forall m \in \mathcal{TM}_p : v_{p,m} = \sum_{r \in \mathcal{R}} \sum_{\lambda \in \Lambda} (b_{m,r}^{p,\lambda} \mathbb{E}[H_t(R, \lambda)]_{\text{dB}}). \tag{19}$$

We further introduce a continuous variable $v_w \in [-100, 0]$ to represent the worst-case expected transmission efficiency with the following constraints:

$$\forall p \in \mathcal{P} : v_w \leq c_p \cdot (1 - cl)_{\text{dB}} + \sum_{m \in \mathcal{M}_p} v_{p,m} \qquad (20)$$

where $(1 - cl)_{\text{dB}} = 10 \log_{10}(1 - cl)$ is the crossing loss in dB.

Thus, the optimization model can be formulated as follows:

$$\textbf{Maximize:} \quad v_w \ ; \ \textbf{Subject to:} \quad (15) - (20).$$

Notably, we use the existing topology, which defines all signal paths and the positions of drop and through MRRs. Our model automatically assigns optimal wavelengths to each path to maximize v_w, preventing signal paths from using the same wavelength when overlapping in waveguide sections.

B. WRONoC-Tailored Simulated Annealing

The NP-hard complexity of integer linear programming makes it difficult to find the optimal solution for large problems. Therefore, we develop a simulated annealing algorithm to approximate the optimal solution in an efficient manner. We denote the number of initial solutions as N_i, the initial temperature as T_i, the current temperature as T_c, and the cooling rate as α. Additionally, for each MRR, we define $\Lambda_{d,r}$ and $\Lambda_{t,r}$ as the sets of resonant and non-resonant wavelengths satisfying $H_d(r, \lambda) \geq \theta_1$ and $H_t(r, \lambda) \geq \theta_2$, where r is the nominal radius of the MRR, and θ_1 and θ_2 are the thresholds of the acceptable signal transmission efficiency at the drop or through port of the MRR. Optimization stops if either: (1) The maximum iteration number N_m is reached, or (2) $\mathbb{E}[H_w]$ does not change in N_s consecutive iterations. After the optimization, we calculate the $\mathbb{E}[H_w]$ in dB, denoted as $\mathbb{E}[H_w]_{\text{dB}}$, based on the following model: $\mathbb{E}[H_w]_{\text{dB}} = 10 \log_{10} \mathbb{E}[H_w]$.

Each initial solution is prepared as follows:

- $\forall m \in \mathcal{M}$, we assign a random radius, denoted as r_m.
- $\forall p \in \mathcal{P}$, we create a set Λ_p of candidate wavelengths with $\Lambda_p = (\bigcap_{m \in \mathcal{DM}_p} \Lambda_{d,r_m}) \cap (\bigcap_{m \in \mathcal{TM}_p} \Lambda_{t,r_m})$. If $\Lambda_p = \emptyset$, it implies that there is no available wavelength that can satisfy the efficiency threshold. In this case, we let $\Lambda_p = \bigcap_{m \in \mathcal{DM}_p} \Lambda_{d,r_m}$. We iterate Λ_p and calculate the expected transmission efficiency, i.e., $\mathbb{E}[H_p]$ on each $\lambda \in \Lambda_p$, and select the one that gives the highest transmission efficiency as the initial wavelength of p, denoted as λ_p.

In the k^{th} solution of our simulated annealing algorithm, we first identify the path with the worst expected transmission efficiency, denoted as p_w^k and $\mathbb{E}[H_{p_w}^k]$, respectively. Then, among $\mathcal{M}_{p_w^k}$, we identify the MRR with the worst expected transmission efficiency, denoted as m_w^k. Next:

- A new radius for m_w^k is chosen randomly from the radii options, denoted as $r'_{m_w^k}$.
- $\forall p^k \in \mathcal{P}_{m_w^k}$, we re-calculate their expected transmission efficiency and select a new wavelength with the same method for preparing the initial solution, denoted as λ'_{p^k}.
- Based on the new calculations, we identify the path with currently worst transmission efficiency, denoted as ${p_w^k}'$ and $\mathbb{E}[H_{p_w}^k]'$, respectively.
- If $\mathbb{E}[H_{p_w}^k]' > \mathbb{E}[H_{p_w}^k]$, we accept the new solution, i.e. we update the radius of m_w^k as $r'_{m_w^k}$, the wavelength of p_w^k as $\lambda'_{p_w^k}$, and the expected transmission efficiency $\mathbb{E}[H_{p_w}^k]$ as $\mathbb{E}[H_{p_w}^k]'$. If $\mathbb{E}[H_{p_w}^k]' \leq \mathbb{E}[H_{p_w}^k]$, the new solution is accepted with a probability $P_k = \mathbb{E}[H_{p_w}^k]' \cdot T_c$.
- After each iteration, the highest $\mathbb{E}[H_{p_w}^k]$ of all solutions is denoted as $\mathbb{E}[H_w]$. N_s is incremented by one if $\mathbb{E}[H_w]$ is unchanged compared to the previous iteration; otherwise, N_s is reset to zero. In the early stages, the algorithm searches all solutions to ensure that no potential optimization opportunities are missed. After several iterations, some solutions with low worst-case expected transmission efficiency are discarded.
- The current temperature T_c decreases based on the cooling schedule α.

VI. Optimization Results

A. Inputs and Parameter Settings

We implement our approach in C++ and run the optimization on an Apple M1 Pro 10-core CPU. We consider two well-known WRONoC full-connectivity topologies: Light [6], Snake [15] as input, where each node has a path for signal transmission with other nodes. Different than Light, a node in Snake also has a signal transmission path to itself. Additionally, we consider two application-specific topologies as input: CustomTopo [3] and FAST [21], where nodes without data transmission requirements are not interconnected. We consider the input MRR radii range as $[5\,\mu\text{m}, 30\,\mu\text{m}]$, and the wavelengths range as $[1500\,\text{nm}, 1600\,\text{nm}]$. We also tested different relative errors with $\eta = 0.01\%$, 0.05%, and 0.1%.

Since the simulated annealing (SA) algorithm is a stochastic optimization method, its solution quality is inherently uncertain. Therefore, we first solved the ILP model, which guarantees global optimal solutions, as a benchmark for evaluating the SA algorithm in small topologies. For the ILP model, we prepare 101 MRR radii options ($\mathcal{R}$) within the input range, with $0.25\,\mu\text{m}$ steps, and 126 wavelength options (Λ) within the input range, with $0.8\,\text{nm}$ steps. A wavelength $\lambda \in \Lambda$ is considered on-resonance with an MRR of radius $r \in \mathcal{R}$ if $H_d(r, \lambda) \geq 0.995$. We exclude MRR radii that cannot resonate with any wavelengths from $\mathcal{R}$ and signal wavelengths off-resonance with every MRR radius from Λ, resulting in 38 valid radii options and 33 valid wavelength options. Due to the high complexity of the ILP problem, we only consider 4-node topologies and solve the model with Gurobi [22]. For the SA algorithm, we prepare 1001 MRR radii options in the input radii range with a $0.025\,\mu\text{m}$ step between each option and 1001 wavelengths options in the input wavelength range with a $0.1\,\text{nm}$ step between each option. It took approximately 6 hours to prepare the expected values for the $1001 \cdot 1001 = 1002001$ radius-wavelength combinations under four relative errors. These are one-time calculations and are not repeated in subsequent steps. From the 50^{th} to the 140^{th} iteration in the SA experiment, five poor-quality solutions are discarded every ten iterations. Thus, after the

TABLE I
INPUT PARAMETERS

Parameter	Description	Value
k	Cross-coupling coefficient	0.4 [10]
t	Self-coupling coefficient	$\sqrt{0.84}$
cl	Crossing loss coefficient	0.009168 [6]
T_i	Initial temperature	1
N_i	Number of initial solutions	100
α	Cooling rate	0.99
θ_1	Selection criterion for $\Lambda_{\mathrm{d},r}$	0.85
θ_2	Selection criterion for $\Lambda_{\mathrm{t},r'}$	0.85
N_s	Consecutive number of no improvement	1000
N_m	Maximum number of iterations	3000

140^{th} iteration, only five high-quality solutions continue to be optimized until the algorithm reaches one of its stopping conditions. Other input parameters are shown in Table I.

Two experimental designs are set up for the comparative analysis. The first design, referred to as the **nominal design**, maximizes the worst-case signal transmission efficiency without considering process variation (i.e., $\eta = 0$). The second design, referred to as **variation-aware design**, maximizes the worst-case expected signal transmission efficiency considering process variation (i.e., $\eta \neq 0$). Then, using the optimally selected MRR radii and signal wavelengths of the nominal design, the expected transmission efficiencies of the signals through the MRRs affected by a given relative error are calculated, and their minimum value is determined and denoted as Nom. $\mathbb{E}$. This is then compared to the worst-case expected signal transmission efficiency under the variation-aware design with the same relative error, termed Var. $\mathbb{E}$. To ensure experimental fairness, both designs use the same initial solutions for the same topology scale under all relative errors in the SA algorithm. The results are shown in Table II and III.

B. Results Discussion

1) Designs: The worst-case expected transmission efficiencies under variation-aware designs are consistently greater than those under nominal designs with the same relative error. As shown in "Diff.", the optimization effect of the variation-aware design becomes more pronounced as the relative error increases. Even with a relative error of only 0.01%, the variation-aware design improves the worst-case expected transmission efficiencies in the range of $0.81\,\mathrm{dB}$ to $1.78\,\mathrm{dB}$. As the error increases, the variation-aware design improves the expected transmission efficiency by up to $7.51\,\mathrm{dB}$. This indicates that our strategy can effectively maximize the expected transmission efficiency.

2) Algorithms: The differences between the optimized expected transmission efficiencies of the SA algorithm and the ILP model are insignificant for the same topological scale and relative error. In all cases, the SA algorithm provides even superior optimized expected transmission efficiencies due to the wider range of optimization options available. Thus, we can consider the results of our WRONoC-tailored SA algorithm in larger-scale topologies under process variation to be reliable.

3) Optimization Time: The optimization time under nominal and variation-aware design is similar for the same network topology and scale. However, the ILP method generally takes much longer than the SA algorithm, except for the Light 4×3 topology with few MRRs. For 8-node and 16-node topologies, the ILP model fails to find the optimal solution within 10 hours. The SA algorithm can find high-quality solutions in 4-node topologies in two minutes, 8-node topologies in ten minutes, and 16-node topologies in only 36 minutes. This confirms that the SA algorithm can efficiently find a design resistant to process variation.

TABLE II
INTEGER LINEAR PROGRAMMING RESULTS

T	S	$\lvert\mathcal{M}\rvert$	η	Nom. $\mathbb{E}$	Var. $\mathbb{E}$	**Diff.**	Opt. T.
Light	4×3	4	0.01%	−1.86 dB	−0.43 dB	**1.43** dB	5.33 s
			0.05%	−6.56 dB	−2.12 dB	**4.44** dB	5.74 s
			0.1%	−9.56 dB	−3.89 dB	**5.67** dB	3.65 s
Snake	4×4	12	0.01%	−2.44 dB	−0.72 dB	**1.72** dB	7247.73 s
			0.05%	−7.75 dB	−2.61 dB	**5.14** dB	1095.46 s
			0.1%	−10.62 dB	−4.62 dB	**6** dB	972.36 s

TABLE III
SIMULATED ANNEALING RESULTS

T	S	$\lvert\mathcal{M}\rvert$	η	Nom. $\mathbb{E}$	Var.$\mathbb{E}$	**Diff.**	Opt. T.
Light	4×3	4	0.01%	−2.15 dB	−0.4 dB	**1.75** dB	31.69 s
			0.05%	−7.14 dB	−1.93 dB	**5.21** dB	41.73 s
			0.1%	−9.96 dB	−3.62 dB	**6.34** dB	49.25 s
	8×7	24	0.01%	−2.82 dB	−1.22 dB	**1.6** dB	232.59 s
			0.05%	−8.39 dB	−3.21 dB	**5.18** dB	161.28 s
			0.1%	−11.99 dB	−5.26 dB	**6.73** dB	245.52 s
	16×15	112	0.01%	−4.91 dB	−4.1 dB	**0.81** dB	1029.26 s
			0.05%	−12.69 dB	−7.79 dB	**4.9** dB	1123.12 s
			0.1%	−17.44 dB	−11.14 dB	**6.3** dB	1125.09 s
Snake	4×4	16	0.01%	−2.47 dB	−0.69 dB	**1.78** dB	101.52 s
			0.05%	−7.7 dB	−2.25 dB	**5.45** dB	111.77 s
			0.1%	−10.85 dB	−4.05 dB	**6.8** dB	116.67 s
	8×8	56	0.01%	−3.61 dB	−2.18 dB	**1.43** dB	543.69 s
			0.05%	−10.14 dB	−5.34 dB	**4.8** dB	523.26 s
			0.1%	−14.61 dB	−8.55 dB	**6.06** dB	510 s
	16×16	240	0.01%	−8.8 dB	−7.9 dB	**0.9** dB	2139.84 s
			0.05%	−20.35 dB	−16.37 dB	**3.98** dB	1845.17 s
			0.1%	−26.85 dB	−20.51 dB	**6.34** dB	1845.1 s
Custom-Topo	44	48	0.01%	−2.87 dB	−1.28 dB	**1.59** dB	315.81 s
			0.05%	−8.74 dB	−3.63 dB	**5.11** dB	305.56 s
			0.1%	−12.66 dB	−6.14 dB	**6.52** dB	310.63 s
FAST	44	36	0.01%	−2.69 dB	−1.14 dB	**1.54** dB	205.54 s
			0.05%	−8.36 dB	−3.08 dB	**5.28** dB	249.62 s
			0.1%	−12.38 dB	−4.87 dB	**7.51** dB	246.05 s

T: the topology type; S: the number of the signal paths; $|\mathcal{M}|$: the number of MRRs; η: the relative error; Nom. $\mathbb{E}$: the worst-case expected transmission efficiency optimized under nominal design; Var. $\mathbb{E}$: the worst-case expected transmission efficiency optimized under variation-aware design; Diff.: the difference of Var. $\mathbb{E}$ and Nom. $\mathbb{E}$; Opt. T.: the optimization time.

VII. CONCLUSION

In this work, we propose a stochastic optimization strategy to maximize the expected signal transmission efficiency of a given WRONoC topology under process variation. Distinct from existing solutions, we accurately evaluate the effects of variation using probability theory. We then construct an ILP model and validate it through a suite of small-scale topologies. To expand the scale of optimization and shorten the optimization time, we developed a WRONoC-tailored simulated annealing algorithm. Compared to the optimization results without considering process variation, our approach can improve the worst-case expected signal transmission efficiency by up to $7.51\,\mathrm{dB}$ and thus provide more robustness of WRONoC under process variation.

ACKNOWLEDGMENT

This work is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 496766278.

REFERENCES

[1] Scott Beamer, Chen Sun, Yong-Jin Kwon, Ajay Joshi, Christopher Batten, Vladimir Stojanović, and Krste Asanović. Re-architecting dram memory systems with monolithically integrated silicon photonics. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, page 129–140, New York, NY, USA, 2010. Association for Computing Machinery.

[2] Luca Ramini, Davide Bertozzi, and Luca P. Carloni. Engineering a bandwidth-scalable optical layer for a 3d multi-core processor with awareness of layout constraints. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 185–192, 2012.

[3] Mengchu Li, Tsun-Ming Tseng, Davide Bertozzi, Mahdi Tala, and Ulf Schlichtmann. Customtopo: A topology generation method for application-specific wavelength-routed optical nocs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2018.

[4] Zhidan Zheng, Liaoyuan Cheng, Kanta Arisawa, Qingyu Li, Alexandre Truppel, Shigeru Yamashita, Tsun-Ming Tseng, and Ulf Schlichtmann. Multi-resonance mesh-based wavelength-routed optical networks-on-chip. In *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[5] Tsun-Ming Tseng, Alexandre Truppel, Mengchu Li, Mahdi Nikdast, and Ulf Schlichtmann. Wavelength-routed optical nocs: Design and eda — state of the art and future directions: Invited paper. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6, 2019.

[6] Zhidan Zheng, Mengchu Li, Tsun-Ming Tseng, and Ulf Schlichtmann. Light: A scalable and efficient wavelength-routed optical networks-on-chip topology. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 568–573, 2021.

[7] Wim Bogaerts, Peter De Heyn, Thomas Van Vaerenbergh, Katrien De Vos, Shankar Kumar Selvaraja, Tom Claes, Pieter Dumon, Peter Bienstman, Dries Van Thourhout, and Roel Baets. Silicon microring resonators. *Laser & Photonics Reviews*, 6(1):47–73, 2012.

[8] Liaoyuan Cheng, Mengchu Li, Tsun-Ming Tseng, and Ulf Schlichtmann. Minimizing worst-case data transmission cycles in wavelength-routed optical noc through bandwidth allocation. In *2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2024.

[9] Asif Mirza, Febin Sunny, Peter Walsh, Karim Hassan, Sudeep Pasricha, and Mahdi Nikdast. Silicon photonic microring resonators: A comprehensive design-space exploration and optimization under fabrication-process variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3359–3372, 2022.

[10] Asif Mirza, Febin Sunny, Sudeep Pasricha, and Mahdi Nikdast. Silicon photonic microring resonators: Design optimization under fabrication non-uniformity. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 484–489, 2020.

[11] Tsui-Wei Weng, Daniele Melati, Andrea Melloni, and Luca Daniel. Stochastic simulation and robust design optimization of integrated photonic filters. *Nanophotonics*, 6(1):299–308, 2017.

[12] Yu-Kai Chuang, Yong Zhong, Yi-Hao Cheng, Bo-Yi Yu, Shao-Yun Fang, Bing Li, and Ulf Schlichtmann. Robustonoc: Fault-tolerant optical networks-on-chip with path backup and signal reflection. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 67–72, 2021.

[13] Zhidan Zheng, Mengchu Li, Tsun-Ming Tseng, and Ulf Schlichtmann. Lightr: A fault-tolerant wavelength-routed optical networks-on-chip topology. *MDPI Applied Sciences*, 13(15):8871, Jul 2023.

[14] Lukas Chrostowski and Michael Hochberg. *Silicon Photonics Design: From Devices to Systems*. Cambridge University Press, 2015.

[15] Luca Ramini, Paolo Grani, Sandro Bartolini, and Davide Bertozzi. Contrasting wavelength-routed optical noc topologies for power-efficient 3d-stacked multicore processors using physical-layer analysis. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1589–1594, 2013.

[16] Yufei Xing, Jiaxing Dong, Umar Khan, and Wim Bogaerts. Capturing the effects of spatial process variations in silicon photonic circuits. *ACS Photonics*, 10(4):928–944, 2022.

[17] Zhengxing Zhang, Sally I. El-Henawy, Carlos Ríos, and Duane S. Boning. Inference of process variations in silicon photonics from characterization measurements. In *2022 Conference on Lasers and Electro-Optics (CLEO)*, pages 1–2, 2022.

[18] Lukas Chrostowski, Xu Wang, Jonas Flueckiger, Yichen Wu, Yun Wang, and Sahba Talebi Fard. Impact of fabrication non-uniformity on chip-scale silicon photonic integrated circuits. In *Optical Fiber Communication Conference*, pages 1–3, 2014.

[19] Saeed Ghahramani. *Fundamentals of Probability with Stochastic Processes*. Pearson/Prentice Hall, 2005.

[20] Bin Wang, Wenzhong Shi, and Zelang Miao. Confidence analysis of standard deviational ellipse and its extension into higher dimensional euclidean space. *PloS one*, 10(3):e0118537, 2015.

[21] Moyuan Xiao, Tsun-Ming Tseng, and Ulf Schlichtmann. Fast: A fast automatic sweeping topology customization method for application-specific wavelength-routed optical nocs. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1651–1656, 2021.

[22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

A Cross-model Fusion-aware Framework for Optimizing (gather-matmul-scatter)$_s$ Workload

Yaoxiu Lian[1], Zhihong Gou[1], Yibo Han[1], Zhongming Yu[5],
Jiaming Xu[1], Sheng Yuan[2], Zhilin Pei[2], Xingcheng Zhang[2], Ningyi Xu[1], Guohao Dai[1,3,4†]
[1]Shanghai Jiao Tong University
[2]Shanghai Artificial Intelligence Laboratory Independent Researcher
[3]Infinigence-AI [4]SII [5]Independent Researcher
[†]Corresponding author: daiguohao@sjtu.edu.cn

***Abstract*—Modern deep learning models, such as Relation Graph Convolutional Network (RGCN), Sparse Convolutional Networks (SpConv), and Mixture of Experts Networks (MoE), are significantly dependent on the (gather-matmul-scatter)$_s$ (abbreviated as (g-mm-s)$_s$) workload as their fundamental computational pattern. While existing works have made optimization attempts, several critical challenges remain unsolved, including domain-specific optimization migration, time-consuming exploration, and inefficient dataflow with dynamic inputs.**

To address these challenges, we introduce Efficient-GMS, a comprehensive framework that enhances (g-mm-s)$_s$ workload across diverse input scenarios. Our framework introduces (1) A Fusion-aware framework enabling cross-model optimization migration. We propose a comprehensive dataflow analysis that identifies shared computational patterns across models, enabling the development of four optimized dataflow patterns with vertical and horizontal fusion strategies. (2) Performance model-guided configuration space reduction. We develop a performance model to predict the relative execution efficiency across configurations, thereby reducing the search space and minimizing search time while ensuring optimal configuration selection. (3) Adaptive dataflow selection mechanism. We implement a lightweight heuristic model that dynamically selects optimal dataflow patterns based on the characteristics of the input and the hardware. Experimental results demonstrate that Efficient-GMS achieves significant performance gains, delivering an average end-to-end speedup of 1.46× in RGCN model, 1.32× in SpConv-based model, and 1.15× in MoE model compared to state-of-the-art methods.

***Index Terms*—gather, scatter, matmul**

I. Introduction

The advancement of deep learning models has enabled us to address complex relationships and patterns in irregular data. Examples include Relational Graph Convolutional Networks (RGCN [19], [29], [31]), Sparse Convolutional Networks (SpConv [12], [23]–[25]), and Mixture of Experts (MoE [7], [15]). These models, which support a wide range of applications from social network analysis [10], [13], [16], [26] to natural language processing [5], [14], [21], share a core computational pattern: (gather-matmul-scatter)$_s$ (abbreviated as (g-mm-s)$_s$), where the subscript "s" indicates the plural form.

The (g-mm-s)$_s$ workload represents a unified abstraction of core computations across these models. It consists of three key operations: (1) "gather", which collects features from relevant sources (e.g., neighboring nodes in RGCN, non-zero elements in SpConv, or expert components in MoE); (2)"matmul", which transforms these features through learnable weight matrices; and (3) "scatter", which distributes the updated results to appropriate destinations. This pattern, though conceptually straightforward, presents significant implementation challenges due to its varied forms across different models.

Although (g-mm-s)$_s$ captures the core computations of these models, traditional implementation methods often struggle to efficiently execute (g-mm-s)$_s$ in models with irregular structures, such as graphs with multiple relationships. For example, in RGCN, when PyG [8] executes the computation flow, it processes each set of gather, matmul, and scatter operations sequentially. This approach is inefficient due to indirect memory access and the challenges of fully utilizing parallel processing resources. Some studies [6], [22], [32], focus on optimizing multiple matmuls within the workload; however, they neglect the inter-operator relationships across the entire workload. Consequently, the performance improvements of the optimized models are limited.

Based on our analysis, we identify three fundamental challenges in optimizing (g-mm-s)$_s$:

- **Domain-specific optimization hindering cross-model migration**. Current optimization techniques are often designed for specific models, and applying these strategies to different models can result in significant performance differences. In some cases, directly using these techniques can cause severe performance drops—for example, applying the SpConv strategy to an RGCN model on the BGS dataset results in a 9.61× decrease in performance.
- **Time-consuming exploration within large configuration space**. The configuration space for operators within (g-mm-s)$_s$ workload is extensive, often encompassing thousands to even tens of thousands of possible configurations. Traditional approaches demand significant search time (up to 3000 seconds) to identify the optimal configuration. Furthermore, directly specifying configurations based on heuristic rule may result in suboptimal performance outcomes.
- **Inefficient static dataflow with dynamic inputs**. Traditional static optimization methods struggle with dynamic inputs, leading to inefficiencies with varying data characteristics. Performance varies widely across dataflow patterns, yet current systems lack the ability to dynamically select the most efficient pattern. Our experiments show that poor dataflow selection can cause up to a 14.77× performance degradation under varying workloads.

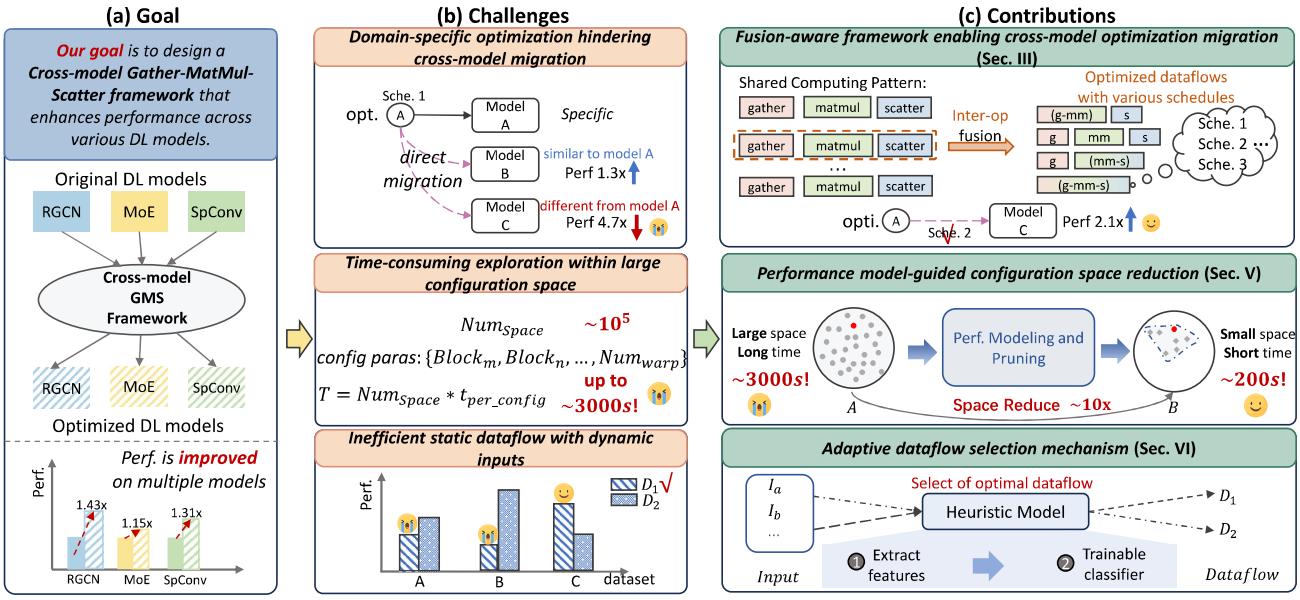


Fig. 1. Overview of Efficient-GMS, cross-model gather-matmul-scatter framework that enhances performance across various DL models.

To address these challenges, we present an efficient (g-mm-s)$_s$ framework with the following key contributions:

- **Fusion-aware framework enabling cross-model optimization migration**. We develop a comprehensive optimization framework that identifies fundamental computational workload shared by different models. By analyzing the commonalities of workloads, we present four optimized dataflow patterns that combine fusion strategies and implement optimal scheduling strategies for different inputs. This approach achieves stable performance improvements from 1.43× to 1.86× in different models.
- **Performance model-guided configuration space reduction**. We develop a performance model based on parallel efficiency and the compute-to-memory access ratio that predicts the relative execution efficiency for each configuration, significantly reduces the search space by 60% to 90% while ensuring optimal configuration selection, and achieves a 5.26× reduction in average search time.
- **Adaptive dataflow selection mechanism**. We propose a lightweight heuristic model that dynamically selects optimal dataflow patterns based on input and hardware characteristics. This approach delivers 1.67× to 6.03× performance improvement compared to static implementations, effectively addressing the dynamic input challenge.

In conclusion, we introduce Efficient-GMS, a comprehensive framework designed to enhances (g-mm-s)$_s$ workload across models and tasks. Experimental results demonstrate that our approach achieves an average speedup of **1.73**× and **1.52**× on the (g-mm-s)$_s$ kernel for RGCN and SpConv-based model, respectively. Furthermore, the end-to-end acceleration effects are significant, with average speedups of **1.43**×, **1.31**×, and **1.15**× on RGCN, SpConv-based model, and MoE tasks, respectively.

II. Background

Relational Graph Convolutional Networks (RGCN), Sparse Convolutional Networks (SpConv), and Mixture of Expert Models (MoE) have emerged as sophisticated computational frameworks for tackling complex tasks in diverse domains. RGCNs excel at capturing relationships in graph-structured data [17], [18], SpConvs efficiently process sparse data like 3D point clouds [1], [27], and MoEs leverage specialized subnetworks to handle different input patterns [14], [30]. While seemingly disparate, these models share an underlying computational theme: they all adaptively aggregate information from a selective subset of the input. In this section, we introduce these three models, focus on their dataflow and demonstrating their adherence to the (g-mm-s)$_s$ workload.

A. Relational Graph Convolutional Network (RGCN)

RGCN [19] extends the gather-matmul-scatter pattern to handle multi-relational graph data. During gathering, it collects information from neighboring nodes based on the graph's connectivity, considering different edge types as distinct relations. The matmul stage applies relation-specific weight matrices to transform the gathered information, enabling the network to learn relation-dependent feature representations. The scatter stage combines transformed information across all relations through summation, producing updated node representations that capture the graph's relational structure.

B. Sparse Convolutional Network

Sparse-based networks [12] implement an efficient gather-matmul-scatter pattern for sparse data like 3D point clouds. The gather stage collects features from neighboring points within a geometrically defined receptive field (e.g., sphere or cube). The matmul stage applies learned filters to gathered features, performing convolution in sparse space while computing only active points. The scatter stage aggregates these convolved features through summation, updating the

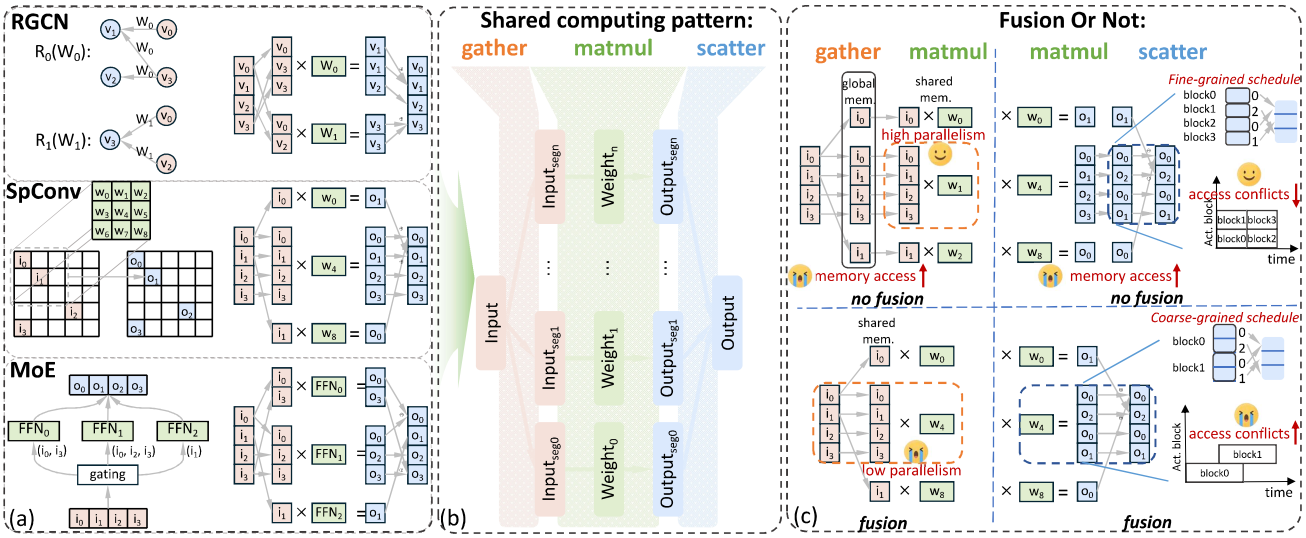


Fig. 2. (a). We demonstrate corresponding dataflows in RGCN, SpConv and MoE model. (b). gather-matmul-scatter is the shared computing pattern among three models. (c). Analyzing the advantages and disadvantages of fusion between operators.

central point's representation while maintaining computational efficiency through sparsity exploitation.

C. Mixture of Expert Model

MoE architectures [7], [15] adapt the gather-matmul-scatter pattern to ensemble learning. The gather stage employs a gating network that routes inputs to appropriate expert subsets. In the matmul stage, selected experts process inputs in parallel, each applying specialized computations. The scatter stage combines expert outputs through gating-weighted aggregation, producing final predictions that leverage multiple specialized networks. This architecture enables efficient scaling by activating only relevant experts for each input.

III. Fusion-aware framework enabling cross-model optimization migration

A. Motivation

The optimization techniques developed for various deep learning models demonstrate notable potential for cross-model applicability. For instance, the gather-matmul-scatter fusion used in SpConv can be adapted to RGCN's feature aggregation, while RGCN's segmentation-based computation strategy can benefit expert computation in Mixture of Experts (MoE). Experiments demonstrate notable performance gains from such migrations; for example, RGCN's feature aggregation on the IMDB dataset achieves a 4.64× speedup using SpConv's strategy.

B. Challenge

Despite the variety of optimization methods, most implementations remain domain-specific. Direct migration often leads to instability due to differences in model implementation details and data performance characteristics. For example, applying the operator fusion strategy to RGCN on the BGS dataset results in a 9.61× performance drop.

C. Insight

Our analysis reveals that seemingly diverse models share a fundamental (g-mm-s)$_s$ computational workload. This commonality enables the construction of a unified optimization framework, dynamically guiding strategy migration across models based on operator and data characteristics. Experiments validate the framework's efficacy, achieving stable performance improvements between 1.15× and 1.46× across various models.

D. Approach

We propose a **systematic optimization framework for (g-mm-s)$_s$ workload** and thoroughly analyze potential optimization methods in both vertical and horizontal dimensions. **Vertically**, parallel operator scheduling focuses on enhancing hardware utilization by combining small-scale gather, matmul, or scatter operations into larger ones. Inspired by segment-matmul methods [32], parallel scheduling of fused matmul operations significantly boosts performance. **Horizontally**, inter-operator fusion analyzes the interrelationships among gather, matmul, and scatter to refine four fundamental dataflows: $G-M_{seg}-S$, executing operators independently; $(GM)_{seg}-S$, combining gather and matmul; $G-(MS)_{seg}$, merging matmul and scatter; and $(GMS)_{seg}$, fully integrating all three operators. At the same time, the above four dataflows are executed based on guaranteeing parallel operator scheduling, and the subscript in the dataflow represents that the operator organizes operations in the way of the segment. These dataflows provide a versatile optimization framework adaptable to varying workloads and hardware architectures. Fusion can enhance data locality and reduce overhead, though it may increase memory access conflicts, affecting efficiency. By evaluating hardware characteristics, input data, and computational workload, the framework enables the dynamic selection of optimal dataflow, achieving significant performance gains across diverse computational scenarios.

E. Op Fusion Analysis

The fusion of gather and matmul operators involves a trade-off, especially for large-scale computations. Independent operators often outperform fused implementations by leveraging

enhanced parallelism. Separating gather operations enables concurrent processing across thread blocks, decoupling irregular memory access from structured matrix multiplication and maximizing GPU parallelism. This design also allows dynamic resource allocation and avoids bottlenecks seen in fused implementations.

For the fused $(MS)_{seg}$ kernel, performance suffers from atomic write conflicts during scatter operations. While fusion improves data reuse and reduces memory accesses, handling more rows per block increases conflicts, extending execution time. Separating matmul and scatter mitigates these conflicts, offering greater configurability and improved efficiency.

IV. Performance Model-guided Configuration Space Reduction

A. Motivation

The selection of configuration parameters in the dataflow directly determines computational efficiency, making configuration space optimization crucial for performance enhancement. In the (g-mm-s)$_s$ workload, gather operations involve non-contiguous memory access, matmul requires effective resource scheduling, and scatter entails complex data distribution. Parameters such as block size, affecting data locality, and thread allocation, influencing parallelism, are critical. Suboptimal configurations can degrade memory access efficiency and under-utilize computational resources.

B. Challenge

The optimization of configuration space is challenging due to its high dimensionality and interdependent parameters. In the (g-mm-s)$_s$ pattern, parameters like tile sizes ($Block_M$, $Block_N$, $Block_K$) influence data partitioning, while thread counts (Num_{warp}) control parallelism. These parameters are interrelated: larger tiles improve data reuse but limit parallelism, while more threads increase parallelism but risk resource contention. Input variations, such as sparsity and data scale, further complicate configuration needs, expanding the space to tens of thousands of possibilities. Exhaustive searches, taking thousands of seconds, are thus impractical.

C. Insight

Our key insight is that **configuration parameters and performance indicators can be quantitatively characterized,** forming a foundation for efficient optimization. Configuration parameters primarily impact performance through parallelism and memory access efficiency: tile size influences data locality and cache usage, while thread allocation affects resource utilization. These relationships are predictable and quantifiable, enabling theoretical modeling to evaluate configuration effects without execution. This facilitates the rapid identification of optimal configurations.

D. Approach

To address these challenges, we propose a novel approach for **optimization configuration through coarse-grained performance modeling**. As shown in Fig. 3(b), The proposed model evaluates the relative execution efficiency under various configurations while keeping the computational overhead

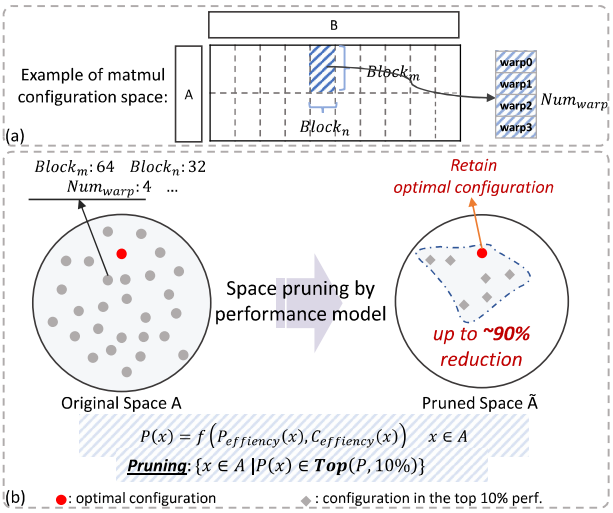


Fig. 3. (a).Example configuration space for matmul, which explains the meaning of the three configurable parameters (b). Performance-guided configuration space reduction.

minimal by analyzing the parallel efficiency and compute-to-memory efficiency of the operators under the current input. Subsequently, configurations within space $\mathcal{A}$ are ranked and pruned to generate a reduced configuration space $\mathcal{B}$, comprising approximately 10% to 40% of the original space. The system then executes configurations exclusively within space $\mathcal{B}$ on the kernel, selecting the configuration with optimal average execution time, thus substantially reducing the optimization process duration.

Taking the $(GM)_{seg}$ operator as an example, its performance can be modeled as follows:

$$P = (1 - \text{Pall}_{\text{efficiency}}) \cdot (1 - \text{Comp}_{\text{efficiency}})$$
$$\text{Comp}_{\text{efficiency}} = T_{\text{comp}} / (T_{\text{load}} + T_{\text{store}} + T_{\text{comp}})$$
$$\text{Pall}_{\text{efficiency}} = N_{\text{blk}} / \left(\lceil N_{\text{blk}} / N_{\text{max_act_blk}} \rceil \cdot N_{\text{max_act_blk}}\right)$$
$$N_{\text{blk/sm}} = \min\left(\text{shm}_{\text{max}} / \text{shm}_{\text{required}}, N_{\text{thread/sm}} / (\text{warp} \cdot 32)\right)$$
$$N_{\text{max_act_blk}} = N_{\text{blk/sm}} \cdot N_{\text{sm}}$$
$$T_{\text{comp}} = 2MNK / \text{PeakFLOPS}$$
$$T_{\text{store}} = (\text{st_bytes/sm} \cdot N_{\text{max_act_blk}} \cdot \lceil N_{\text{blk}} / N_{\text{max_act_blk}} \rceil) / \text{L2_bw}$$
$$T_{\text{load}} = T_{\text{load_A}} + T_{\text{load_B}}$$
$$T_{\text{load}_{A/B}} = \frac{\text{bytes}_{A/B}}{(1 - \text{L2_hit}_{A/B}) \cdot \text{dram_bw} + \text{L2_bw} \cdot \text{L2_hit}_{A/B}}$$

The modeling process for other operators is similar to that of $(GM)_{seg}$. Additionally, we utilize a caching mechanism to associate optimal configurations with their corresponding data characteristics. This enables direct retrieval of optimal configurations when encountering similar data characteristics, eliminating the need for redundant search processes.

V. Adaptive Dataflow Selection Mechanism

A. Motivation

Dynamically selecting the optimal dataflow pattern is essential to optimize the performance of the (g-mm-s)$_s$ framework. Experimental results demonstrate that no single dataflow pattern consistently achieves optimal performance across diverse input characteristics. This performance variance is primarily driven by the dynamic changes in input data attributes, such as data scale, sparsity, and access patterns,

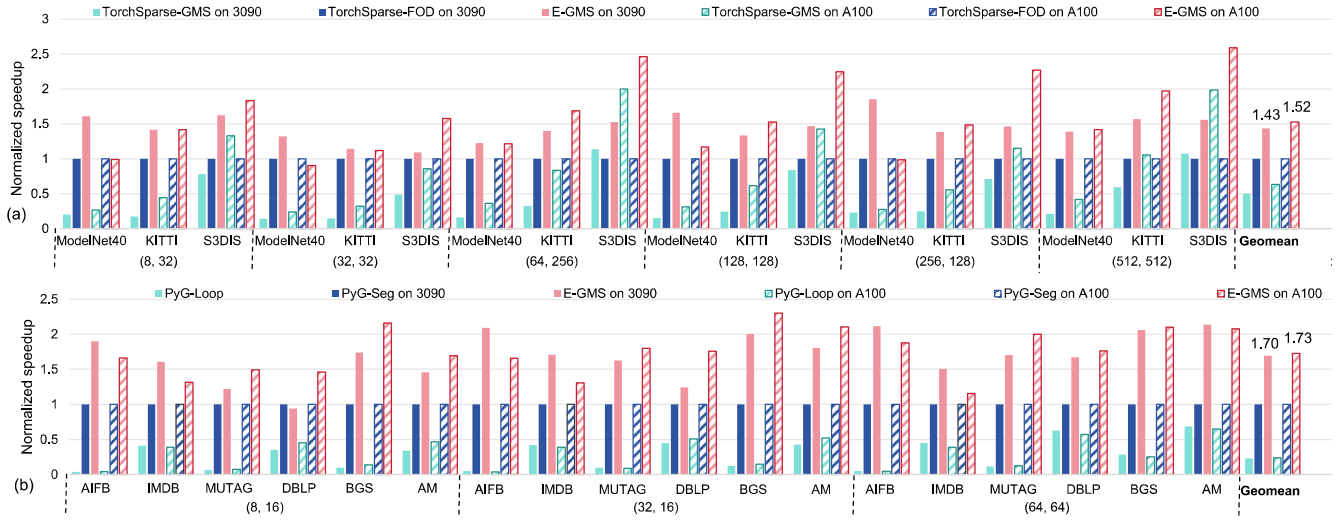


Fig. 4. Kernel performance. (a) kernel performance comparison between E-GMS and Torchsparse on the point cloud datasets. (b) kernel performance comparison between E-GMS and PyG on the heterogeneous graphs.

all of which significantly affect the execution efficiency of dataflow. Consequently, it is crucial to design a mechanism that dynamically selects the most suitable dataflow based on the specific input characteristics.

B. Challenge

Traditional static optimization strategies are not efficient in dynamic scenarios. Existing methods typically rely on predefined heuristic rules or fixed optimization approaches, which make it difficult to adjust the optimization strategy in time when facing input data features, leading to significant performance fluctuations. As shown in Fig. 8, selecting a static dataflow pattern can result in performance degradation of up to 14.77×.

C. Insight

A critical insight reveals **that dataflows exhibit different relative performances on different data**. Our analysis reveals two critical characteristics of dataflow performance: (1) the $(GM)_{seg} - S$ and $(GMS)_{seg}$ dataflows outperform other patterns in most cases, collectively accounting for 96.50% of optimal cases (52.50% and 44.00%, respectively); and (2) while $G - M_{seg} - S$ or $G - (MS)_{seg}$ may achieve optimal performance in rare scenarios (2.75% and 0.75% of cases, respectively), their advantages are marginal, with a maximum performance gap of 1.06×. These findings simplify dataflow selection by focusing on $(GM)_{seg} - S$ and $(GMS)_{seg}$, ensuring effective performance optimization while reducing selection complexity.

D. Approach

To address dynamic dataflow selection in the $(\text{g-mm-s})_s$ workload, we propose an **adaptive machine learning framework** leveraging **Singular Value Decomposition (SVD)** for dimensionality reduction and an XGBoost classifier for dataflow selection. Simple features, such as input-output dimensions, undergo standardization, while complex mapping features are processed using SVD to retain essential features. The framework merges these features into a unified vector, preserving interpretability and core mapping information. The best data flow is predicted by a trained XGBoost classifier. Supporting $(GM)_{seg} - S$ and $(GMS)_{seg}$ modes, the framework dynamically selects efficient dataflow, enabling real-time performance optimization.

VI. Evaluation

A. Experimental Setup

Our evaluation platform comprises NVIDIA GeForce RTX 3090 and A100 (80GB) GPUs, testing three key architectures: sparse point cloud models (SparseResNet [4]) with sparse convolution, heterogeneous graph neural networks (RGCN), and Mixture of Experts models (Misture-8x7B). Comparisons include Torchsparse implementations with gather-matmul-scatter and fetch-on-demand dataflows for point cloud models, PyG implementations using for-loop and segment-matmul for RGCN and HuggingFace implementation for MoE models. We utilize standard datasets: KITTI [11], ModelNet40 [28], and S3DIS [2] for point clouds, and AIFB, MUTAG, BGS, AM [20], IMDB [3], DBLP [9] for heterogeneous graphs.

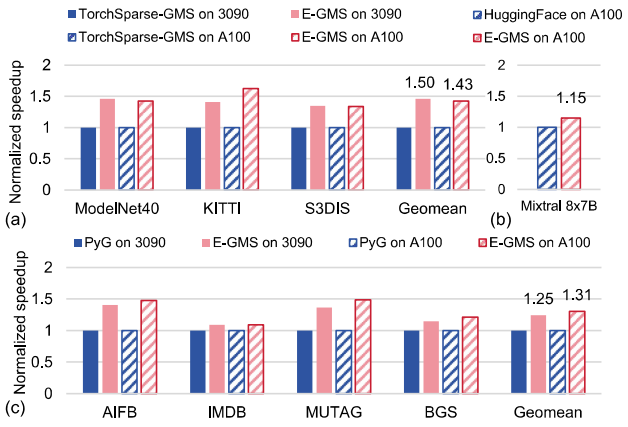


Fig. 5. End to end performance. (a)Performance comparison between E-GMS and Torchsparse on the point cloud datasets. (b)Performance comparison between E-GMS and PyG on the heterogeneous graphs.

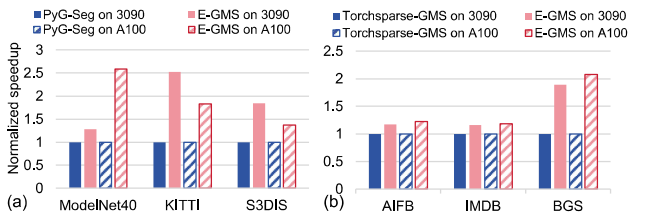


Fig. 6. Performance comparison between E-GMS and direct cross-model scheduling methods. (a) and (b) are the performance on SpConv-based and RGCN networks, respectively

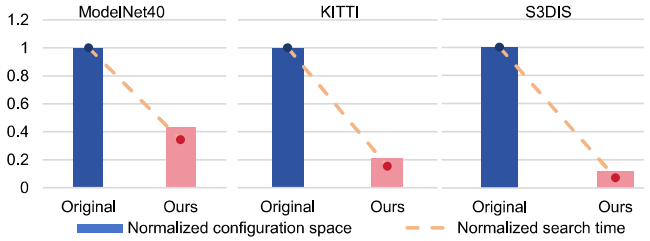


Fig. 7. Configuration space reduce and search time speedup

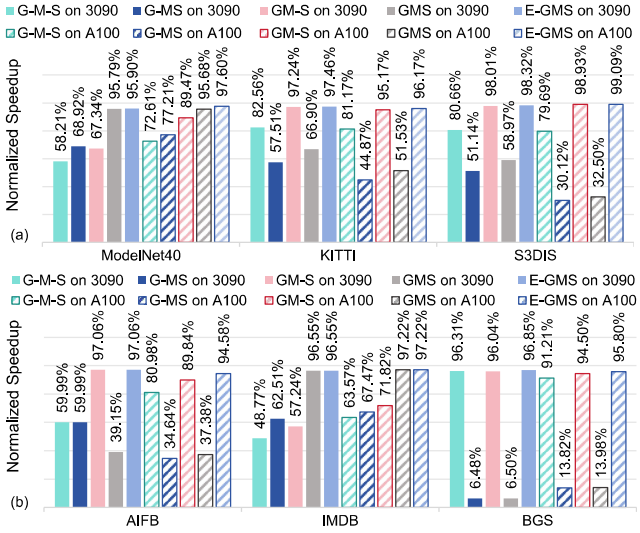


Fig. 8. Heuristic-model performance

B. Overall Performance

End-to-end Performance. We evaluated the end-to-end performance of our proposed Efficient-GMS (E-GMS) method and the state-of-the-art baseline engines across three different scenarios, as shown in Fig. 5. The results show that on the SpConv-based models, E-GMS achieves an average speedup of 1.43× and 1.46× with respect to TorchSparse-FOD on RTX 3090 and A100, respectively. On RGCN, E-GMS achieves an average speedup of 1.25× and 1.32× on RTX 3090 and A100 compared to PyG, respectively. On MoE, E-GMS achieves an average speedup of 1.15× compared to HuggingFace on A100. Notably, E-GMS achieves higher speedups over TorchSparse-FOD and PyG on A100 in two scenarios. This can be attributed to E-GMS's superior GPU utilization, which results in higher performance gains on more powerful GPUs.

Kernel Performance. We further evaluate the kernel performance of GMS operations, showcasing the effectiveness of E-GMS by varying input and output channel sizes for each dataset. We set six and three different channel sizes $(c_{\text{in}}, c_{\text{out}})$ for point cloud and RGCN model, respectively. The results are shown in Fig. 4. E-GMS achieves an average arithmetic speedup of 1.44× and 5.25× over TorchSparse-FOD and TorchSparse-GMS using the segment-matmul method on RTX 3090, respectively. On A100, these values are 1.60× and 2.58×, respectively. In particular, E-GMS achieves a peak speedup of 2.59× over TorchSparse-FOD on large channel size (i.e., (512, 512) on A100), with $(GM)_{seg} - S$ being the optimal dataflow configuration. For the RGCN model, E-GMS achieves an average arithmetic speedup of 14.05× and 1.7× compared to PyG-Loop and PyG-Seg on RTX 3090, respectively. Similarly, on A100, E-GMS provides speedups of 13.78× and 1.76×, showing its efficiency across different hardware platforms.

C. Ablation Study

Fusion-aware framework enabling cross-model optimization migration. As illustrated in Fig. 6, our approach outperforms the direct cross-model method, achieving an average performance improvement of 1.86× on point cloud and a 1.43× speedup on heterogeneous graphs on A100. These results demonstrate efficient scheduling of our method across diverse input types. **Performance model-guided configuration space reduction**. Fig. 7 highlights the efficiency of our performance model. For the GMS_{Seg} operator on the point cloud, the performance model reduces the configuration space by 60%–90% and achieves an average search time reduction of 5.26×, with a notable 13.89× reduction on large-scale datasets. **Heuristic Adaptive Dataflow**. Fig. 8 shows that the heuristic adaptive dataflow shows significant performance benefits, achieving 1.14× to 1.67× speedup on RTX 3090 and 1.03× to 2.23× on A100 for point cloud scenarios. RGCN scenarios show even more substantial improvements, with 1.23× to 6.03× speedup on RTX 3090 and 1.14× to 3.70× on A100, validating the effectiveness of our adaptive approach.

VII. Conclusion

In conclusion, this paper introduced Efficient-GMS, a comprehensive framework designed to enhances $(\text{g-mm-s})_{\text{s}}$ workload across models and tasks. By incorporating fusion-aware optimization, performance model-guided configuration selection, and an adaptive dataflow mechanism, Efficient-GMS successfully improves the workload under diverse input scenarios and hardware configurations. Our experimental results demonstrate that our approach achieves an average speedup of **1.73**× and **1.52**× on the $(\text{g-mm-s})_{\text{s}}$ kernel for RGCN and SpConv-based networks, respectively. Furthermore, the end-to-end acceleration effects are significant, with average speedups of **1.43**×, **1.31**×, and **1.15**× on RGCN, SpConv-based networks, and MoE tasks, respectively. We believe that the insights and techniques can benefit all tasks that fit in $(\text{g-mm-s})_{\text{s}}$ abstraction.

Acknowledgment

This work was sponsored by the National Natural Science Foundation of China (No. U21B2031), Shanghai Rising-Star Program (No. 24QB2706200).

REFERENCES

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1534–1543, 2016.

[2] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1534–1543, 2016.

[3] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In Proceedings of the twelfth ACM international conference on web search and data mining, pages 384–392, 2019.

[4] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 3075–3084, 2019.

[5] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018.

[6] NVIDIA Corporation. Cutlass grouped kernel schedulers. https://github.com/NVIDIA/cutlass/blob/main/media/docs/grouped_scheduler.md. Accessed: 11-18-2024.

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research, 23(120):1–39, 2022.

[8] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428, 2019.

[9] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In Proceedings of the web conference 2020, pages 2331–2341, 2020.

[10] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. Utilizing graph machine learning within drug discovery and development. Briefings in bioinformatics, 22(6):bbab159, 2021.

[11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In 2012 IEEE conference on computer vision and pattern recognition, pages 3354–3361. IEEE, 2012.

[12] Benjamin Graham and Laurens Van der Maaten. Submanifold sparse convolutional networks. arXiv preprint arXiv:1706.01307, 2017.

[13] Yixin Guo, Pengcheng Li, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. Exploring gnn based program embedding technologies for binary related tasks. In Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, pages 366–377, 2022.

[14] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

[15] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024.

[16] Yangyang Li, Yipeng Ji, Shaoning Li, Shulong He, Yinhao Cao, Yifeng Liu, Hong Liu, Xiong Li, Jun Shi, and Yangchao Yang. Relevance-aware anomalous users detection in social network via graph neural network. In 2021 international joint conference on neural networks (IJCNN), pages 1–8. IEEE, 2021.

[17] Petar Ristoski, Gerben Klaas Dirk De Vries, and Heiko Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15, pages 186–194. Springer, 2016.

[18] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In International semantic web conference, pages 498–514. Springer, 2016.

[19] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15, pages 593–607. Springer, 2018.

[20] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15, pages 593–607. Springer, 2018.

[21] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. arXiv preprint arXiv:1811.00937, 2018.

[22] PyTorch Geometric Team. Pytorch geometric (pyg)-fastrgcnconv. https://github.com/pyg-team/pytorch_geometric/blob/master/torch_geometric/nn/conv/rgcn_conv.py. Accessed: 11-18-2024.

[23] Xiaogang Wang, Marcelo H Ang, and Gim Hee Lee. Voxel-based network for shape completion by leveraging edge generation. In Proceedings of the IEEE/CVF international conference on computer vision, pages 13189–13198, 2021.

[24] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (tog), 38(5):1–12, 2019.

[25] Xin Wen, Tianyang Li, Zhizhong Han, and Yu-Shen Liu. Point cloud completion by skip-attention network with hierarchical folding. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 1939–1948, 2020.

[26] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. ACM Computing Surveys, 55(5):1–37, 2022.

[27] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1912–1920, 2015.

[28] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1912–1920, 2015.

[29] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1):4–24, 2020.

[30] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? arXiv preprint arXiv:1905.07830, 2019.

[31] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. AI open, 1:57–81, 2020.

[32] Keren Zhou, Karthik Ganapathi Subramanian, Po-Hsun Lin, Matthias Fey, Binqian Yin, and Jiajia Li. Fasten: Fast gpu-accelerated segmented matrix multiplication for heterogenous graph neural networks. In Proceedings of the 38th ACM International Conference on Supercomputing, pages 511–524, 2024.

HiSpTRSV: Exploring Tile-Level Parallelism for SpTRSV Acceleration on FPGAs

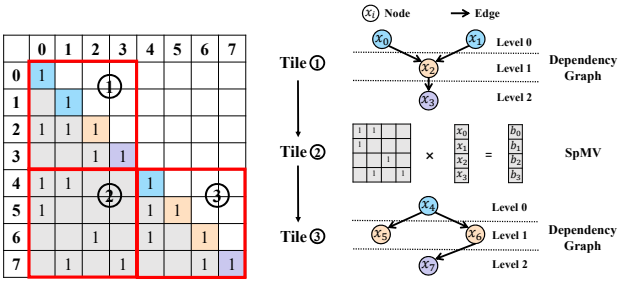
Fan Sun, Fang Dong*, Dian Shen
School of Computer Science and Engineering
Southeast University
{fansun, fdong, dshen}@seu.edu.cn

***Abstract*—Sparse Triangular Solve (SpTRSV) is a critical level-2 kernel in sparse Basic Linear Algebra Subprograms (BLAS). While Field-Programmable Gate Array (FPGA) accelerators for SpTRSV focus on optimizing individual tiles, they overlook inter-tile parallelism. Designing an inter-tile parallelism accelerator poses challenges, including constructing fine-grained dependency graph, handling communication overhead, and balancing workloads. HiSpTRSV addresses these challenges through dependency graph parsing, tile-based highly parallel algorithm, filtering mechanisms, and bidirectional matching with modular indexing. Experiments show that HiSpTRSV outperforms the state-of-the-art SpTRSV accelerator in terms of a 34.3% performance improvement. HiSpTRSV achieves a 3.58× speedup and 9.59× higher energy efficiency compared to GPUs.**

***Index Terms*—Hardware accelerator, Computer systems organization, Reconfigurable computing.**

I. Introduction

SpTRSV is a fundamental computational operation for solving sparse triangular linear systems, expressed as $L\vec{x} = \vec{y}$, where L represents a sparse lower triangular matrix, $\vec{x}$ is the dense vector to be solved, and $\vec{y}$ is a known dense vector. SpTRSV is widely applied across various fields, including scientific computing [4], graph learning in machine learning [5], and dynamics analysis in physical simulations [6].



(a) 8×8 Sparse Matrix L (b) Sequential Computation for Three Tiles.

Fig. 1. Level-ST Algorithm Workflow.

In solving the SpTRSV problem, calculations have strict data dependencies: to compute an element $x[i]$, all the elements from $x[0]$ to $x[i-1]$ must first be computed. To address this challenge, researchers introduce the Level-Set algorithm to improve the parallelism of SpTRSV computations [15]. This algorithm represents matrix L as a dependency graph with **nodes** and **edges**. It achieves faster computation by processing these components in parallel. Furthermore, when handling large-scale matrices, the matrix is typically divided into tiles, and the Level-Set [7] algorithm applies to each tile sequentially. Fig. 1 shows an example to explain this computational process. Fig. 1(a) shows an 8×8 sparse triangular matrix L, divided into 3 tiles (indicated by red boxes). Fig. 1(b) illustrates the sequential process for solving tiles 1, 2, and 3, where dependency graphs are generated for tiles 1 and 3 based on the Level-Set algorithm.

Taking tile 1 from Fig. 1(b) as an example, nodes $x[0]$ and $x[1]$ in Level 0 are first solved in parallel, and the associated edges (arrows, indicating node dependencies) are computed. Next, node $x[2]$ in Level 1 is solved, along with its edges, followed by solving node $x[3]$ in Level 2. When tile 1 is solved, the values of $x[0]$ to $x[3]$ are determined. Next, tile 2 perform Sparse Matrix-Vector Multiplication (SpMV) computation based on $x[0]$ to $x[3]$. Finally, tile 3 is solved using results from tile 2 following similar steps.

Since the nodes and edges within tiles can be computed in parallel, researchers attempt to accelerate SpTRSV computation using FPGAs or GPUs [7]–[9]. However, these studies consider either inner-tile or inter-tile parallelism individually, failing to fully exploit SpTRSV's parallel potential. Therefore, a natural question arises: can both inner-tile and inter-tile parallelism be achieved simultaneously? In fact, this is feasible. Fig. 2 illustrates the computation flow for this concept, we overlapped computations between tile 1 and tile 2 using a fine-grained pipelined design, realizing inter-tile parallelism. Additionally, nodes and edges within each tile are computed in parallel, realizing inner-tile parallelism. It should be noted that for larger matrices, more tiles can be computed in parallel. We will discuss this in detail in Section II.

However, implementing an accelerator with the aforementioned characteristics on FPGAs presents several challenges. **Dependency Parsing (C1):** Inter-tile parallelism requires more complex dependency graphs to describe the relationships, and randomness in matrix sparsity and size further increases the challenge in constructing the dependency graph. **Routing and Resources (C2):** Fine-grained computation requires more submodules, which not only increases on-chip resource consumption but also introduces additional communication overhead. Meanwhile, the interconnection between submodules further complicates the routing stage. **Load Balanc-**

*The corresponding author is Fang Dong.

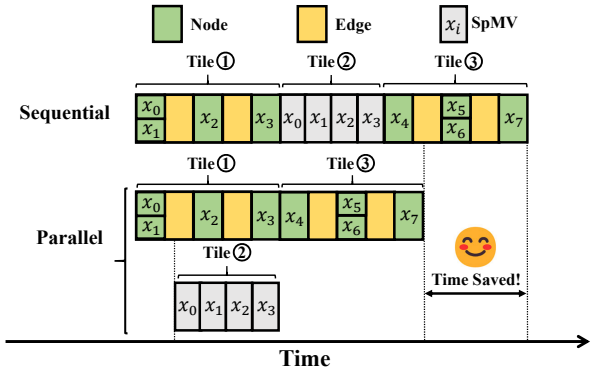
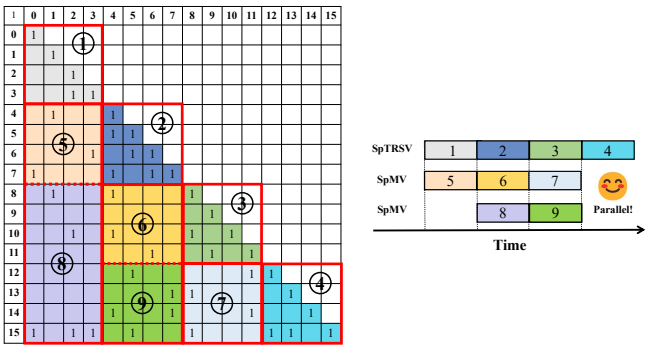


Fig. 2. Comparison of Computational Flows.



(a) 16×16 Sparse Matrix L (b) Parallel Computation Flow

Fig. 3. High Parallelism via Dependency Decomposition

ing (C3): Randomness in sparse matrices makes the element count for each Processing Element (PE) uncertain, leading to imbalanced computational loads across PEs.

In this paper, we propose HiSpTRSV to address these challenges, with the following innovation:

- **Flexible Data Dependency Parsing:** We design a fine-grained data dependency graph to describe the dependencies in the SpTRSV solution process. This enables highly parallel computation. Additionally, HiSpTRSV enables automated data dependency graph construction for matrices of any size. This addresses challenge **C1**.
- **Fully Streamed High-Parallelism SpTRSV:** We develop a novel Level-Set algorithm that achieves high parallelism through fine-grained tile partitioning, unlocking SpTRSV's parallel potential. Within the module, resource overhead reduces through resource reuse. Between modules, routing complexity and communication overhead are reduced through stream-based communication and data filtering. This addresses challenge **C2**.
- **Mod-Based Bidirectional Index Match:** We design bidirectional indexing strategy based on modular arithmetic, which allocating elements to each PE, ensuring correct matching between dependent elements across modules, and achieving load balancing among PEs. This addresses challenge **C3**.

II. Motivation and Tiled Highly Parallelized Level-Set Algorithm

In this section, we analyze the dependencies among different tiles in matrix L at a fine-grained level, demonstrating that these inter-tile dependencies are not strongly correlated. This observation creates opportunities for designing a highly parallel SpTRSV algorithm. For example, in Fig. 3(a), solving tile 2 depends on the SpMV result from tile 5, but it does not depend on tile 8. This means that the computations involving tile 2 and tile 8 can be carried out in parallel. Fig. 3 illustrates the analysis of inter-tile dependencies in the matrix, with the aim of implementing a highly parallel SpTRSV algorithm.

Fig. 3(a) presents a 16x16 lower triangular sparse matrix, which is divided into two types of tiles: tiles 1, 2, 3, and 4 represent the SpTRSV tiles, while tiles 5, 6, 7, 8, and 9 represent the SpMV tiles. When solving tile 1, tile 5 performs the SpMV computation in a pipelined parallel manner based on the results from tile 1, and stores the result on-chip. Once tile 1 is solved, tiles 2, 6, and 8 are computed in parallel. Similarly, tiles 3, 7, and 9 are computed in parallel, and once completed, the results of tiles 9 and 7 are accumulated. Finally, once tile 3 has completed its solution, tile 4 uses the stored result from tile 7 to proceed with its own computation. The computational flow is illustrated in Fig. 3(b), demonstrating that HiSpTRSV achieves higher parallelism.

Based on the above analysis, we design the Tiled High-Parallelism Level-Set Algorithm (THLS). This algorithm employs a tile-based strategy for computation, achieving high parallelism in SpTRSV while adhering to FPGA resource constraints. It enables full parallelism both inter-tile and inner-tile.

In the THLS algorithm, we organize the computation sequence for the SpTRSV and SpMV tiles as data dependency graphs G and M, with the computation step size at each stage managed by the $index$. The data dependency graph is allocated to P Processing Elements for parallel execution. Each PE executes SpTRSV computations (including node and edge computations) and SpMV computations in parallel, based on dependency graph indexing. When the tile completes, the PEs accumulate the results from the SpMV and SpTRSV tiles, storing the results on-chip in preparation for the next tile computation.

III. HiSpTRSV Design and Implementation

A. HiSpTRSV Architecture Overview

As shown in Fig. 4, HiSpTRSV is a dataflow-based architecture that consists of multiple Processing Element Groups (PEGs), several data load modules, a data merge module, and a data write module. Each PEG connects to a High Bandwidth Memory (HBM) channel. The data load module is responsible for reading the data dependency graph and the dense vector $\vec{y}$, streaming this data to the PEGs. The data merge module consolidates the computation results from the various PEGs,

Algorithm 1 Tiled High-Parallelism Level-Set Algorithm

Require: (1) SpTRSV Dependency Graph G, (2) SpMV Computation Graph M, (3) Dependency Graph Computation Index $Index_G$, (4) SpMV Region Computation Index $Index_M$, (5) Vertical Tiling Size W, (6) Matrix Dimension N, (7) Number of PEs P.

Ensure: $L\vec{x} = \vec{y}$

for $t = 0$ to N/W **do**
  Read y_t on chip
  for $p = 0$ to $P - 1$ **parallel do**
    Update $layer$
    parallel do ▷ SpTRSV and SpMV are parallel.
      for $l = 0$ to $layer - 1$ **do** ▷ SpTRSV
        $(node, edge) \leftarrow Index_G[l]$
        for $n = idx_g$ to $idx_g + node - 1$ **do**
          (row, col, val) $\leftarrow G_{node}[n]$
          $x[row] \leftarrow (y_t[row] - y_{add}[row])/val$
        end for
        Update idx_g
        for $e = idx_g$ to $idx_g + edge - 1$ **do**
          $(row, col, val) \leftarrow G_{edge}[e]$
          $y_e[row] \leftarrow x[row] * val + y_e[row]$
        end for
      end for
      $m \leftarrow Index_M[l]$
      for $s = idx_m$ to $idx_m + m - 1$ **do** ▷ SpMV
        $(row, col, val) \leftarrow M_{spmv}[s]$
        $y_s[row] \leftarrow x[row] * val + y_s[row]$
      end for
    end parallel
    Update idx_m
    $y_{add} \leftarrow y_e + y_s$
  end for
  Write x_t
end for

while the data write module handles writing the dense vector $\vec{x}$ back to host memory.

Each PEG includes four core modules: computation module, filter module, cache module, and control module. The computation module includes the SpTRSV and SpMV modules, responsible for performing calculations in different tiles; the filter module filters out invalid elements during the synchronization process; the cache module manages the storage for intermediate computation results; and the control module serves as the core unit that oversees the operation of the PEG. We employ a fully streaming communication mechanism between the modules to optimize hardware resource utilization and simplify routing complexity.

In architecture-level communication, data synchronization between different PEGs is achieved through synchronous streams. We use a loopback data stream at the terminal PEG to return the synchronized data. This effectively implements a circular data synchronization stream among the PEGs, ensuring that each PEG receives the current solution $\vec{x}$, which

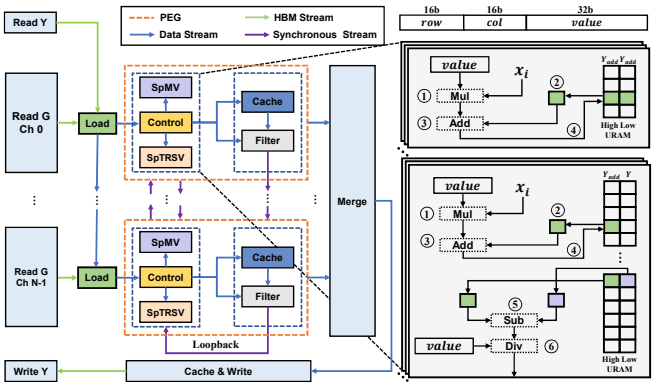


Fig. 4. HiSpTRSV Architecture and Computation Order for SpMV(Top) and SpTRSV(Bottom) Modules.

is required for the SpTRSV and SpMV computations in the next iteration. The dense vector $\vec{y}$ is loaded through a single HBM channel and then distributed to each loading module via a chained broadcast mechanism.

B. Processing Element Group (PEG) Design

To fully utilize HBM bandwidth, the PEG receives 512-bit wide data, containing 8 elements from the dependency graph. These 8 elements are distributed across 8 PEs, with each element using a 64-bit storage format that includes a 16-bit row index, a 16-bit column index, and a 32-bit value. The SpMV and SpTRSV serve as the core computational units in the PEG. The numerical labels in Fig. 4 indicate the execution sequence for the computation steps.

The SpMV module utilizes UltraRAM (URAM) units to store computation results, with the execution sequence comprising multiplication, read, addition, and write-back operations. Each URAM address is determined by the row index of the element. Once SpMV computations complete, data in URAM transfers in parallel to the SpTRSV module for accumulation. Given that the URAM unit is 72 bits wide, we divide the storage unit into two 36-bit regions (high and low) to store two accumulated values, optimizing the utilization of URAM resources. The design of Serpens and Sextans was referenced to accelerate the SpMV computation process [12], [18].

The SpTRSV module completes edge computations through steps similar to SpMV. The high region in URAM stores accumulation results from edge computations and SpMV computations, while the low region stores the $\vec{y}$ values. When the computation dependent on $x[i]$ is completed, the PEG first subtracts the Y_{add} value from the Y value. Then, it calculates $x[i]$ by performing a division operation.

C. Fine-Grained Data Dependency Graph

The dependency graph describes the SpTRSV computation process. We performed dependency analysis on matrix L at a fine-grained level and constructed a data dependency graph through tiling to support inter-tile parallel computation in SpTRSV. As shown in Fig. 5, we present an example data dependency graph for tiles 1, 5, and 8 from Fig. 3(a), with

row indices, column indices, and values labeled to correspond to elements in the sparse matrix.

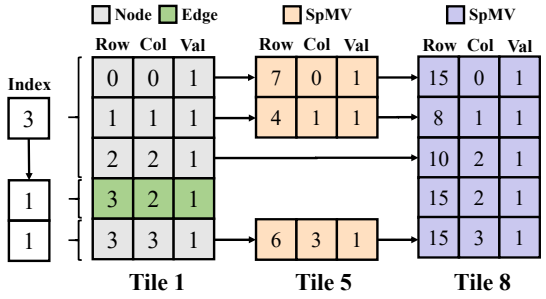


Fig. 5. Data Dependency Graph for Tiles 1, 5, and 8 in the 16x16 Matrix in Fig. 3(a).

The index list in Fig. 5 illustrates the solution process: First, three nodes are solved in parallel, followed by the computation of one edge, and finally, one node is solved. HiSpTRSV synchronously computes tile 1 and tile 5 based on dependency relationships. Once tile 1 is solved and the SpMV computation for tile 5 is complete, HiSpTRSV computes SpMV of tile 8 use dependency relationships and the values $x[0]$ to $x[3]$. The computation results from tile 8 are stored on-chip to enable subsequent computations for other tiles. Notably, HiSpTRSV can generate dependency graphs for matrices of any size, ensuring flexibility in its application.

D. Data Filtering Unit

HiSpTRSV assigns elements to PEs through modulo operations on row indices. Due to the randomness of sparse matrices and element dependencies, some PEs may be temporarily idle. Taking the sparse matrix in Fig. 3(a) as an example, Fig. 6 illustrates the solution sequence for tiles 1 and 2 by two PEGs (assuming each PEG contains two PEs). The number is the row index for the element. Since node 3 depends on node 2, nodes 0, 1, and 2 are solved in parallel during the first solution cycle while PE3 remains idle. In the second cycle, only PE3 solves node 3 while other PEs are idle. To align PE computation lengths, "bubble elements" (invalid elements) are added to the dependency graph. It should be noted that HiSpTRSV consists of 16 PEGs, each containing 8 SpTRSV PEs and 4 SpMV PEs, resulting in a total of 192 PEs to compute in parallel. Although increasing the PE count expands the scale for parallel computation, it also raises the likelihood for bubble elements. Therefore, we design a data filtering unit to handle bubble elements.

During the node solving process, the solution results stored in URAM need to be synchronized. For example, when PE3 solves node 7, it requires the values of $x[4]$ and $x[6]$, where $x[4]$ is stored in the URAM of PEG0. Therefore, data synchronization is required to ensure each PEG can access all $\vec{x}$ values within its own URAM. However, bubble elements introduce additional communication overhead to the synchronization process. We design a data filtering unit that identifies which elements should be written to URAM in parallel as PEGs perform write operations. When elements are

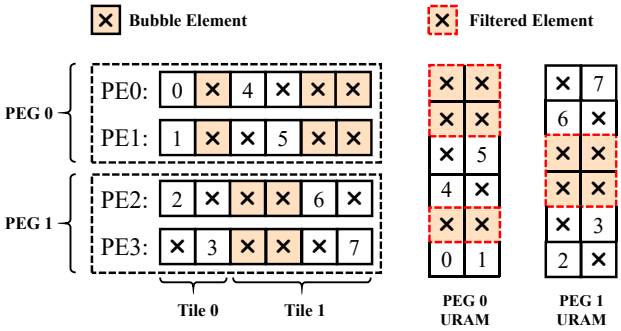


Fig. 6. Bubble Element Filtering Process.

bubble elements, URAM will reject storing them and record how many it successfully stores. This count represents the transmission length for data synchronization. Fig. 6 shows URAM-stored elements for PEG0 and PEG1, with red dashed lines indicating filtered elements that are neither stored in URAM nor participate in data synchronization.

E. Mod-Based Bidirectional Index Match

HiSpTRSV organizes the elements of SpMV and SpTRSV into a data dependency graph, assigns elements to PEs for computation, and ensures optimal load balancing. Fig. 7 shows the element allocation using modulo-based bidirectional matching.

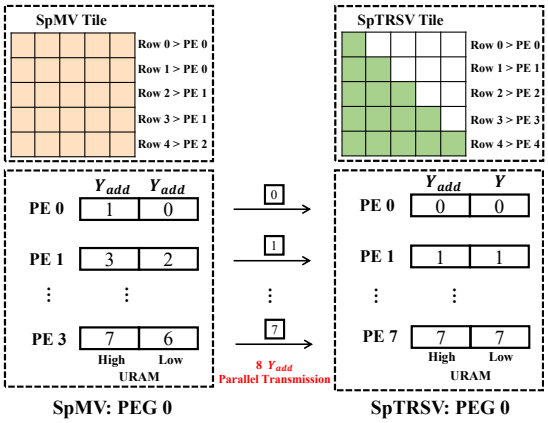


Fig. 7. Tiled SpMV and SpTRSV Element Mapping to PEs and URAM Layout.

For the elements in the SpMV tile, we assign two consecutive elements to the same PE because the URAM is divided into high and low regions to store accumulated values. The allocation method is: $row/2\%N_PEs$, where N_PEs represents the total number processing elements. For the elements in the SpTRSV region, only the high region of the URAM stores accumulated values. Meanwhile, one element is assigned to one PE at a time, and the allocation method is: $row\%N_PEs$. It is worth noting that SpTRSV relies on the results generated by SpMV. Therefore, the SpMV units transfer the accumulated values Y_{add} in parallel to the SpTRSV units, using row as a common index to ensure the accumulated values match correctly between SpMV and SpTRSV.

TABLE I
PERFORMANCE OF HISPTRSV ON ALVEO U55C AND CUSPARSE ON NVIDIAV100 AND RTX3060

Matrix Properties					Performance (GFLOPS)				Improvement
ID	Matrix Name	Size	NNZ	Density	RTX 3060	NVIDIA V100	LevelST	HiSpTRSV	Percentage
M1	lp1	534,388	1,088,904	3.81E-06	0.95	0.68	1.71	**2.48**	45.0%
M2	parabolic_fem	525,825	2,100,225	7.60E-06	1.41	1.21	4.02	**4.82**	19.9%
M3	com-Amazon_trig	334,863	718,060	1.63E-07	0.89	0.51	2.07	**2.78**	34.3%
M4	gsm_106857_trig	589,446	2,932,426	8.44E-06	1.36	1.37	3.62	**4.81**	32.9%
M5	hollywood-2009_trig	1,139,905	2,155,683	1.66E-06	0.88	1.21	1.51	**2.46**	62.9%
M6	ins2_trig	309,412	1,392,214	1.45E-05	1.25	0.61	2.81	**3.86**	37.4%
M7	c-big_trig	345,241	1,343,050	1.13E-05	1.71	0.79	6.65	**7.50**	12.8%
M8	TSOPF_FS_b162_c4_trig	40,798	791,443	4.39E-07	0.53	0.39	2.63	**3.52**	33.8%
M9	ASIC_680k_trig	682,862	1,023,329	2.19E-06	0.082	0.077	0.61	**0.88**	44.3%
M10	hood	220,542	5,494,489	1.13E-04	1.26	1.63	5.11	**6.10**	19.4%
M11	roadNet-CA_trig	1,971,281	4,737,888	1.22E-06	0.35	0.61	1.28	**1.45**	13.3%
M12	com-DBLP_trig	317,080	440,407	4.38E-06	0.81	0.34	1.49	**2.15**	44.0%
M13	G3_circuit_trig	1,585,478	4,188,828	1.67E-06	0.075	0.21	0.92	**1.26**	37.0%
M14	tx2010_trig	914,231	5,579,489	6.68E-06	0.65	0.63	1.57	**1.89**	20.4%
M15	net50	16,320	480,760	1.81E-03	2.55	0.41	4.01	**6.68**	66.6%
M16	c-73b	169,422	724,348	2.52E-05	1.71	0.57	4.28	**5.89**	37.6%

IV. EVALUATION

A. Experiment Setup

We design the circuit using AMD Vitis HLS C++ and implement the prototype with Vitis 2023.1. The experiments are mainly conducted on the Xilinx Alveo U55C platform [1], Nvidia RTX 3060 [2], and Tesla V100 [3]. The HLS code is developed based on the RapidStream TAPA [11] streaming framework, with layout optimization achieved through Autobridge [10]. The performance and power consumption are measured on actual hardware: GPU power is recorded using the NVML tool, while FPGA power is collected with the xbutil tool during kernel execution. The measurement method for kernel execution time is kept consistent with LevelST [7] to ensure fair comparisons.

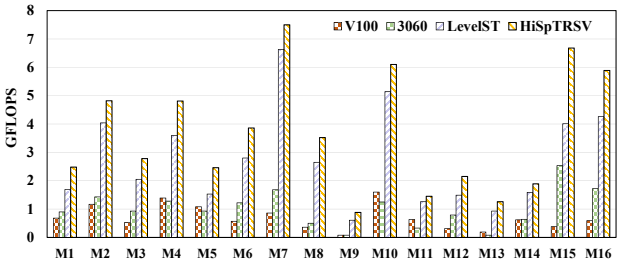


Fig. 8. FPGA and GPU Performance in GFLOPS.

We comprehensively compare our design against the state-of-the-art open-source SpTRSV accelerator LevelST [7] and the NVIDIA cuSparse library. The evaluation use 16 matrices from the SuiteSparse database [13], which are the same as those in the LevelST test set. Matrix sizes range from 16.3K to 1.97M rows, with non-zero elements between 440K and 5.58M. Tab. I lists detailed information for each matrix, including its size, the number of non-zero elements, and density. Matrix names with the suffix "trig" denote those derived through decomposition, while names without a suffix indicate directly extracted lower triangular matrices.

B. Overall Performance

The formula we use to compute GFLOPS is given by:

$$GFLOPS = \frac{2 \times NNZ}{10^9 \times \text{execution time}} \quad (1)$$

Here, NNZ represents the non-zero element count. We use GFLOPS as the primary performance metric, which directly reflects the computational efficiency and solving speed.

Tab. I provides a detailed list of the test results, with the last column showing the percentage improvement in performance compared to the LevelST. The 16 matrices exhibit throughput ranging from 0.88 to 7.50 GFLOPS. Compared to the state-of-the-art SpTRSV accelerator LevelST, HiSpTRSV achieves a 1.34× geometric mean speedup (up to 1.66×).

C. Communication

The random distribution in sparse matrix non-zero elements directly impacts the number of bubble elements within the dependency graph. The fewer the bubble elements, the lower the benefit of the filtering module. Fig. 9 shows the ratio of communication volume before and after using the filter module. Matrices M7, M15, and M16 display a relatively uniform pattern in non-zero element distribution, leading to balanced workloads across PEs and fewer bubble elements; consequently, communication volume does not decrease.

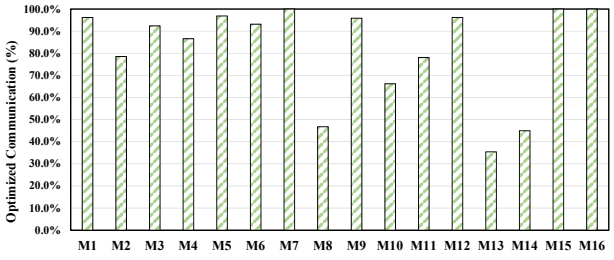


Fig. 9. Filter Module Optimization Ratio.

Conversely, matrices M8, M10, M11, M13, and M14 exhibit significant element clustering, with non-zero elements densely concentrated in specific rows. This distribution leads to severe load imbalance across PEs and generates numerous bubble elements, allowing the filter module to achieve significant optimization in these cases. The variation in optimization effectiveness not only confirms the design concept of the

filtering mechanism but also highlights the importance of matrix structure in parallel computation scheme design.

D. Resource Utilization and Design Frequency

Tab. II presents the resource utilization of HiSpTRSV on the U55C board. The SpTRSV module contains 8 PEs, while the SpMV module contains 4 PEs. We employ optimization directives to partition the Block RAM (BRAM) blocks within PEs to prevent BRAM conflicts between PEs, which implies that storing a single dense vector $\vec{x}$ consumes 8 BRAM blocks. Since BRAM features dual ports and can serve two PEs in a single cycle, the SpTRSV and SpMV modules utilize 32 and 16 BRAM blocks, respectively.

TABLE II
RESOURCE UTILIZATION OF HISPTRSV PROTOTYPE ON XILINX U55C FPGA BOARD

Resource	Available	Used	Utilization (%)
BRAM	2016	968	48.02
DSP	9024	1220	13.52
FF	2,607,360	821,979	31.53
LUT	1,303,680	547,430	41.99
URAM	960	640	66.67

URAMs are primarily utilized to cache intermediate results during computation in HiSpTRSV. The SpTRSV and SpMV modules each employ 8 URAM blocks, while the filter module, cache module, and write module collectively share 8 URAM blocks. Consequently, each PEG consumes a total of 40 URAM blocks. Given that the system incorporates 16 PEGs, the overall design occupies 640 URAM blocks. To fully utilize HBM's maximum bandwidth at 450 MHz with 512-bit ports, our kernels require only a frequency of 225 MHz [14]. HiSpTRSV operates at 246 MHz, successfully meeting this requirement.

E. Comparison to GPU

Fig. 8 compares the performance between Nvidia cuSparse running on 3060 and V100, and HiSpTRSV running on U55C. We select the better results between the 3060 and V100 for comparison with HiSpTRSV. For performance, HiSpTRSV achieve a geometric mean 3.58× speedup. For energy efficiency, HiSpTRSV achieve a geometric mean 9.59× improvement.

3060 GPU vs. HiSpTRSV. For performance, on average (geomean), HiSpTRSV is 4.10× faster than the 3060 GPU (up to 16.80×). On the other hand, Fig. 10 compares their energy efficiency. HiSpTRSV achieves better energy efficiency than the 3060 across all matrices, showing a geomean improvement by 9.82× (reaching up to 39.56×). When processing smaller sparse matrices, the RTX 3060 performs better due to its higher memory clock speed.

V100 GPU vs. HiSpTRSV. For performance, on average, HiSpTRSV is 5.44× faster than the V100 GPU (up to 16.29×). The geomean energy efficiency improvement is 25.08× (up to 59.94×). When processing larger sparse matrices, the V100 outperforms the RTX 3060 due to its larger on-chip memory and higher off-chip memory bandwidth.

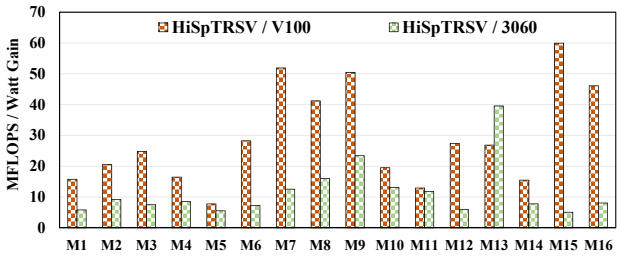


Fig. 10. Energy Efficiency Improvement of FPGA over GPU.

Due to the sparsity and dependencies in SpTRSV, some threads within the GPU remain idle during SpTRSV computations. However, these idle threads continue to consume power, leading to lower core utilization and higher energy consumption for the GPU. When handling sparse workloads, GPU peak utilization remains at only 0.6%, while average utilization reaches just 0.1% for workloads in [12], [16]. In addition, branch prediction and higher clock speeds significantly increase GPU kernel latency and power consumption [2], [17]. In contrast, we leverage FPGA reconfigurability to customize designs for specific applications, reducing additional latency and power consumption.

V. CONCLUSION

In this work, we develop the HiSpTRSV design to address the challenges faced in accelerating tile-based SpTRSV on HBM-based FPGAs. First, we implemented fine-grained dependency resolution at a granular level, enabling highly parallel computation in SpTRSV. Second, we develop a tile-based highly parallel SpTRSV algorithm and a stream-based filtering mechanism to achieve high parallel computation and low-overhead communication. Third, we design a modular bidirectional index matching mechanism to achieve balanced workload distribution. Moreover, HiSpTRSV can automatically construct and compute dependency graphs for matrices with any size. Experimental results demonstrate that our design significantly outperforms the state-of-the-art FPGA-based SpMV accelerator, LevelST, and Nvidia GPUs using cuSparse.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under Grants, No. 62232004, 62272101, Jiangsu Provincial Frontier Technology Research and Development Program under Grant BF2024070, Natural Science Foundation of Jiangsu Province under grant BK20230083, Shenzhen Key S&T Special Projects under Grant KJZD20240903100814018, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No.93K-9, and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization and Collaborative Innovation Center of Wireless Communications Technology. We also thank the Big Data Computing Center of Southeast University for providing the experiment environment and computing facility.

REFERENCES

[1] AMD, "Alveo U55C High Performance Compute Cards," [Online]. Available: https://www.amd.com/en/products/accelerators/alveo/u55c/a-u55c-p00g-pq-g.html.

[2] NVIDIA, "GeForce RTX 3060," [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3060/.

[3] NVIDIA, "Tesla V100," [Online]. Available: https://www.nvidia.com/en-us/data-center/v100/.

[4] T. A. Davis and S. Rajamanickam, "An Efficient Multifrontal Solver for Sparse Matrices," SIAM Journal on Scientific Computing, vol. 32, no. 2, pp. 521–544, 2010.

[5] Y. Yu, J. Chen, T. Gao, and M. Yu, "DAG-GNN: DAG structure learning with graph neural networks," in Proceedings of the International Conference on Machine Learning, 2019, pp. 7154–7163.

[6] D. Negrut, R. Serban, and H. Mazhar, "Large-Scale Parallel Multi-Body Dynamics with Application to Vehicle Mobility Simulations," Journal of Computational and Nonlinear Dynamics, vol. 9, no. 4, pp. 041010, 2014.

[7] Z. He, L. Song, R. F. Lucas, and J. Cong, "LevelST: Stream-based Accelerator for Sparse Triangular Solver," in Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, Mar. 2024, pp. 67–77.

[8] Z. Lu and W. Liu, "TileSpTRSV: a tiled algorithm for parallel sparse triangular solve on GPUs," CCF Transactions on High Performance Computing, vol. 5, pp. 129–143, 2023.

[9] N. Ahmad, B. Yilmaz, and D. Unat, "A Split Execution Model for SpTRSV," IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 11, pp. 2809–2822, Nov. 2021.

[10] L. Guo, Y. Chi, J. Wang, J. Lau, W. Qiao, E. Ustun, Z. Zhang, and J. Cong, "AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs," in Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, Feb. 2021, pp. 229–240.

[11] L. Guo, P. Maidee, Y. Zhou, C. Lavin, J. Wang, Y. Chi, W. Qiao, A. Kaviani, Z. Zhang, and J. Cong, "RapidStream: Parallel Physical Implementation of FPGA HLS Designs," in Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, Feb. 2022, pp. 106–117.

[12] L. Song, Y. Chi, L. Guo, and J. Cong, "Serpens: A High Bandwidth Memory Based Accelerator for General-Purpose Sparse Matrix-Vector Multiplication," in Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, CA, USA, Jul. 2022, pp. 211–216.

[13] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," ACM Transactions on Mathematical Software, vol. 38, no. 1, pp. 1:1–1:25, 2011.

[14] X. Chen, F. Cheng, H. Tan, Y. Chen, B. He, W.-F. Wong, and D. Chen, "Demystifying the Memory System of Modern Datacenter FPGAs for Software Programmers through Microbenchmarking," in Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual Event, USA, Feb. 2021, pp. 24–35.

[15] X. Wang, L. Liu, W. Xue, and L. Wu, "swSpTRSV: A fast sparse triangular solve with sparse level tile layout on Sunway architectures," in Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Vienna, Austria, Feb. 2018, pp. 338–350.

[16] L. Song, L. Guo, S. Basalama, Y. Chi, R. F. Lucas, and J. Cong, "Callipepla: Stream centric instruction set and mixed precision for accelerating conjugate gradient solver," in Proceedings of the 2023 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, Feb. 2023, pp. 247–258.

[17] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between FPGA and GPU," in Proceedings of the 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, Apr. 2019, pp. 127–135.

[18] L. Song, Y. Chi, A. Sohrabizadeh, Y.-k. Choi, J. Lau, and J. Cong, "Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication," in Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual Event, Feb. 2022, pp. 65–77.

Rewire: Advancing CGRA Mapping Through a Consolidated Routing Paradigm

Zhaoying Li[1], Dan Wu[1], Dhananjaya Wijerathne[2§], Dan Chen[1✉], Huize Li[3§], Cheng Tan[4] and Tulika Mitra[1]
[1]*School of Computing, National University of Singapore*, [2]*AMD*, [3]*University of Central Florida*, [4]*Google*
{*zhaoying, danwu20, chend, tulika*}*@comp.nus.edu.sg*, DMD.Wijerathne@amd.com, chengtan@google.com, huize.li@ucf.edu

***Abstract*—Coarse-Grained Reconfigurable Arrays (CGRAs) balance the performance and power efficiency in computing systems. Effective compilers play a crucial role in fully realizing its potential. The compiler maps Data Flow Graphs (DFGs), which represent compute-intensive loop kernels, onto CGRAs. However, existing compilers often tackle DFG nodes individually, neglecting their intricate inter-dependencies. We introduce a novel mapping paradigm called Rewire that can place and route multiple nodes in one shot. Rewire first generates routing information that is shareable among multiple nodes via propagation. Then, Rewire intersects the routing information to generate individual placement candidates for each node. Finally, Rewire innovatively utilizes data dependencies as constraints to quickly find suitable placement for multiple nodes together. Our evaluation demonstrates that Rewire can generate more near-optimal mappings than prior works. Rewire achieves 2.1x and 1.3x performance improvement and 13.5x and 4.7x compilation time reduction, respectively, compared to two popular mappers.**

I. Introduction

Coarse-grained reconfigurable arrays (CGRAs) [1]–[20] have gained significant attention in recent years due to their excellent balance in terms of flexibility, performance, and energy efficiency. The domain-agnostic characteristic allows CGRAs to accelerate a broad spectrum of applications, such as machine learning, video processing, and graph processing. However, achieving superior performance and power efficiency for these applications heavily relies on efficient compilers to explore the enormous mapping space.

Mapping DFG, which represents the compute-intensive loop kernel, onto CGRAs is a well-known challenging problem [10], [14], [21]–[30]. Figure 1 shows a typical 4×4 CGRA architecture. The CGRA is programmable, consisting of processing elements (PE) executing the operations, on-chip memory to store the data, and a Network on Chip (NoC) to route the data among the PEs. The details of the architecture are exposed to the mapper to generate cycle-by-cycle configurations for the programmable units, including the PEs and the routers.

The mapping involves the placement of the DFG nodes onto the PEs of the CGRA, while routing the data dependencies among these nodes. The complex inter-dependencies among the nodes pose significant difficulty to the mapping algorithm. Complex data dependencies refer to rich but irregular connections among multiple nodes. These nodes subtly affect each other's mapping due to resource contention, which are aggravated by intricate dependencies. Conventional mapping approaches iterate over the DFG nodes individually. For each node, they first select the PE placement candidate and then evaluate the routability of the relevant edges. It is theoretically possible to choose the placement of multiple nodes in one shot and then find the routing of the relevant edges together. Unfortunately, such consolidated placement does not yield feasible routing due to complex

§Dhananjaya Wijerathne and Huize Li were with the National University of Singapore when this research was conducted. This research is supported by the National Research Foundation, Singapore under its Competitive Research Program Award NRF-CRP23-2019-0003. The corresponding author is Dan Chen.

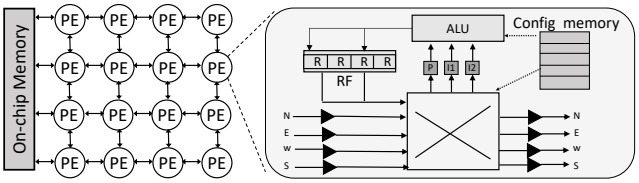


Fig. 1: A typical 4×4 CGRA.

data dependencies. Several works use a hierarchical approach [24]–[26], [31] of partitioning the DFG into several sub-DFGs. However, they still map the nodes individually within each sub-DFG.

There are two disadvantages to existing approaches. First, by mapping each node individually, they lack the ability to handle complex data dependencies collectively. As a result, these approaches need to backtrack whenever they cannot map the current node, causing numerous iterations to explore the multi-node mapping. Nevertheless, they still may fail to reach a feasible mapping within a reasonable compilation time. Second, they are unable to leverage the routing knowledge established for one edge when handling the next edge. Hence, they need to visit the same network repeatedly to evaluate the routability of the PE placement candidates.

To overcome these limitations, we propose *Rewire**, a novel CGRA mapping paradigm that can quickly generate feasible mapping for multiple nodes in one shot. As existing mappers usually create an initial mapping and improve step by step to achieve a valid mapping, Rewire focuses on amending the initial mapping and can take any initial mapping from other mappers, making Rewire orthogonal to these mappers. Rewire first leverages the inherent DFG structure characteristic to share the routing information among multiple DFG nodes and intersects the routing information to find placement candidates for each of the DFG nodes. Compared to previous path-finding-based routing, Rewire devises a propagation mechanism to generate the routing information for multiple nodes together. Finally, Rewire analyzes routing information and prunes the search space using inter-dependencies as constraints to generate multi-node placement. Therefore, Rewire can simultaneously coordinate the placement and routing of multiple nodes, advancing the current mapping paradigm.

We make the following key contributions:

- We identify a fundamental limitation of traditional CGRA mapping approaches that they can only map the DFG nodes individually, limiting the ability to coordinate multi-node mapping. In contrast, we propose a new mapping paradigm that can generate multi-node mapping in one shot.
- To the best of our knowledge, we are the first to propose sharing the routing information among multiple nodes. Through the sharing, we can reuse the routing information and further coordinate the mapping of multiple nodes.

*We name it Rewire due to the reuse of wire (routing) information and its ability to amend an invalid mapping through minimal alterations.

- Our evaluation demonstrates that Rewire can generate much more near-optimal mappings than prior works. Compared to the two popular works, Rewire can achieve 2.1x and 1.3x performance improvement and 13.5x and 4.7x compilation time reduction, respectively.

II. Related Work

Existing mappers can be classified into three main categories: (1) These approaches choose the placement that minimizes a predefined cost function to map a node [21], [23], [32]–[35]. For example, to map a DFG node, they need to evaluate the multiple placement candidates and select the candidate with the minimal cost. This process necessitates evaluating all potential candidates for the current node and often leads to local minima. If the current node cannot be mapped, a higher cost is assigned to the existing placement, and multiple rounds of partial remappings are performed to achieve a feasible solution. (2) These approaches find any placement that can route the edges connected to the current DFG node but may require many iterations to find a feasible mapping [10], [22], [29], [36]. To map a node, they can select any placement candidate if it can route the current data dependencies, and they do not need to evaluate any other candidates. (3) These approaches partition the DFG or the hardware to reduce the search space using a divide-and-conquer approach [24]–[26], [31], but may still rely on the aforementioned methods for each partitioned DFG. For example, if a partitioned sub-DFG is assigned to a sub-CGRA, they only need to evaluate candidates in the corresponding sub-CGRA. Despite substantial research, state-of-the-art CGRA compilers still only map one node at a time and cannot comprehensively manage multi-node data dependencies together. In contrast, Rewire can simultaneously map multiple DFG nodes together, greatly reducing the number of iterations needed to map multiple DFG nodes.

Furthermore, conventional approaches cannot reuse the routing knowledge, with the exception of EMS [28]. There are three levels in mapping: a mapper needs to map multiple DFG nodes, where mapping a DFG node needs to explore multiple placement candidates, and evaluating a candidate needs to route multiple edges. EMS can reuse the routing knowledge for a single edge when attempting different placement candidates. To map a node, EMS starts routing from its parent node and does not specify the PE candidate for the target node. Upon reaching a potential PE candidate, EMS employs a path search to route the other relevant edges. If unsuccessful, EMS still uses the routing information of the first edge and continues to find new potential PE candidates. EMS can only reuse the routing knowledge for a single edge and cannot extend to other edges. In contrast, Rewire can share the routing information among multiple edges, placement candidates, and DFG nodes, making it possible to coordinate the mapping of multiple DFG nodes.

III. Motivation and Intuition

In this section, we use a motivating example to demonstrate the fundamental limitations of the state-of-the-art CGRA compilation techniques and propose concrete strategies to overcome these limitations. Figure 2(a) shows a DFG example where three nodes A, B, and G have been placed onto PEs in the initial mapping. Figure 2(b) provides an example of initial mapping for the DFG depicted in Figure 2(a) onto a 4×4 CGRA. Figure 2(c) illustrates how typical conventional mappers [22], [26] attempt to map the nodes C, D, E, and F: map each node individually in multiple steps and can require checking multiple placement candidates for each DFG node.

The key problem is that such approaches cannot simultaneously coordinate the mapping of multiple nodes. For example, when it

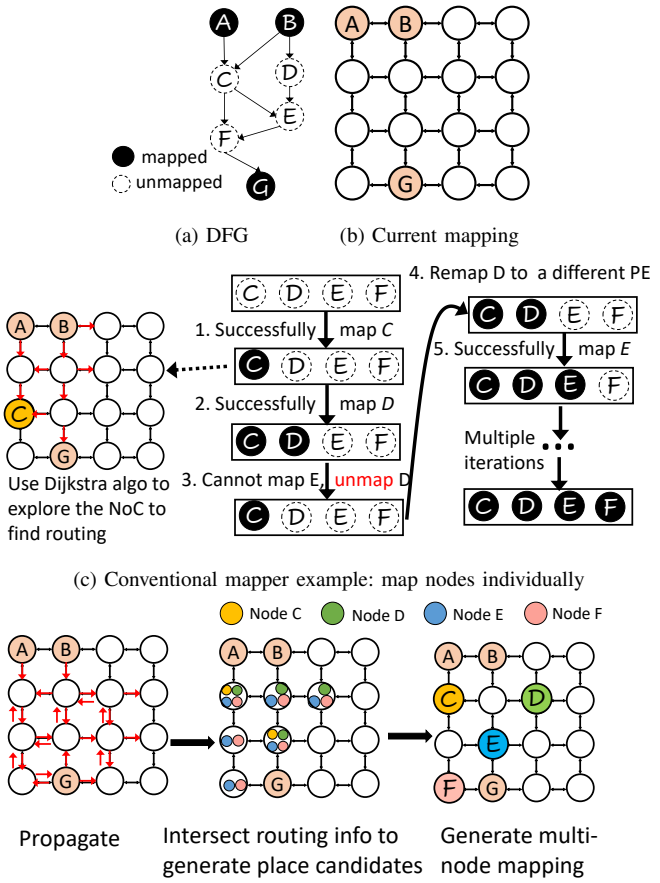


Fig. 2: Comparison between conventional mappers and proposed mapper

attempts to place node C onto a PE, it cannot guarantee that this placement leads to a successful mapping of the remaining nodes D, E, and F. Thus, when node E cannot be mapped (step 3), it usually performs partial remapping by removing a part of the current mapping and changing the placement of the affected nodes to different PEs. In this example, we unmap node D and move it to a different PE. Hence these conventional mappers need multiple backtracking iterations to find a feasible mapping. Another issue is that these conventional mappers cannot reuse routing knowledge either for multiple edges, placement candidates, or DFG nodes. For each edge or placement candidate, the mapper needs to explore the NoC to establish the routing and need to visit the same network multiple times. Yet, they did not reuse the routing information. Furthermore, simple reuse or routing information memorization cannot be applied as the resource status can change from routing one edge to another edge.

In this paper, we demonstrate that it is possible to simultaneously map multiple nodes and share the routing information among relevant edges. Our approach is shown in Figure 2(d). We first propagate the output value of A along the network without specifying any destination node. Any reachable PEs from the PE containing DFG node A can be a potential placement candidate for node C, which is dependent on node A. Moreover, as the nodes E and F are dependent on node C, by transitivity, they are also dependent on node A. Therefore, if C is placed on a reachable PE from A, we can place E and F on the PEs reachable from C. All these reachable PEs that are potential placement candidates for DFG nodes E and F have been identified during the propagation from A. We can use

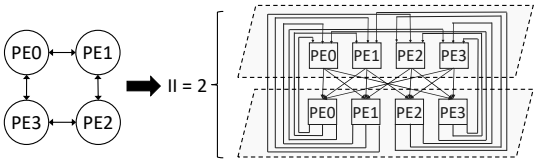


Fig. 3: High-level MRRG example for 2×2 CGRA.

the same propagation information to route $A \rightarrow C$, $C \rightarrow E$, and $C \rightarrow F$. Similarly, we can propagate the output of B and use this information to find potential placement candidates for the nodes D, E, and F, and routing the edges $B \rightarrow D$, $D \rightarrow E$, and $E \rightarrow F$. Finally, we need to conduct backward propagation from node G that has already been mapped, just as we do forward propagation from nodes A and B. The above process identifies the potential placement candidates via forward and backward reachability searches from the already placed ancestor nodes (A and B in the example) and the descendant nodes (G in the example).

However, the placement of a DFG node X needs to satisfy multiple constraints as it needs to be reachable from all the source nodes with which it has dependencies and all the destination nodes that are dependent on this node X. For example, node C should be on the forward propagation routes of nodes A, B, and the backward propagation routes of node G. A simple intersection operation among the propagation routes from the nodes A, B, and G can identify the set of potential feasible placement candidates for node C. Similarly, we can identify the set of potential candidates for nodes D, E, and F. Given these potential candidates for all the nodes we intend to map, we have to select the actual placement candidate for multi-node such that (a) only one DFG node is mapped to a PE candidate, (b) we can route all the data dependencies among these nodes together.

IV. Rewire Algorithm

In this section, we first formulate the mapping problem and then present the Rewire mapper that maps the DFG onto the CGRA. The CGRA mapper first generates a DFG representing the application program and then maps the DFG onto the CGRA with respect to the data dependency. CGRA uses Initiation Interval (II) to evaluate the mapping quality, where II is the number of cycles between consecutive iterations of a loop kernel. The lower the II, the better the performance.

A. Problem Formulation

DFG: We define DFG $D = (V_D, E_D)$ as a directed acyclic graph with V_D representing operations and E_D representing dependencies between operations.

CGRA Graph: CGRA is defined as a graph $G = (V_G, E_G)$. Modulo Routing Resource Graph (MRRG) $H_{II} = (V_H, E_H)$ is a resource graph of the CGRA that is time extended to II cycles [37]. V_H consists of two types of nodes: ALUs (V_H^F) in each PE and ports (V_H^P) in interconnects and Register Files (RFs) [37]. As the CGRA schedule repeats after II cycles, the resources at cycle $II-1$ have connectivity with the resources at cycle 0 in the MRRG. $H^{'} = (V_{H'}, E_{H'})$ is defined as the time extended resource graph of the $G^{'}$ ($V_{H'} \subseteq \mathcal{P}(V_H)$). Figure 3 shows a high-level MRRG example for 2×2 CGRA where we only show the inter-PE links and do not include internal links in the PE.

Problem Definition: Given a kernel and a CGRA, the problem is to construct a time extended MRRG $H_{II} = (V_H, E_H)$ of the CGRA for which there exists a mapping $\phi = (\phi_V, \phi_E)$ from $D = (V_D, E_D)$ to H_{II} which minimizes II. We decompose the problem into three sub-problems: selecting multiple DFG nodes to form a cluster U, propagation for U, and generating the multi-node mapping of U.

Algorithm 1: Rewire Algorithm

```
Input: DFG, CGRA
Output: Valid mapping
1  Sort DFG nodes by topological order;
2  minII = Minimum(recurrenceMII, resourceMII);
3  while mapping is not valid do
4      Generate the initial mapping for current II ;
5      while mapping is not valid do
6          Generate target cluster U;
7          while U is not mapped and is less than α do
8              Propagate for U;
9              Intersect propagation tuples to find placement candidates;
10             if Generate multi-node mapping then
11                 Break;
12             else
13                 Append a node to U;
14         if U are not mapped then
15             Break;
16     II = II + 1;
```

B. Overall Algorithm

Algorithm 1 shows the Rewire algorithm. We first sort DFG nodes by topological order (line 1). The mapper calculates the theoretical Minimum Initiation Interval (MII) based on the hardware resource and inter-iteration data dependency [38] (line 2). Given a DFG and a CGRA, we initiate the mapping process starting from MII. If the mapping is not successful, we increase the II by one and attempt to map again. This process continues until we reach the maximum II (hardware-decided) or exceed the mapping time limit (lines 3-16).

As we mentioned earlier, Rewire takes the initial mapping from conventional approaches [22] (line 2) and focuses on amending the initial invalid mapping to achieve a valid mapping (lines 5-15). Given the invalid initial mapping, Rewire identifies nodes that are non-mapped or have congested routing as ill-mapped nodes. Rewire randomly selects several unmapped connected nodes to let them share the routing information, denoted as U (line 6). We need to limit the size of U. The reason is: (1) too many DFG nodes in U will significantly increase the search space and make it very time-consuming to find suitable placement candidates for U; (2) increasing the number of DFG nodes might not always work as the DFG is not mappable at current II (MII is not always achievable). In practice, we limit the size of U to 15. Given U, Rewire simultaneously propagates the source nodes, which are the parents and children of the cluster U (line 8). Then, Rewire summarizes the propagation information and uses the intersection to generate the placement candidates for each node $v \in U$ (line 9). Rewire uses data dependencies as constraints to prune invalid cluster placement candidates during building the placement of cluster U (line 10). The propagation and mapping generation will be detailed in Section IV-C and Section IV-D.

It is possible that the original U is not mappable due to the limited routing resource and complex data dependencies. Hence, we progressively add the connected node to U (line 13) until we map these nodes or reach the size limit of U (line 7). To append, we select the node that has the least depth-fist search (DFS) distance to the cluster U. Reaching the size limit means we cannot find a valid mapping with current II and need to increase II by one.

C. Routing Propagation

Rewire conducts forward propagation of values from the parents of U ($Parents(U)$) and backward propagation from the children of U ($Children(U)$) across the network. $Parents(U)$ is a set of DFG nodes that have been mapped and have a child node in U, but the node itself does not belong to U, denoted as $Parents(U) =$

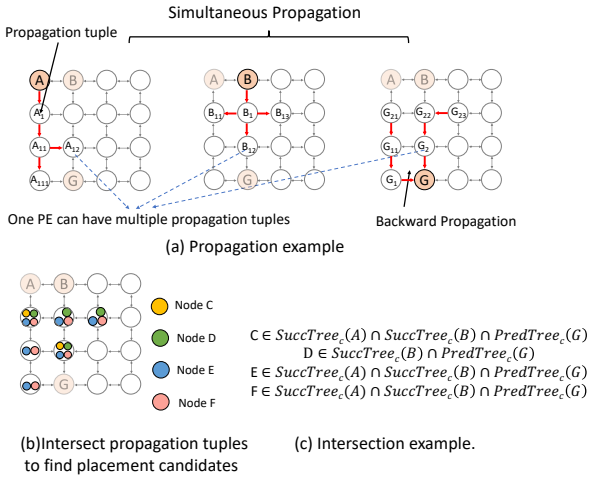


Fig. 4: Propagation and intersection example.

$v \in V \setminus U : \exists u \in U, (v,u) \in E$ and $\phi(v) \neq \emptyset$. Similarly, $Children(U) = v \in V \setminus U : \exists u \in U, (u,v) \in E$ and $\phi(v) \neq \emptyset$. Rewire propagates as long as the network path is not used by current mapping. Once reaching a PE, we will generate a propagation tuple. The propagation tuple contains four essential components: the source node, the propagation direction (forward/backward), the source PE, and the number of routing cycles. Thus, each tuple is associated with a specific PE in the network. The number of routing cycles is the cycle number from the source PE to the current PE.

The propagation tuple is a probe of network utilization that helps Rewire find suitable placement candidates. To keep the only necessary information, when a PE receives a propagation message from an adjacent PE, Rewire generates a new propagation tuple only if no existing tuple at that PE has the identical combination of source node, routing cycle count, and propagation direction. This mechanism prevents redundant exploration while covering all potential routing paths. To determine the number of propagation rounds, we heuristically use the maximum cycle difference between $Parents(U)$ and $Children(U)$ multiplied by three as the number of propagation rounds. When either $Parents(U)$ or $Children(U)$ is empty, we heuristically use the length of the longest path within U multiplied by five. This value allows the Rewire to generate sufficient propagation tuples.

Figure 4(a) provides an example of propagation for the motivating example in Section III. We note that we simultaneously propagate these sources nodes and we show them separately for simplicity. The IDs, like A_{12}, are to distinguish from each other and do not have any special meaning. The simultaneous propagation of multiple source nodes introduces resource contention in the network. Since the objective of propagation is to explore potential routing paths rather than perform final resource allocation, we continue propagation even when hardware resources have been traversed by other propagation tuples. These propagations generate a hierarchical structure where we define the successor tree $SuccTree(v)$ as the set of all propagation tuples originating from node v through forward propagation and the predecessor tree $PredTree(v)$ for backward propagation. We define $Tree(v)$ to represent all propagation tuples propagated from v.

D. Multi-node Mapping Generation

Propagation Information Intersection: After propagation, we first intersect propagation tuples to find placement candidates for DFG nodes in target cluster U. Now, each PE V_H virtually owns multiple propagation tuples, which are from different source nodes and arrive at different cycles. Figure 4(b) shows an example of intersecting these propagation tuples to find placement candidates for the unmapped DFG nodes in Figure 2(a). For a node v in U, PE V_H can be a placement candidate if it has the required propagation tuples for v. The requirement is, for each edge (v,u) (or (u,v)), or for each parent and child, we need to have a corresponding tuple, and these corresponding tuples must arrive at the same cycle to ensure the "right" execution of node v_i. If a parent or child node of v in U is not the source node of propagation, we use DFS to find a source node to represent this parent or child. For example, for the DFG in Figure 2(a), F is not a propagation source node, and we use DFS to find G as the propagation source node. In this example, three edges are connected to C; therefore, a PE can be a placement candidate of C only if the PE has all three corresponding tuples to ensure routing paths. In other words, we need to have three tuples originating from A, B, and G, and these tuples must arrive at the same cycle.

Hence, we formalize the intersection rule as below:

$$PCandidates(v_i) = \{V_H^F | \exists c \in \mathbb{N} : V_H^F \in (\bigcap_{v_j \in Parents(v_i)} SuccTree_c(source(v_j))) \cap (\bigcap_{v_k \in Children(v_i)} PredTree_c(source(v_k)))\} \quad (1)$$

where $SuccTree_c(v)$ represents the successor tree from node v at cycle c, $PredTree_c(v)$ represents the predecessor tree from node v at cycle c, and c is the common arrival cycle for all required tuples. For example, for node C, we can obtain the candidates with:

$$PCandidates(C) = \{V_H^F | \exists c \in \mathbb{N} : V_H^F \in (SuccTree_c(A) \cap SuccTree_c(B) \cap PredTree_c(G))\} \quad (2)$$

Figure 4(c) provides a few simplified examples using intersection to find the placement candidates for unmapped nodes. With the propagation and intersection example, we can find we only need several rounds of propagation to let these DFG nodes find potential placement candidates, even though the process has not yielded the final multi-node mapping yet. In contrast, conventional approaches usually need to invoke the path routing algorithm multiple times to check multiple candidates for only one DFG node placement.

Algorithm 2: Generate multi-node mapping

Input: DFG, CGRA, cluster U, $PCandidates(v_i)$ for each $v_i \in U$
Output: Mapping

1 Sort U by topological order $U = \{v_i | i = 0, ..., m\}$;
2 **foreach** $v_i \in U$ **do**
3 | Sort placement candidates by available execution cycles;
4 **while** *not exceed the time limitation* **do**
| // generate $Placement(U)$
5 | **foreach** v_i ($0 \leq i \leq m$) **do**
6 | | **foreach** v_j ($0 \leq j \leq i-1$) **do**
7 | | | Generate time constraints related to current node v_i;
8 | | select a candidate obeying execution cycle constraints;
9 | **if** *routing verified* **then**
10 | | return the $Placement(U)$;

Multi-node Mapping Generation: Algorithm 2 shows how to generate multi-node mapping. After the intersection of propagation tuples, we generate placement candidates $PCandidates(v_i)$ for each node $v_i \in U$, where placement candidates can execute at different cycles (decided by the arriving cycles of propagation tuples). We utilize the execution cycle information to filter invalid placement for U. We first sort nodes in cluster U by topological order (line 1) and sort placement candidates for each DFG node v_i by their execution cycles (lines 2 and 3). Then, we build placement candidates for U (represented as $Placement(U)$), which need to obey the execution order among nodes in U. For example, for the DFG in

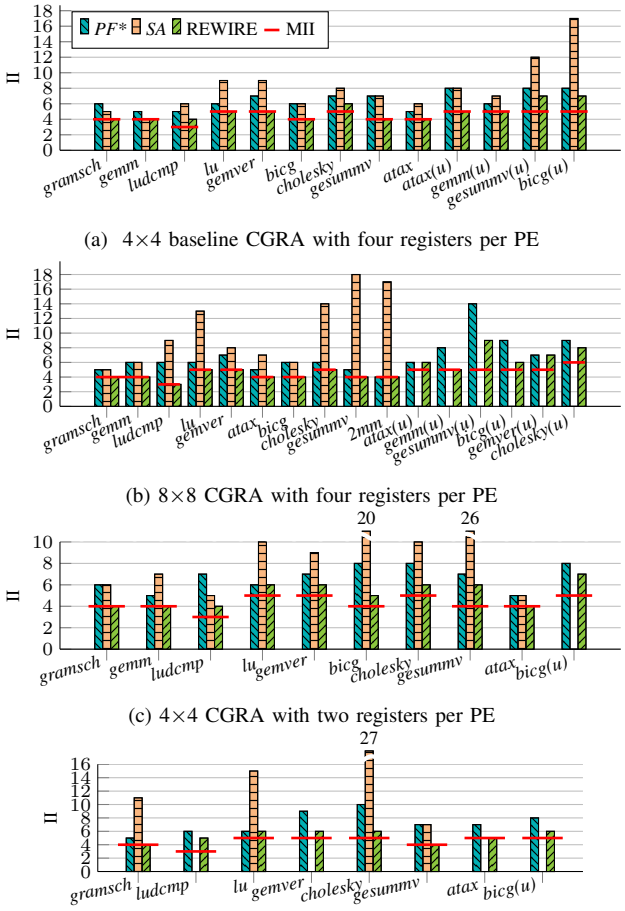
Figure 2(a), the execution cycle of E's placement candidate must be larger than the cycle of C's with respect to the data dependency. As $PCandidates(v_i)$ can have different execution cycles, we need to remove the invalid combinations during $Placement(U)$ generation. Hence, we iterate from v_0 to v_m to build $Placement(U)$ (lines 5-8) and check the execution cycle constraints (lines 6-8) during this process. Moreover, we use an index vector to track these placement candidates and iterate them to ensure we do not explore the same $Placement(U)$. When we add a placement candidate for v_i, we check the cycle execution constraints from v_0 to v_{i-1} if it has any data dependency with v_i. We find that this method is quite effective in pruning invalid $Placement(U)$.

As we mentioned in propagation, the propagation tuple may not always be valid, as we do not check resource contention among tuples during propagation. Hence, we must verify the $Placement(U)$ through routing (lines 9 and 10). If not successfully verified, we keep finding valid $Placement(U)$ (lines 4-10). After propagation, only part of PE can be the placement candidate of the DFG node, while the cycle execution constraints prune most invalid $Placement(U)$, meaning we only need to check a small portion of $Placement(U)$. Our evaluations show that generated $Placement(U)$ has a very high success rate of around 95% for verification. The reason is that the execution cycle constraints have successfully pruned many invalid $Placement(U)$, and the propagation mechanism has already performed a preliminary evaluation of routability.

V. Evaluation

In this section, we evaluate Rewire against two recent CGRA mapping techniques in terms of application performance (iteration interval II) and compilation time. We compare Rewire against two mapping approaches: SA and PF*. Simulated Annealing (SA) has been widely used in different CGRA compilation frameworks [10], [29], [32]. PathFinder is a popular algorithm used by several CGRA mappers [2], [22], [39], [40]. PathFinder and SA are two representative examples of the first type and the second type of mapper as discussed in Section II, respectively. We refer to our fine-tuned implementation of PathFinder with 3K lines of C++ code as PF*. The idea of PF* is to generate an initial mapping by selecting the placement with the minimal routing cost for the edges and then amend the mapping through multiple remapping iterations until a feasible solution is reached. We use the initial mapping of PF* as the initial mapping for Rewire, and then use our multi-node mapping to reach a feasible solution from the initial infeasible mapping.

We evaluate these mappers on the following CGRA architectures: (a) 4×4 CGRA with four registers per PE; (b) 8×8 CGRA with four registers per PE; (c) 4×4 CGRA with two registers per PE; (d) 4×4 CGRA with one register per PE. All the 4×4 CGRAs have two local memory banks, and PEs on the left-most column can access the memory. The 8×8 CGRA have eight local memory banks, and the left-most and right-most PE columns can access the memory, making 16 PEs can access the memory. Registers play a crucial role in buffering data during routing, particularly when the producer and consumer are distant nodes along spatial or temporal dimensions. We evaluate with different numbers of registers to evaluate Rewire's capability to handle different routing resources. 4×4 CGRA with one register per PE is not practical as it has very limited routing resources. Yet, we use it to evaluate the capability to handle extreme cases. We use the PolyBench [41], MachSuite [42] and MiBench [43] for our applications. We also used unrolled versions (unroll factor of 2) of some of these benchmarks to stress the compiler, specially on 8×8 CGRA. The number of DFG nodes varies from 26 to 51 and the average number of DFG nodes is 38. We measure the compilation time on the Intel Xeon Gold CPU (2.60GHz).



(a) 4×4 baseline CGRA with four registers per PE

(b) 8×8 CGRA with four registers per PE

(c) 4×4 CGRA with two registers per PE

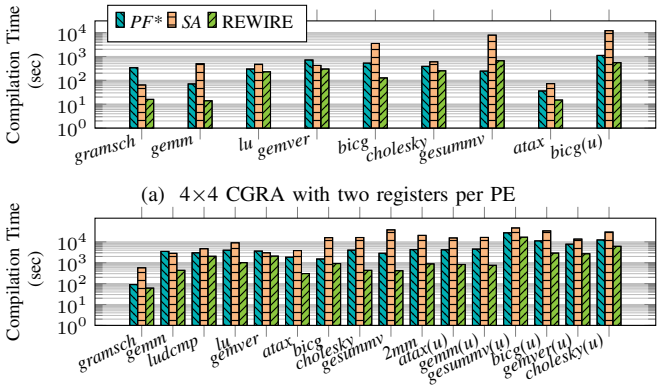
(d) 4×4 CGRA with one register per PE

Fig. 5: Comparison on the mapping quality of Rewire with PF* and SA on different benchmarks and CGRA configurations. The lower the II value, the higher the performance. *u* next to a benchmark represents the unrolled version with a factor 2. MII is the theoretically minimal II that may not be achievable in practice. Missing bars for SA for some benchmarks indicate that SA was not successful in mapping those benchmark-architecture configurations. We did not include a benchmark-architecture configuration if none of the three approaches can map it due to inadequate availability of resources, e.g., unrolled loops on architectures with few registers per PE.

A. Quality of Mapping

Figure 5 shows the comparison of the mapping quality for the different approaches (Rewire, PF*, and SA) in terms of the II values on four different CGRA configurations. Recall that II is the number of cycles between starting the execution of consecutive iterations of a loop kernel. The lower II value indicates lower execution time and hence higher performance. MII is the theoretically minimal II which is calculated by resource and inter-iteration data dependencies. However, achieving the MII may not always be possible as complex data dependency constraints are not considered in the theoretical calculation. Nevertheless, we consider a mapping optimal if it achieves the MII. If the difference between the II achieved and MII is one, we call the mapping near-optimal.

This evaluation uses 47 different DFG and architecture combinations. Rewire can achieve superior performance than PF* and SA if not the same on every application and CGRA combination. Both Rewire and PF* successfully map all the combinations, whereas SA



(a) 4×4 CGRA with two registers per PE

(b) 8×8 CGRA with four registers per PE

Fig. 6: Compilation time comparison of Rewire with PF* and SA on 4×4 CGRA with four registers per PE and with two registers per PE.

fails to map 12 combinations. Among the 47 combinations, Rewire delivers optimal or near-optimal mapping in 38 cases. In comparison, PF* and SA achieve optimal or near-optimal mapping for only 15 and 3 cases, respectively. Overall, Rewire delivers an average performance speedup of 1.3x and 2.1x compared to PF* and SA, respectively. The mapping quality difference shows Rewire's ability to handle complex data dependencies across application kernels on different CGRAs.

Rewire is also versatile and scalable to support various CGRA configurations. Rewire consistently achieves 1.33x, 1.31x, 1.29x, and 1.32x performance improvement compared to PF* on 4×4 baseline CGRA with four registers per PE, 4×4 CGRA with one register per PE, 4×4 CGRA with two registers per PE, and 8×8 CGRA with four registers per PE, respectively. This consistency underlines Rewire's ability to achieve superior performance across varied architectures. Moreover, compared to SA, Rewire can achieve 1.58x, 3.25x, 2.10x, and 2.21x performance improvement on these architectures, respectively. The most significant improvement comes from the 4×4 CGRA with one register. This architecture has very limited routing resources to evaluate the capability to handle extreme cases, as mentioned earlier, and a few DFGs can be mapped on the architecture. This improvement also highlights Rewire's capability to handle various architectures. The key factor behind the success of Rewire is its ability to coordinate the mapping of multiple DFG nodes, while PF* and SA make localized and myopic decision, making them struggle to handle intricate data dependencies.

Rewire achieves significant performance improvement from 4×4 CGRA with 4 registers per PE to 8×8 CGRA with 4 registers per PE. Rewire generates optimal mapping for *cholesky*, *ludcmp*, and *gesumv(u)* for the 8×8 CGRA, while fails on 4×4 CGRA to generate mappings due to insufficient hardware resources. This demonstrates the scalability of Rewire to handle larger search space by effectively pruning away infeasible candidates to avoid unnecessary searches.

B. Compilation Time

Figure 6 shows the compilation time comparison among these mappers on 8×8 CGRA with four registers per PE and 4×4 CGRA with two registers per PE. Note that the Y-axis is in log scale. Mappers can explore for a maximum of one hour per II, and can terminate early at each II due to the backtracking limitation (PF*) or no mapping cost improvement after 100 iterations (SA). As mentioned earlier, SA cannot map several benchmarks even after numerous iterations. In those cases, we choose the termination time as the compilation time. We note that the propagation time usually takes less than one second in Rewire, and the most time-consuming step happens at pruning during multi-node mapping generation.

TABLE I: Number of single-node remapping iterations for PF* and SA on 4×4 CGRA with four registers per PE and with one register per PE.

4×4 CGRA with 1 register per PE			4×4 CGRA with 4 registers per PE		
Benchmark	**PF***	**SA**	**Benchmark**	**PF***	**SA**
gramsch	242	1335	gramsch	323	1084
ludcmp	362	1422	ludcmp	315	1494
lu	233	1231	lu	233	1440
gemver	339	1428	gemver	293	1208
cholesky	332	1390	cholesky	332	1116
gesummv	935	1440	gesummv	228	1259
atax	232	1451	atax	242	920
bicg(u)	442	1425	bicg(u)	423	1341

On 4×4 CGRA with two registers per PE, Rewire achieves 4.2x and 13.5x compilation time reduction compared to PF* and SA, respectively. On 8×8 CGRA with four registers per PE, Rewire achieves 2.47x and 6.97x compilation time reduction compared to PF* and SA, respectively. Overall, Rewire achieves significant compilation time reduction compared to other approaches. The main reason is that Rewire maps multiple nodes in one go and does not need numerous iterations to map each node individually and backtrack (remap) if it fails to map future nodes. Moreover, the compilation time on 8×8 CGRA is significantly higher than 4×4 CGRA due to the larger search space, but Rewire can still achieve substantial compilation time reduction compared to the other two approaches, demonstrating the efficiency of the pruning algorithm.

Table I shows the number of iterations for PF* and SA on 4×4 CGRA with four registers per PE and 4×4 CGRA with one register per PE. In each iteration, both methodologies select one node to unmap. As the mappers need to generate a new initial mapping for each explored II value, we use the average number of remapping iterations from the start II to the final mapped II. Both PF* and SA need a significant number of single-node mapping iterations, while Rewire only needs to construct the right cluster of multiple nodes and amend their invalid mappings in one shot. SA needs much more remapping iterations than PF*. The reason is that PF* evaluates all the placement candidates for each single-node remapping and selects the best one, while SA selects one candidate randomly. Thus, PF* has a better chance of success. Moreover, PF* and SA generally need more remapping iterations on CGRA with one register per PE compared to CGRA with four registers per PE. The reason is that the former has fewer routing resources and needs more iterations to find a mapping. Rewire might need multiple iterations to build U as we progressively append the node to U. However, U cannot be bigger than the size of DFG, and we have limited the size to 15, as mentioned previously. Hence, Rewire takes significantly fewer iterations than both methodologies.

In summary, for almost all applications and CGRA combinations, Rewire substantially outperforms PF* and SA in mapping quality (application performance) and compilation time. With the consolidated routing paradigm, Rewire scales with CGRA size and complex DFGs and well handles architectures with limited routing resources.

VI. Conclusion

Existing mapping approaches place and route nodes individually, limiting the ability to comprehensively coordinate multi-node mapping. Rewire presents a consolidated routing paradigm for CGRA mapping that effectively handles complex multi-node dependencies. By mapping multiple nodes in one shot through propagation and consolidation of routing information, our approach achieves significant improvements in both performance and compilation time compared to two popular mappers.

REFERENCES

[1] Zhaoying Li et al. Coarse-grained reconfigurable array (cgra). *Handbook of Computer Architecture*, pages 1–41, 2022.

[2] Manupa Karunaratne et al. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *DAC*, 2017.

[3] Raghu Prabhakar et al. Plasticine: A reconfigurable architecture for parallel patterns. In *ISCA*, pages 389–402. IEEE, 2017.

[4] Dhananjaya Wijerathne et al. Cascade: High throughput data streaming via decoupled access-execute cgra. *TECS*, 2019.

[5] Yaqi Zhang et al. Scalable interconnects for reconfigurable spatial architectures. In *ISCA*. IEEE, 2019.

[6] Manupa Karunaratne et al. 4d-cgra: Introducing branch dimension to spatio-temporal application mapping on cgras. In *ICCAD*. IEEE, 2019.

[7] Ankita Nayak et al. A framework for adding low-overhead, fine-grained power domains to cgras. In *DATE*. IEEE, 2020.

[8] Michael Pellauer et al. Efficient control and communication paradigms for coarse-grained spatial architectures. *TOCS*, 2015.

[9] Rick Bahr et al. Creating an agile hardware design flow. In *DAC*. IEEE, 2020.

[10] Jian Weng et al. Dsagen: Synthesizing programmable spatial accelerators. In *ISCA*. IEEE, 2020.

[11] Tan Cheng et al. Aurora: Automated refinement of coarse-grained reconfigurable accelerators. In *DATE*. IEEE, 2021.

[12] Kaushalya Bandara Thilini et al. Revamp: A systematic framework for heterogeneous cgra realization. In *ASPLOS*, 2022.

[13] Tony Nowatzki et al. Stream-dataflow acceleration. In *ISCA*, 2017.

[14] Karthikeyan Sankaralingam et al. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. In *ISCA*, 2003.

[15] Graham Gobieski et al. Riptide: A programmable, energy-minimal dataflow compiler and architecture. In *MICRO*, 2022.

[16] Graham Gobieski et al. Snafu: an ultra-low-power, energy-minimal cgra-generation framework and architecture. In *ISCA*. IEEE, 2021.

[17] Yaqi Zhang et al. Sara: Scaling a reconfigurable dataflow accelerator. In *ISCA*. IEEE, 2021.

[18] Zhaoying Li et al. Enhancing cgra efficiency through aligned compute and communication provisioning. In *ASPLOS*, 2025.

[19] Miaomiao Jiang et al. Fhe-cgra: Enable efficient acceleration of fully homomorphic encryption on cgras. In *DAC*, 2024.

[20] Cheng Tan et al. Iced: An integrated cgra framework enabling dvfs-aware acceleration. In *MICRO*. IEEE, 2024.

[21] Zhaoying Li et al. Lisa: Graph neural network based portable mapping on spatial accelerators. In *HPCA*. IEEE, 2022.

[22] Stephen Friedman et al. Spr: an architecture-adaptive cgra mapping tool. In *FPGA*, 2009.

[23] Mahdi Hamzeh et al. Regimap: Register-aware application mapping on coarse-grained reconfigurable architectures (cgras). In *DAC*, 2013.

[24] Dhananjaya Wijerathne et al. Panorama: divide-and-conquer approach for mapping complex loop kernels on cgra. In *DAC*, 2022.

[25] Xingchen Man et al. Casmap: agile mapper for reconfigurable spatial architectures by automatically c lustering intermediate representations a nd s cattering mapping process. In *ISCA*, 2022.

[26] Mingyang Kou et al. Geml: Gnn-based efficient mapping method for large loop applications on cgra. In *DAC*, 2022.

[27] Shail Dave et al. Ramp: Resource-aware mapping for cgras. In *ISCA*, 2018.

[28] Hyunchul Park et al. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *PACT*, 2008.

[29] S Alexander Chin et al. Cgra-me: A unified framework for cgra modelling and exploration. In *ASAP*, pages 184–189. IEEE, 2017.

[30] Zhaoying Li et al. Chordmap: Automated mapping of streaming applications onto cgra. *TCAD*, 2021.

[31] Matthew JP Walker et al. Generic connectivity-based cgra mapping via integer linear programming. In *FCCM*. IEEE, 2019.

[32] Dhananjaya Wijerathne et al. Morpher: An open-source integrated compilation and simulation framework for cgra. In *WOSET*, 2022.

[33] Cheng Tan et al. Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras. In *ICCD*. IEEE, 2020.

[34] Yan Zhuang et al. Towards high-quality cgra mapping with graph neural networks and reinforcement learning. In *ICCAD*, 2022.

[35] Tony Nowatzki et al. A general constraint-centric scheduling framework for spatial architectures. *ACM SIGPLAN Notices*, 48, 2013.

[36] S Alexander Chin and Jason H Anderson. An architecture-agnostic integer linear programming approach to cgra mapping. In *DAC*, 2018.

[37] Bingfeng Mei et al. Dresc: A retargetable compiler for coarse-grained reconfigurable architectures. In *FPT*. IEEE, 2002.

[38] B Ramakrishna Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Micro*, pages 63–74, 1994.

[39] Frederick Tombs et al. Mocarabe: High-performance time-multiplexed overlays for fpgas. In *FCCM*, pages 115–123. IEEE, 2021.

[40] Su Zheng et al. Fastcgra: A modeling, evaluation, and exploration platform for large-scale coarse-grained reconfigurable arrays. In *ICFPT*. IEEE, 2021.

[41] Jeyhun Karimov, Tilmann Rabl, and Volker Markl. Polybench: The first benchmark for polystores. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 24–41. Springer, 2018.

[42] Brandon Reagen et al. Machsuite: Benchmarks for accelerator design and customized architectures. In *IISWC*. IEEE, 2014.

[43] Matthew R Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization.* IEEE, 2001.

Measurement-based uncomputation of quantum circuits for modular arithmetic

Alessandro Luongo*†, Antonio Michele Miti‡, Varun Narasimhachar§, Adithya Sireesh¶
*Center for Quantum Technologies (CQT), Singapore
†Inveriant Pte. Ltd., Singapore
‡Department of Mathematics, Sapienza Università di Roma, Rome, Italy
§Agency for Science, Technology and Research (A*STAR), Singapore
¶Quantum Software Lab, University of Edinburgh, United Kingdom

***Abstract*—Measurement-based uncomputation (MBU) is a technique used to perform probabilistic uncomputation of quantum circuits. We formalize this technique for the case of single-qubit registers, and we show applications to modular arithmetic. Using MBU, we reduce Toffoli count and depth by 10% to 15% for modular adders based on the architecture of [1], and by almost 25% for modular adders based on the architecture of [2]. Our results have the potential to improve other circuits for modular arithmetic, such as modular multiplication and modular exponentiation, and can find applications in quantum cryptanalysis.**

***Index Terms*—Modular arithmetic, Quantum computing, Circuit optimization, Qubit, Design Automation.**

I. Introduction

Given an invertible Boolean function $f : \{0,1\}^n \to \{0,1\}^m$, a quantum circuit implementation is given by a unitary gate U_f mapping $|x\rangle_n \mapsto |f(x)\rangle_m$. In general, computing f may require an out-of-place implementation, where U_f maps $|x\rangle_n |0\rangle_m$ to $|x\rangle_n |f(x)\rangle_m$, possibly introducing auxiliary qubits in a garbage state, $|g(x)\rangle_k$, which must be uncomputed to preserve reversibility. As it turns out, out-of-place implementations with garbage often arise in binary arithmetic. For instance, a *ripple carry* addition algorithm computes the carry bitwise from the least significant bits of the addends, leaving the carry in memory. To ensure correctness, the ancillary carry bit must be uncomputed from the registers. Other operations, such as (modular) addition, subtraction, and comparison by a constant, can be built from these adder gates. Efficient quantum arithmetic circuits are pivotal for advancing quantum computing, underpinning numerous key algorithms. Effective implementations of binary arithmetic are fundamental to many quantum algorithms, such as quantum algorithms for factoring [3], [4] and discrete logarithm [5], quantum walks [6], and for algorithms that construct oracles for Grover's search and quantum phase estimation, e.g. [7]–[9]. Therefore, finding cheaper ways of performing uncomputation on qubits storing garbage/intermediate computations is essential for efficient quantum circuit design.

a) Contributions: In this work, we enhance quantum circuits for modular arithmetic, specifically focusing on modular addition two n-bit quantum registers. Our approach has the potential to improve not only modular multiplication and modular exponentiation but also quantum circuits used in cryptographic attacks, see for example [1], [10], [11]. In Section II we formalize the measurement-based uncomputation technique introduced in [12]–[14] providing with Lemma 1 a plug-and-play result that can be applied to any single-qubit garbage. In Section III, we introduce a modular and composable framework to construct arithmetic circuits from other known arithmetic primitives. In particular, we quantify the resource cost for some previously known adder implementation circuits for addition, namely [1], [12], [15], [16]. We also discuss how to compute subtraction, comparison, and modular addition together with their controlled and constant variants. Section IV is dedicated to deriving formal statements for modular addition circuits optimized by MBU. Notably, table V provides resource counts for modular addition, and highlights the cost reductions achieved by using MBU. For the controlled modular adder derived from [15], MBU can reduce the Tof (Toffoli) count by $n + \frac{1}{2}$ gates in expectation. For modular addition by a constant, MBU reduces the Tof count by n gates in expectation (see theorem 4), leading to a 16.7% improvement on the original circuit. We obtain similar improvements for controlled modular addition by a constant. Finally, to our knowledge, we are the first to discuss a quantum circuit to compute whether a given register's value is contained within two other register's values (theorem 6). This circuit is a prime candidate for MBU, leading to a nearly 16.7% reduction in the cost of the circuit.

b) Notation and convention: As in [15], we denote the quantum registers by capital letters A, B, C, etc. Each register is encoded with several qubits; we denote by A_i the i-th qubit of register A. Where necessary, we put the number of qubits in a register as a subscript following the ket representing the register's state, e.g. $|x\rangle_n$ for an n-qubit register. For an n bit string $\mathbf{x} = x_{n-1}x_{n-2}\dots x_0 \in \{0,1\}^n$ we shall often equate it with the decimal number x encoded in binary by $\mathbf{x}$. We denote with $a|_n$ the n bit string encoding the number $a \in \{-2^n, \dots, 0, 1, \dots, 2^{n-1}-1\}$ in the 2's complement notation. For a number $a \in \mathbb{N}$, $|a|$ denotes the Hamming weight of the binary representation of a. We define the function $\mathbf{1}[\mathsf{condition}]$ to be 1 if $\mathsf{condition}$ is true, otherwise is 0. We call *bit string addition* the binary operator $+ : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+1}$, $(\mathbf{x}, \mathbf{y}) \mapsto \mathbf{s}$ defined bit-wise by $s_i = x_i \oplus y_i \oplus c_i$, where the bit c_i, called i-th carry, is defined recursively by $c_0 = 0$ and $c_{i+1} = \mathrm{maj}(x_i, y_i, c_i)$. The function maj is a logic gate called *majority* defined by

$$\mathrm{maj}(x_i, y_i, c_i) = x_i y_i \oplus x_i c_i \oplus y_i c_i \ , \tag{1}$$

which gives 1 whenever at least half of the inputs (at least two out of three) entries are different 1. We call 1*'s complement* the operator $\overline{\cdot} : \{0,1\}^n \to \{0,1\}^n$, $\mathbf{x} \mapsto \overline{\mathbf{x}}$ defined bit-wise as $(\overline{\mathbf{x}})_i = x_i \oplus 1$. We call 2*'s complement* the operator defined as $\overline{\overline{\mathbf{x}}} = \overline{\mathbf{x}} + \mathbf{1}$, where $\mathbf{1}$ denotes the n bits string $0 \dots 0\,1$. We call *bit string subtraction* the binary operator $- : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+1}$, $(\mathbf{x}, \mathbf{y}) \mapsto \overline{(\overline{\mathbf{x}} + \mathbf{y})}$. The string $\mathbf{d} = \mathbf{x} - \mathbf{y}$ is given bit-wise by $d_i = x_i \oplus y_i \oplus b_i$, where the bit b_i, called i-th borrow, is defined recursively by $b_0 = 0$ and $b_{i+1} = \mathrm{maj}(x_i \oplus 1, y_i, b_i)$. Let Q be a quantum circuit composed of m gates Q_j from a gateset $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_K\}$. Define the gate count $c_i := |\{j \in [m] \mid \mathsf{Q}_j = \mathcal{G}_i\}|$ for $i \in [K]$. We represent the gate counts of Q as $\mathsf{count}(\mathsf{Q}) = \{(\mathcal{G}_i, c_i) \mid i \in [K]\}$. According

Corresponding author: asireesh@ed.ac.uk

to [17], a Quantum fourier transform QFT_m has a circuit size of $\mathsf{count}(\mathsf{QFT}_m) = \{(\mathsf{H}, m)\} \cup \{(\mathsf{C\text{-}R}(\theta_i), m+1-i) \mid i \in [1, m+1]\}$.

II. Ancillary garbage uncomputation

In this section, we dive deeper into the importance of uncomputation. Consider a quantum circuit U_f computing a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ between bit strings. When f is invertible (meaning that f is injective and $n = m$) we say that U_f computes f *in-place* if it performs the mapping

$$U_f : |x\rangle_n \mapsto |f(x)\rangle_m \tag{2}$$

for any bit string $x \in \{0,1\}^n$, where we use the subscripts to denote the number of qubits in a quantum register. An in-place implementation is not always possible. In the most general case, one needs to compute f *out-of-place*, meaning that the circuit U_f realises the following mapping

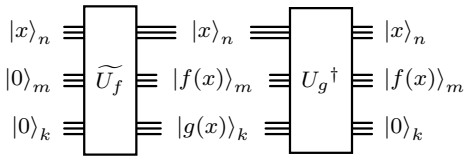
$$U_f : |x\rangle_n |0\rangle_m \mapsto |x\rangle_n |f(x)\rangle_m \ . \tag{3}$$

However, a more realistic implementation of the above gate will, in principle, involve a number k of auxiliary qubits which get mapped in a *garbage* state $|g(x)\rangle$ as follows:

$$\widetilde{U}_f : |x\rangle_n |0\rangle_m |0\rangle_k \mapsto |x\rangle_n |f(x)\rangle_m |g(x)\rangle_k . \tag{4}$$

The auxiliary qubits are entangled with the other registers and must be uncomputed, i.e. restored to their original value, in order to get a reversible out-of-place implementation of f. Such uncomputation can be achieved in several ways. A first solution, due to [18], simply prescribes copying the output $|f(x)\rangle_m$ into an auxiliary m-qubit register via a set of m CNOT gates and then applying $\widetilde{U}_f^\dagger$. This strategy doubles the resource cost of the reversible computation of f, also requiring m more ancillary qubits. Alternatively, one could look for an out-of-place implementation of the garbage function U_g, seeing g as a function of x and $f(x)$. When this unitary is available, one can implement U_f as in figure 1.

Fig. 1: Garbage gate uncomputation.



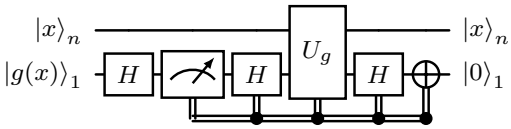
In principle, invoking the garbage circuit U_g can be quite resource intensive. Therefore, it is necessary to develop techniques to optimize the synthesis of quantum circuits and their uncomputation.

A. Measurement-Based Uncomputation (MBU).

Measurement-based uncomputation (MBU) was developed to efficiently address the challenge of uncomputing Boolean functions. This technique performs uncomputation probabilistically, by measuring the qubits slated for uncomputation on the X basis. Depending on the measurement outcome, the uncomputation circuit may either be executed or considered already completed, which helps to minimize overall quantum resource requirements. MBU has already been applied in various contexts to improve resource efficiency in uncomputation [12]–[14], [19]. However, to our knowledge, ours is the first formalization of this technique, enabling a clearer understanding and easy application in diverse contexts, as we show in Section III.

Lemma 1 (Measurement-based uncomputation lemma)**.** *Let $g : \{0,1\}^n \to \{0,1\}$ be a function, and let U_g be a self-adjoint $(n+1)$-qubit unitary that maps $U_g |x\rangle_n |b\rangle_1 = |x\rangle_n |b \oplus g(x)\rangle_1$.*

Fig. 2: A circuit for measurement-based uncomputing the unitary U_g.



There is an algorithm performing the mapping $\sum_x \alpha_x |x\rangle_n |g(x)\rangle_1 \mapsto \sum_x \alpha_x |x\rangle_n |0\rangle_1$ using U_g, 2 H gates, and 1 NOT gate with $1/2$ probability. It also involves one single-qubit computational basis measurement and a single H gate.

Proof. The algorithm is described in figure 2. We first apply the Hadamard gate on the register G and measure it in the computational basis. Since $g(x) \in \{0,1\}$, the resulting outcome is 0 or 1 with equal probabilities. If we obtain 0, we successfully uncomputed the register, leaving the registers in the state $\sum_x \alpha_x |x\rangle_A |0\rangle_G$. On the other hand, if we get the outcome 1, the state at this point is $\sum_x \alpha_x (-1)^{g(x)} |x\rangle_A |1\rangle_G$. We then apply another Hadamard to G, followed by U_g. This performs a phase kickback, leaving the quantum computer in the state

$$\sum_x \alpha_x (-1)^{g(x)} |x\rangle_A \frac{|0 \oplus g(x)\rangle_G - |1 \oplus g(x)\rangle_G}{\sqrt{2}} = \sum_x \alpha_x (-1)^{g(x) \oplus g(x)} |x\rangle_A |-\rangle_G \ .$$

Applying on G another Hadamard and a NOT gate results in the final state $\sum_x \alpha_x |x\rangle_A |0\rangle_G$. □

Applying the MBU lemma leads to gate costs expressed *"in expectation"*. The expected value is calculated based on a Bernoulli distribution with a probability $p = \frac{1}{2}$. This is because, for any quantum state $\sum_x \alpha_x |x\rangle_n |g(x)\rangle_1$, if $g(x) \in \{0,1\}$, when the last register is measured in the X basis, there is an equal probability of obtaining a $|0\rangle$ or $|1\rangle$ state.

III. Quantum Arithmetic

Early proposals for quantum addition [1], [15], [20], some of which are employed in fault-tolerant algorithms with state-of-the-art resource estimates [4], [21], [22], were inspired by classical circuits for addition, with an emphasis on minimizing the T count and/or the Tof (Toffoli) count, and the number of ancilla qubits. Subsequent approaches have explored more quantum-specific methods of performing arithmetic [2], [16], [23], [24]. All these arithmetic circuits serve as building blocks for more complex operations such as modular addition, multiplication, and exponentiation, to name a few. A comprehensive survey on quantum adders can be found in [11], [25].

We define a *(constant[1], controlled) quantum adder* as any unitary gate that implements the addition of n-bit strings according to circuits in figures 3, 4, and 5. Note that the initial extra qubit, prepared in $|0\rangle$, in the second register is needed to take into account the possible overflow. The circuit can perform unsigned integer addition by interpreting the bit string x as its corresponding decimal number, as well as signed integer addition, using the 2's complement representation. Table I summarizes the gate and qubit counts for various quantum implementations of bit string addition from the literature.

[1]By constant we mean a classically known value employed in the quantum computation.

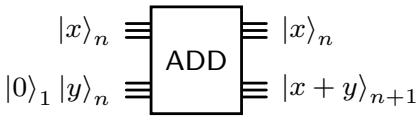


Fig. 3: (Plain) adder.

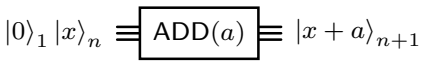


Fig. 4: Constant adder.

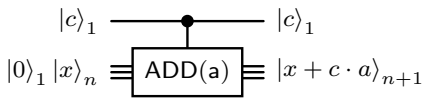


Fig. 5: Controlled constant adder.

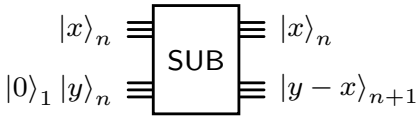


Fig. 6: (Plain) subtractor.

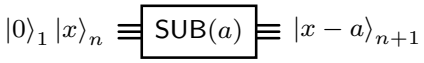


Fig. 7: Constant subtractor

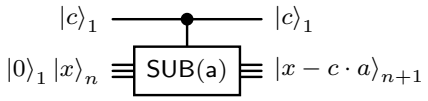


Fig. 8: Controlled constant subtractor.

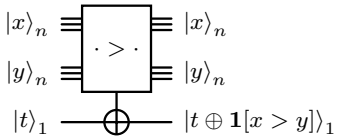


Fig. 9: Comparator.

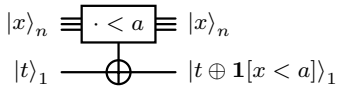


Fig. 10: Constant comparator.

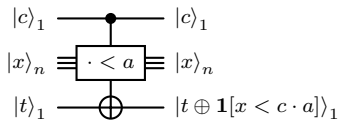


Fig. 11: Controlled constant comparator.

Procedure	Toff. Count	Ancillas	CNOT
VBE plain adder [1]	$4n$	n	$4n+4$
CDKPM plain adder [15]	$2n$	1	$4n+1$
Gidney plain adder [12]	n	n	$6n-1$
	QFT_{n+1}	**Ancillas**	
Draper [16]	3	0	–

TABLE I: Plain adders, gate count.

One can easily construct a controlled quantum adder circuit using an existing n-bit quantum adder. The construction requires $n+s$ ancilla qubits and $r+2n$ Toffoli gates, where s and r are the number of ancilla and Toffoli gates used by the original adder. It involves loading the value $c \cdot x$ in a register initially in the $|0\rangle_n$ state, performing plain addition, and unloading the control using a sequence of Toffoli gates. An improvement can be achieved by implementing Gidney's adder [12] by loading the value $c \cdot x$ via temporary logical-AND and then uncomputing it using measurements. Alternatively, the [15] adder implementation can be easily modified to perform controlled addition with a single ancilla qubit. To do so, it is sufficient to substitute a suitable CNOT in the "unmajority" gate with a Tof gate on the control register. Regarding the Draper implementation [20], circuit symmetries can be exploited to avoid controls on all gates. Only the central adder gate would need to be controlled, whereas the quantum Fourier transforms at the beginning and the end of the circuit remain uncontrolled. In total, this would require a single ancilla and n extra Tof gates. See table II for the gate count of some well-known implementations.

Procedure	Tof Count	Ancillas	CNOT
CDKPM	$3n$	1	$4n+1$
Gidney	$2n$	$n+1$	$7n-1$
	Tof Count	**Ancillas**	QFT_{n+1}
Draper	n	1	3

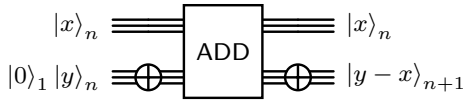
TABLE II: Controlled addition, gate count.

As suggested in [15], one can implement a quantum circuit that adds a classically known constant value a to an n-bit quantum register using any given adder circuit by loading the constant a (via $|a|$ NOT gates) on an ancillary register, performing the addition and then unloading a using the same NOT gates. In the case of Draper's QFT adder, Beauregard [2] introduced a method to add constants without requiring ancillas. Relying on the procedure sketched above, one can obtain a constant controlled adder starting with any plain adder implementation, and using additional $2|a|$ CNOT gates to conditionally Load/Unload the constant value. In [2], they propose another method based on Draper's implementation, which adds controls to rotation gates and achieves the same result without any ancilla qubits, with the circuit's complexity depending on the binary representation of the constant a. In table III we record the gate count for the above implementations.

Procedure	Tof Count	Ancillas	CNOT
CDKPM	$2n$	$n+1$	$4n+1+2\|a\|$
Gidney	n	$2n$	$6n-1+2\|a\|$
	QFT_{n+1}	**Ancillas**	$\mathsf{C\text{-}\Phi_{ADD}}(a)$
Draper	2	0	1

TABLE III: Controlled addition by a constant 'a', gate count.

We define a *(controlled) constant quantum subtractor* as any unitary gate that implements the subtraction of the n strings according to the circuits in figures 6,7, and 8. Due to the unitarity of the quantum adder gate and the uniqueness of the adjoint, it follows that $(\mathsf{ADD})^\dagger$ implements a quantum subtractor. Alternatively, one can implement a quantum subtractor directly from the definition of bit-string subtraction as follows.



In principle, a subtractor has the same count of ancillas, Tof gates, and CNOT gates as the corresponding adder implementations. We define a *(controlled, constant) quantum comparator* as any unitary gate that implements the n string comparison according to the circuits in figures 9,10, and 11. Note that $\mathbf{1}[\mathsf{x} > \mathsf{y}]$ coincides with the most

significant bit of $y - x$, i.e. it is 1 when $x > y$ and 0 otherwise. Given a quantum subtractor, one can easily obtain a string comparator reading the sign (i.e. the most significant bit) of the difference. In table IV we record the gate count for comparators with respect to the considered adder implementations. Given any circuit implementing

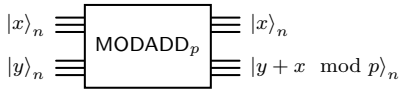
Procedure	Tof Count	Ancillas	CNOT
CDKPM	$2n$	1	$4n+1$
Gidney	n	n	$6n+1$
	QFT_{n+1}	**Ancillas**	–
Draper	6	1	–

TABLE IV: Comparator, gate count.

the comparison $(x > y)$ one can get the opposite comparison $(x \leq y)$ by simply postcomposing the first circuit with an X gate acting on the target register.

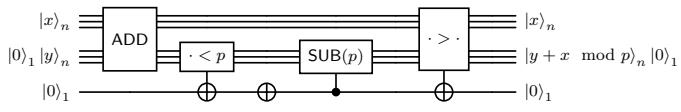
A. Quantum modular addition

Let $x \in \{0,1\}^n, y \in \{0,1\}^n$ be two bit strings, and $p \in \{0,1\}^n$ classically known with $0 \leq x, y < p$. We define a *quantum modular adder* as any unitary gate implementing the n-bit strings *addition modulo* p according to the following circuit:



A modular adder performing the operation, $|x\rangle|y\rangle \mapsto |x\rangle|x+y \pmod p\rangle$, can be decomposed into four separate subroutines [1]: quantum-quantum addition, comparison with a classical constant (by first loading in the constant using a set of X gates into an empty register, using two half subtractors [15] for the comparison, and then unloading the constant), controlled subtraction by a classical constant (by first loading the constant using a set of CNOTs into an empty register, then performing the subtraction using an inverse adder, and finally unloading the constant) and quantum-quantum comparison (using two half subtractor). The following implementation is a slight modification of [1] and [21]. It uses any available combination of adders, (controlled) subtractors (by a constant), and comparators (by a constant).

Fig. 12: Implementation of a modular adder using previous circuits.



The cost estimates for different specific combinations of the required building blocks are subsumed in table V.

IV. Measurement-based uncomputation speed-ups

In this section, we demonstrate the versatility of the MBU lemma in improving modular arithmetic circuits. While the technique is applicable to a wide range of modular arithmetic operations, we focus on a subset of key results to illustrate its impact. Specifically, we highlight their application to modular addition, controlled modular addition, modular addition by a constant, and controlled modular addition by a constant. For instance, in the modular adder architecture of VBE discussed in the previous section, the first three subroutines compute the desired value $|x+y \mod p\rangle$ in the output register, but leave an entangled ancilla qubit. The quantum-quantum comparison is required for the uncomputation of this single ancilla storing the most significant qubit of the adder circuit. Thus, the MBU Lemma can be applied to probabilistically uncompute this garbage qubit, reducing overall costs.

The theorems (thms. 1, 2, 3, 4, 5) detailed in this section represent abstract architectures of modular addition circuits which we optimize with the MBU lemma. Each subroutine in these theorems can be replaced by a specific implementation of a plain adder architecture of one's choice. In table V, we present the savings achieved by the MBU optimized modular adder, using specific plain adder architectures ([1], [12], [15], [16]) for the subroutines. Using our technique, we can derive similar results (e.g. between 15% to 25% reduction in gate count) for modular addition by a constant, controlled modular addition, and controlled modular addition by a constant.

Theorem 1 (MBU modular adder - VBE architecture)**.** *Let* $\mathsf{Q}_{\mathsf{ADD}}$ *be a quantum circuit that performs a quantum addition using* s_{ADD} *ancillas, and* r_{ADD} Tof *gates;* $\mathsf{Q}_{\mathsf{COMP}}(p)$ *be a quantum comparator using* s_{COMP} *ancillas,* r_{COMP} Tof *gates;* $\mathsf{C\text{-}Q}_{\mathsf{SUB}}(p)$ *be a quantum circuit that performs a (controlled) subtraction by a constant* p *using* $s_{\mathsf{C\text{-}SUB}}$ *ancillas, and* $r_{\mathsf{C\text{-}SUB}}$ Tof *gates;* $\mathsf{Q}'_{\mathsf{COMP}}$*: be a quantum comparator using (*s'_{COMP} *ancillas,* r'_{COMP} Tof *gates). There is a circuit,* $\mathsf{Q}_{\mathsf{MODADD}_p}$*, performing the* n*-bit modular addition operation using* $r = (r_{\mathsf{ADD}} + r_{\mathsf{COMP}} + r_{\mathsf{C\text{-}SUB}} + (r'_{\mathsf{COMP}}/2))$ Tof *gates in expectation and* s *ancillas with:*

$$s = 2 + \max(s_{\mathsf{ADD}}, s_{\mathsf{COMP}}, s_{\mathsf{C\text{-}SUB}}, s'_{\mathsf{COMP}}) \ .$$

Proof. The first three subroutines map $|0\rangle_s |x\rangle_n |y\rangle_n$ to

$$|0\rangle_{s-1} |x\rangle_n |x+y \mod p\rangle_n |\mathbf{1}[x+y \geq p]\rangle_1 \ .$$

The comparator qubit storing $\mathbf{1}[x+y \geq p]$ is finally supposed to be uncomputed using $\mathsf{Q}'_{\mathsf{COMP}}$. We can apply the MBU lemma on the comparator qubit and the uncomputation subroutine $\mathsf{Q}'_{\mathsf{COMP}}$, thus halving the cost of $\mathsf{Q}'_{\mathsf{COMP}}$ in expectation. Therefore, we finally get a cost of $r = (r_{\mathsf{ADD}} + r_{\mathsf{COMP}} + r_{\mathsf{C\text{-}SUB}} + (r'_{\mathsf{COMP}}/2))$ Tof gates in expectation, and the same ancilla count of the modular addition implementation. □

By suitably substituting certain adder gates with the corresponding controlled version, one obtains a routine for controlled modular arithmetic.

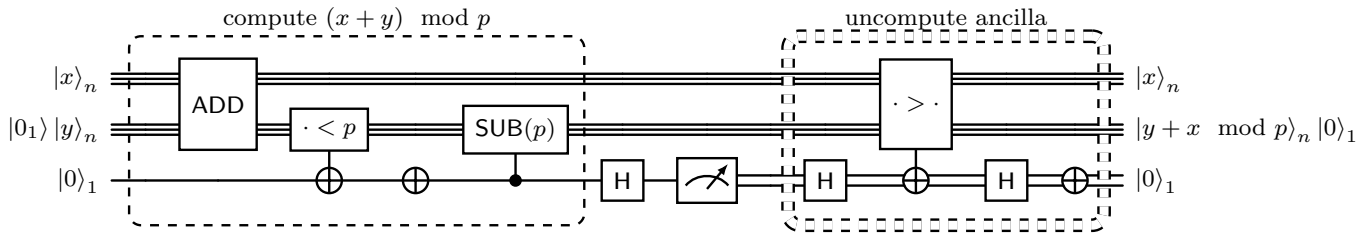
Theorem 2 (MBU controlled modular adder - VBE architecture)**.** *Let* $\mathsf{C\text{-}Q}_{\mathsf{ADD}}$ *be a quantum circuit that performs a quantum (controlled) addition using* $s_{\mathsf{C\text{-}ADD}}$ *ancillas, and* $r_{\mathsf{C\text{-}ADD}}$ Tof *gates;* $\mathsf{Q}_{\mathsf{COMP}}(p)$ *be a quantum comparator by a constant* p*, using* s_{COMP} *ancillas,* r_{COMP} Tof *gates;* $\mathsf{C\text{-}Q}_{\mathsf{SUB}}(p)$ *be a quantum circuit that performs a quantum subtraction by a constant* p*, using* $s_{\mathsf{C\text{-}SUB}}$ *ancillas, and* $r_{\mathsf{C\text{-}SUB}}$ Tof *gates;* $\mathsf{C\text{-}Q}'_{\mathsf{COMP}}$*: be a controlled quantum comparator using* $s'_{\mathsf{C\text{-}COMP}}$ *ancillas and* $r'_{\mathsf{C\text{-}COMP}}$ Tof *gates. There exists a circuit* $\mathsf{C\text{-}Q}_{\mathsf{MODADD_p}}$ *that implements an* n*-bit controlled quantum modular addition using* $r = (r_{\mathsf{C\text{-}ADD}} + r_{\mathsf{COMP}} + r_{\mathsf{C\text{-}SUB}} + (r'_{\mathsf{C\text{-}COMP}}/2))$ Tof *gates in expectation and* $s = 2 + \max(s_{\mathsf{C\text{-}ADD}}, s_{\mathsf{COMP}}, s_{\mathsf{C\text{-}SUB}}, s'_{\mathsf{C\text{-}COMP}})$ *ancillas.*

Proof. The first three subroutines in figure 13 map $|0\rangle_s |c\rangle_1 |x\rangle_n |y\rangle_n$ to

$$|0\rangle_{s-1} |c\rangle_1 |x\rangle_n |c \cdot x + y \mod p\rangle_n |\mathbf{1}[c \cdot x + y \geq p]\rangle_1 \ .$$

The comparator qubit storing $\mathbf{1}[c \cdot x + y \geq p]$ is finally supposed to be uncomputed using $\mathsf{C\text{-}Q}'_{\mathsf{COMP}}$. We can apply the MBU lemma here on the comparator qubit and the uncomputation subroutine $\mathsf{C\text{-}Q}'_{\mathsf{COMP}}$, thus

Fig. 13: Implementation of a MBU modular addition. The double dotted lines represent the non-computation of the ancilla, controlled on the measurement value.



halving the cost of C-$\mathsf{Q}'_{\mathsf{COMP}}$ in expectation. Therefore, we finally get a cost of $r = (r_{\text{C-ADD}} + r_{\mathsf{COMP}} + r_{\text{C-SUB}} + (r'_{\text{C-COMP}}/2))$ Tof gates in expectation, and the same ancilla count as the controlled modular addition implementation. □

Similarly, one can get the circuit implementation of modular addition with a constant.

Theorem 3 (MBU modular addition by a constant - VBE architecture)**.** *Let* $\mathsf{Q}_{\mathsf{ADD}}(\mathsf{a})$ *be a quantum circuit that performs a constant quantum addition using* s_{ADD} *ancillas, and* r_{ADD} Tof *gates;* $\mathsf{Q}_{\mathsf{COMP}}(p)$ *be a quantum comparator using* s_{COMP} *ancillas,* r_{COMP} Tof *gates;* C-$\mathsf{Q}_{\mathsf{SUB}}(p)$ *be a quantum circuit that performs a (controlled) subtraction by a constant* p *using* $s_{\text{C-SUB}}$ *ancillas, and* $r_{\text{C-SUB}}$ Tof *gates;* $\mathsf{Q}'_{\mathsf{COMP}}(a)$*: be a constant quantum comparator using (*s'_{COMP} *ancillas,* r'_{COMP} Tof *gates). There exists a circuit* $\mathsf{Q}_{\mathsf{MODADD_p}}(a)$ *that implements an* n*-bit quantum modular addition by a constant using* $r = (r_{\mathsf{ADD}} + r_{\mathsf{COMP}} + r_{\text{C-SUB}} + (r'_{\mathsf{COMP}}/2))$ Tof *gates and* $s = 2 + \max(s_{\mathsf{ADD}}, s_{\mathsf{COMP}}, s_{\text{C-SUB}}, s'_{\mathsf{COMP}})$ *ancillas.*

Proof. The first three subroutines map $|0\rangle_s\,|x\rangle_n$ to

$$|0\rangle_{s-1}\,|x\rangle_n\,|x+a \mod p\rangle_n\,|\mathbf{1}[x+a \geq p]\rangle_1 \ .$$

The comparator qubit storing $|\mathbf{1}[x+a \geq p]\rangle_1$ is finally supposed to be uncomputed using $\mathsf{Q}'_{\mathsf{COMP}}(a)$. We can apply the MBU lemma here on the comparator qubit and the uncomputation subroutine $\mathsf{Q}'_{\mathsf{COMP}}(a)$, thus halving the cost of $\mathsf{Q}'_{\mathsf{COMP}}(a)$ in expectation. Therefore, we finally get a cost of $r = (r_{\mathsf{ADD}} + r_{\mathsf{COMP}} + r_{\text{C-SUB}} + (r'_{\mathsf{COMP}}/2))$ Tof gates in expectation, and the same ancilla count as the constant modular addition implementation. □

Theorem 4 (MBU modular adder by a constant - in Takahashi Architecture [26])**.** *Let* $x \in \{0,1\}^n$*. Assume to know classically* $p, a \in \{0,1\}^n$ *with* $0 \leq a, x < p$*. Also, assume we have access to the following subroutines:* $\mathsf{Q}_{\mathsf{SUB}}(\mathsf{a})$ *be a quantum circuit that performs a constant quantum subtraction using* s_{SUB} *ancillas, and* r_{SUB} Tof *gates;* C-$\mathsf{Q}_{\mathsf{ADD}}(p)$ *be a quantum circuit that performs a (controlled) addition by a constant* p *using* $s_{\text{C-ADD}}$ *ancillas, and* $r_{\text{C-ADD}}$ Tof *gates;* $\mathsf{Q}_{\mathsf{COMP}}(a)$ *be a classical comparator using* s_{COMP} *ancillas,* r_{COMP} Tof *gates. Then, there exists a circuit* $\mathsf{Q}_{\mathsf{MODADD_p}}(a)$ *that implements quantum modular addition by a constant using* $r = (r_{\mathsf{SUB}} + r_{\text{C-ADD}} + (r_{\mathsf{COMP}}/2))$ Tof *gates in expectation and* $s = 1 + \max(s_{\mathsf{SUB}}, s_{\text{C-ADD}}, s_{\mathsf{COMP}})$ *ancillas.*

Proof. As seen in the Takashi constant modular adder [26], the first two subroutines map $|0\rangle_s\,|x\rangle_n$ to

$$|0\rangle_{s-1}\,|x\rangle_n\,|x+a \mod p\rangle_n\,|\mathbf{1}[x+a < p]\rangle_1 \ .$$

The comparator qubit storing $|\mathbf{1}[x+a < p]\rangle_1$ is finally supposed to be uncomputed using $\mathsf{Q}_{\mathsf{COMP}}(a)$ and a NOT gate. We can apply the MBU lemma here on the comparator qubit and the uncomputation subroutine $\mathsf{Q}_{\mathsf{COMP}}(a)$, thus halving the cost of $\mathsf{Q}_{\mathsf{COMP}}(a)$ in expectation. Therefore, we finally get a cost of $r = (r_{\mathsf{SUB}} + r_{\text{C-ADD}} + (r_{\mathsf{COMP}}/2))$ Tof gates in expectation, and the same ancilla count of the [26] constant modular adder. □

Theorem 5 (MBU controlled modular adder by a constant - VBE architecture)**.** *Let* C-$\mathsf{Q}_{\mathsf{ADD}}(\mathsf{a})$ *be a quantum circuit that performs a controlled addition by a constant using* $s_{\text{C-ADD}}$ *ancillas, and* $r_{\text{C-ADD}}$ Tof *gates;* $\mathsf{Q}_{\mathsf{COMP}}(p)$ *be a quantum comparator using* s_{COMP} *ancillas,* r_{COMP} Tof *gates;* C-$\mathsf{Q}_{\mathsf{SUB}}(p)$ *be a quantum circuit that performs a (controlled) subtraction by a constant* p *using* $s_{\text{C-SUB}}$ *ancillas, and* $r_{\text{C-SUB}}$ Tof *gates;* C-$\mathsf{Q}'_{\mathsf{COMP}}(a)$*: be a controlled constant quantum comparator using (*$s'_{\text{C-COMP}}$ *ancillas,* $r'_{\text{C-COMP}}$ Tof *gates). There exists a circuit* C-$\mathsf{Q}_{\mathsf{MODADD_p}}(a)$ *that implements an* n*-bit controlled quantum modular addition by a constant using* $r = (r_{\text{C-ADD}} + r_{\mathsf{COMP}} + r_{\text{C-SUB}} + (r'_{\text{C-COMP}}/2))$ Tof *gates and* $s = 2 + \max(s_{\text{C-ADD}}, s_{\mathsf{COMP}}, n + s_{\text{C-SUB}}, s'_{\text{C-COMP}})$ *ancillas.*

Proof. A controlled modular adder by a constant in VBE architecture, the first three subroutines map $|0\rangle_s\,|c\rangle_1\,|x\rangle_n$ to

$$|0\rangle_{s-1}\,|c\rangle_1\,|x\rangle_n\,|x+c\cdot a \mod p\rangle_n\,|\mathbf{1}[x+c\cdot a \geq p]\rangle_1 \ .$$

The comparator qubit storing $|\mathbf{1}[x+c\cdot a \geq p]\rangle_1$ is finally supposed to be uncomputed using C-$\mathsf{Q}'_{\mathsf{COMP}}(a)$. We can apply the MBU lemma here on the comparator qubit and the uncomputation subroutine C-$\mathsf{Q}'_{\mathsf{COMP}}(a)$, thus halving the cost of C-$\mathsf{Q}'_{\mathsf{COMP}}(a)$ in expectation. Therefore, we finally get a cost of $r = (r_{\text{C-ADD}} + r_{\mathsf{COMP}} + r_{\text{C-SUB}} + (r'_{\text{C-COMP}}/2))$ Tof gates in expectation, and the same ancilla count of a controlled constant modular adder. □

Finally, we introduce a new quantum arithmetic subroutine called the two sided comparison circuit which checks if a register $|x\rangle$ has a value in the range specified by two other quantum registers $|y\rangle$ and $|z\rangle$, and also show how its cost can be further reduced with MBU. Compared to a non-MBU implementation, we save 16.5% for the Tof gate cost.

Theorem 6 (Two-sided comparator)**.** *Let* $x, y, z \in \{0,1\}^n$*, and* $t \in \{0,1\}$ *Let us assume we have access to* $\mathsf{Q}_{\mathsf{COMP}}$*, with*

$$|x\rangle_n\,|y\rangle_n\,|t\rangle_1 \xmapsto{\mathsf{Q}_{\mathsf{COMP}}} |x\rangle_n\,|y\rangle_n\,|t \oplus \mathbf{1}[x<y]\rangle_1$$

[1]We denote by $|p|$ the Hamming weight of the bit string of the number p expressed in binary expansion. PCQFT stands for "partially-classical" quantum Fourier transform, where each controlled rotation in the QFT is replaced by a classically controlled rotation. Results with MBU are considered "in expectation". Draper (Expec) represents the repeated use of the Draper addition circuit. In expectation, the cost of the first QFT and last IQFT become negligible compared to the rest of the circuit.

	Logical Qubits	**Toffoli**		**CNOT,CZ**		**X**	
		w/o MBU	with MBU	w/o MBU	with MBU	w/o MBU	with MBU
(5 adder) VBE [1]	$4n+2$	$20n+10$	$16n+8$	$20n+2\|p\|+22$	$16n+2\|p\|+18$	$\|p\|+2$	$\|p\|+2.5$
(4 adder) VBE	$4n+2$	$16n+4$	$14n+4$	$20n+2\|p\|+18$	$17n+2\|p\|+15.5$	$2\|p\|+1$	$2\|p\|+1.5$
CDKPM [15]	$3n+2$	$8n$	$7n$	$16n+2\|p\|+4$	$14n+2\|p\|+3.5$	$2\|p\|+1$	$2\|p\|+1.5$
Gidney [12]	$4n+2$	$4n$	$3.5n$	$26n+2\|p\|+4$	$22.75n+2\|p\|+3.5$	$2\|p\|+1$	$2\|p\|+1.5$
CDKPM+Gidney	$3n+2$	$6n$	$5.5n$	$21n+2\|p\|+4$	$17.75n+2\|p\|+3.5$	$2\|p\|+1$	$2\|p\|+1.5$
	Logical Qubits	QFT_{n+1}		PCQFT_{n+1}		**-**	
		w/o MBU	with MBU	w/o MBU	with MBU	-	-
Draper [16]	$2n+2$	10	8	1	1		
Draper (Expect)	$2n+2$	8	6	1	1		

TABLE V: Cost of modular addition using different plain adder circuits in the VBE architecture of modular addition.

using s_{COMP} ancillas and r_{COMP} Tof gates. Assuming we also have access to $\mathsf{C\text{-}Q}'_{\mathsf{COMP}}$, with

$$|c\rangle_1 |x\rangle_n |y\rangle_n |t\rangle_1 \xmapsto{\mathsf{C\text{-}Q}'_{\mathsf{COMP}}} |c\rangle_1 |x\rangle_n |y\rangle_n |t \oplus c \cdot \mathbf{1}[x<y]\rangle_1$$

using $s'_{\mathsf{C\text{-}COMP}}$ ancillas and $r'_{\mathsf{C\text{-}COMP}}$ Tof gates. Then, there exists a circuit that performs the operation $\mathsf{Q}_{\mathsf{IN_RANGE}}$

$$|x\rangle_n |y\rangle_n |z\rangle_n |t\rangle_1 \xmapsto{\mathsf{Q}_{\mathsf{IN_RANGE}}} |x\rangle_n |y\rangle_n |z\rangle_n |t \oplus \mathbf{1}[x \in (y,z)]\rangle_1$$

using $s = 1 + s_{\mathsf{COMP}}$ ancillas and $r = 1.5r_{\mathsf{COMP}} + r'_{\mathsf{C\text{-}COMP}}$ Tof gates.

Proof. We begin by first checking if $y < x$. Applying $\mathsf{Q}_{\mathsf{COMP}}$ to the state $|0\rangle_s |x\rangle_n |y\rangle_n |z\rangle_n |t\rangle_1$ one gets

$$|0\rangle_{s-1} |x\rangle_n |y\rangle_n |z\rangle_n |\mathbf{1}[y<x]\rangle |t\rangle_1 \ .$$

Next we compute whether $x < z$ controlled on $\mathbf{1}[y < x]$, and store the output in our target. We know that $\mathbf{1}[x \in (y,z)] \equiv (\mathbf{1}[y<x] \cdot \mathbf{1}[x < z])$, and since we have both the comparator values stored in memory mapping the previous state to

$$|0\rangle_{s-1} |x\rangle_n |y\rangle_n |z\rangle_n |\mathbf{1}[y<x]\rangle_1 |t \oplus \mathbf{1}[x \in (y,z)]\rangle_1 \ .$$

We finally uncompute the intermediate value $\mathbf{1}[y < x]$ using $\mathsf{Q}_{\mathsf{COMP}}$ obtaining the state

$$|0\rangle_s |x\rangle_n |y\rangle_n |z\rangle_n |t \oplus (\mathbf{1}[y<x] \cdot \mathbf{1}[x<z])\rangle_1 \ .$$

Therefore, we obtain a cost of $r = 2r_{\mathsf{COMP}} + r'_{\mathsf{C\text{-}COMP}}$. This cost can be decreased by applying the MBU lemma on the $\mathsf{Q}_{\mathsf{COMP}}$ uncomputing $\mathbf{1}[y < x]$, thus leading to a cost of $r = 1.5r_{\mathsf{COMP}} + r'_{\mathsf{C\text{-}COMP}}$. □

V. Conclusions

Through this work, we aim to emphasize the importance of enhancing existing subroutines in known quantum algorithms, as these optimizations play a crucial role in refining resource estimates for algorithms with substantial practical applications. In modular arithmetic circuit, the efficiency of the circuit highly depends on the specific plain adder used in the implementation (e.g. [1], [12], [15], [16]). This can be seen by comparing the gate counts of various modular addition implementations in the columns "w/o MBU" in table V. The flexibility of our formalization for (controlled) quantum modular addition (by a constant) allows us to select any plain adder circuit for each of its four subroutines. An interesting variant we discovered optimizes the use of available ancilla qubits by applying the Gidney adder solely for the initial addition and final comparison steps, while using the CDKPM adder for constant subtraction and constant comparison. This approach results in a lower Tof count than a modular adder fully based on CDKPM adders ($6n$ vs. $8n$) and reduces the logical qubit count compared to a fully Gidney-based modular adder ($3n+2$ vs. $4n+2$). In section. II-A, we introduced the MBU lemma. We then applied MBU to different modular addition architectures to show the savings in Tof gate counts (see thms. 1, 2, 3, 4, 5). As seen in table. V, when comparing the Tof gate costs of the original implementations and their MBU optimized versions, we saw savings anywhere from $15-25\%$ in the gate costs. We stress that our techniques can similarly improve modular addition by a constant, controlled modular addition, and controlled modular addition by a constant.

This work opens up numerous promising research directions. First, the MBU Lemma can be studied and applied in the multi-qubit setting. Our improved circuit for modular addition can be plugged in the vast literature of quantum modular arithmetic subroutines, leading to savings in the gate count, see e.g. [27]–[30]. However, approximate modular addition techniques such as the coset representation [14] seem to currently be the dominant approach in scenarios that involve a large number of repeated additions, hence making a fruitful application of MBU in this setting non-trivial. Last but not least, we leave for future research the exploration of the resource implications of our new modular addition circuits when employed as subroutines in quantum algorithms.

Acknowledgements

This work started when AS was working at Inveriant Pte. Ltd. AS is supported by Innovate UK under grant *10004359*. AMM has received funding from the European Union's *Horizon 2020 research and innovation programme* under the grant agreement *N°101034324* and has been partially supported by the *Italian Group for Algebraic and Geometric Structures and their Application* (GNSAGA–INdAM). This work is supported by the National Research Foundation, Singapore, and A*STAR under its CQT Bridging Grant. We also acknowledge funding from the Quantum Engineering Programme (QEP 2.0) under grants *NRF2021-QEP2-02-P05* and *NRF2021-QEP2-02-P01*. We thank Anupam Chattopadhyay and Zeyu Fan for useful discussions. AMM and AL thank Niels Benedikter for the hospitality at Università degli Studi di Milano.

References

[1] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, p. 147, 1996.

[2] S. Beauregard, "Circuit for shor's algorithm using 2n+ 3 qubits," *arXiv preprint quant-ph/0205095*, 2002.

[3] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, pp. 124–134. [Online]. Available: https://ieeexplore.ieee.org/document/365700

[4] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[5] M. Ekerå, "Quantum algorithms for computing general discrete logarithms and orders with tradeoffs," *IACR Cryptol. ePrint Arch.*, p. 797, 2018. [Online]. Available: https://eprint.iacr.org/2018/797

[6] A. M. Childs, "Quantum information processing in continuous time," *Quantum Information Processing*, vol. 1, pp. 35–43, 2002. [Online]. Available: https://link.springer.com/article/10.1023/A:1019636608477

[7] D. W. Berry, C. Gidney, M. Motta, J. R. McClean, and R. Babbush, "Qubitization of arbitrary basis quantum chemistry leveraging sparsity and low rank factorization," *Quantum*, vol. 3, p. 208, 2019.

[8] J. F. Doriguello, A. Luongo, J. Bao, P. Rebentrost, and M. Santha, "Quantum algorithm for stochastic optimal stopping problems with applications in finance," in *17th Conference on the Theory of Quantum Computation, Communication and Cryptography*, 2022.

[9] D. Aggarwal, G. K. Brennen, T. Lee, M. Santha, and M. Tomamichel, "Quantum attacks on bitcoin, and how to protect against them," *arXiv preprint arXiv:1710.10377*, 2017.

[10] S. Wang, S. Deb, A. Mondal, and A. Chattopadhyay, "Optimal toffoli-depth quantum adder," *arXiv preprint arXiv:2405.02523*, 2024.

[11] S. Wang, X. Li, W. J. B. Lee, S. Deb, E. Lim, and A. Chattopadhyay, "A comprehensive study of quantum arithmetic circuits," *arXiv preprint arXiv:2406.03867*, 2024.

[12] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, 2018.

[13] N. Kornerup, J. Sadun, and D. Soloveichik, "Tight bounds on the spooky pebble game: Recycling qubits with measurements," *arXiv preprint arXiv:2110.08973*, 2021.

[14] C. Gidney, "Approximate encoded permutations and piecewise quantum adders," *arXiv preprint arXiv:1905.08488*, 2019.

[15] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv preprint quant-ph/0410184*, 2004.

[16] T. G. Draper, "Addition on a quantum computer," *arXiv preprint quant-ph/0008033*, 2000.

[17] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.

[18] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[19] C. Gidney, "Windowed quantum arithmetic," *arXiv preprint arXiv:1905.07682*, 2019.

[20] T. Draper, S. Kutin, E. Rains, and K. Svore, "A logarithmic-depth quantum carry-lookahead adder," *Quantum Information and Computation*, vol. 6, no. 4&5, pp. 351–369, 2006.

[21] D. Litinski, "How to compute a 256-bit elliptic curve private key with only 50 million toffoli gates," *arXiv preprint arXiv:2306.08585*, 2023.

[22] E. Gouzien, D. Ruiz, F.-M. Le Régent, J. Guillaud, and N. Sangouard, "Performance analysis of a repetition cat code architecture: Computing 256-bit elliptic curve logarithm in 9 hours with 126133 cat qubits," *Physical Review Letters*, vol. 131, no. 4, 7 2023. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.131.040602

[23] C. Zalka, "Shor's algorithm with fewer (pure) qubits," *arXiv preprint quant-ph/0601097*, 2006.

[24] Y. Yuan, C. Wang, B. Wang, Z.-Y. Chen, M.-H. Dou, Y.-C. Wu, and G.-P. Guo, "An improved qft-based quantum comparator and extended modular arithmetic using one ancilla qubit," *New Journal of Physics*, vol. 25, no. 10, p. 103011, 2023.

[25] Y. Takahashi, "Quantum arithmetic circuits: a survey," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 5, pp. 1276–1283, 2009.

[26] Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," *Quantum Information & Computation*, vol. 10, no. 9, pp. 872–890, 2010.

[27] I. L. Markov and M. Saeedi, "Constant-optimized quantum circuits for modular multiplication and exponentiation," *arXiv preprint arXiv:1202.6614*, 2012.

[28] R. Van Meter and K. M. Itoh, "Fast quantum modular exponentiation," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 71, no. 5, p. 052320, 2005.

[29] R. Rines and I. Chuang, "High performance quantum modular multipliers," *arXiv preprint arXiv:1801.01081*, 2018.

[30] A. Pavlidis and D. Gizopoulos, "Fast quantum modular exponentiation architecture for shor's factoring algorithm," *Quantum Information and Computation*, vol. 14, no. 7&8, pp. 0649–0682, 2014.

PatternPaint: Practical Layout Pattern Generation Using Diffusion-Based Inpainting

Guanglei Zhou*, Bhargav Korrapati†, Gaurav Rajavendra Reddy†, Chen-Chia Chang*, Jingyu Pan*, Jiang Hu‡, Yiran Chen*, and Dipto G. Thakurta†
*Dept. of Electrical & Computer Engineering, Duke University, Durham, USA
†Intel Corporation, Hillsboro, USA
‡Dept. of Electrical & Computer Engineering, TAMU, College Station, USA

***Abstract*—Generating diverse VLSI layout patterns is essential for various downstream tasks in design for manufacturing, as design rules continually evolve during the development of new technology nodes. However, existing training-based methods for layout pattern generation rely on large datasets. In practical scenarios, especially when developing a new technology node, obtaining such extensive layout data is challenging. Consequently, training models with large datasets becomes impractical, limiting the scalability and adaptability of prior approaches.**

To this end, we propose PatternPaint, a diffusion-based framework capable of generating legal patterns with limited design-rule-compliant training samples. PatternPaint simplifies complex layout pattern generation into a series of inpainting processes with a template-based denoising scheme. Furthermore, we perform few-shot finetuning on a pretrained image foundation model with only 20 design-rule-compliant samples. Experimental results show that using a sub-3nm technology node (Intel 18A), our model is the only one that can generate legal patterns in complex 2D metal interconnect design rule settings among all previous works and achieves a high diversity score. Additionally, our few-shot finetuning can boost the legality rate by 1.87X compared to the original pretrained model. As a result, we demonstrate a production-ready approach for layout pattern generation in developing new technology nodes.

I. Introduction

Generating synthetic pattern libraries is an essential and high-value element in technology development. However, this process faces significant challenges at advanced technology nodes. Engineers must first understand hundreds of design rules (DRs), and then create or modify pattern generators accordingly, resulting in lengthy turnaround times and substantial engineering effort. Moreover, this becomes more challenging as the DRs are constantly changing at the early stage of technology development. Each new DR set requires diverse patterns to support many downstream tasks, such as optical proximity correction (OPC) elements [1], [2], [3], [4], [5], [6], hotspot detection [7], [8], [9], [10], [11], design rule manual qualification [12]. These tasks require a wide spectrum of patterns to test/improve their methodologies and avoid unanticipated patterns that cause systematic failure.

Before the rise of machine learning, several rule/heuristic-based methods [13], [14], [15] were proposed to generate synthetic layout patterns. However, these heuristic methods demanded substantial engineering effort during development, as they required hundreds of design rules to be converted into algorithmic constraints. Moreover, these methods were often closely coupled with the DR set of a specific technology node, resulting in considerable time and effort to adjust them for new technology nodes. More recently, leveraging recent advances in generative models [16], [17], a number of training-based ML methods such as GANs, Transformers, TCAEs, and Diffusion models [18], [19], [20], [21], [22], [23] have been proposed with the promise of reduced engineering effort and high pattern diversity.

Despite these advancements, their practical application remains limited due to their dependence on large training datasets of clean DR layout samples. This limitation becomes particularly challenging in the early stage of technology node development. During this stage, design rules are continuously changing, and very few realistic layout patterns are available. Creating these thousands of training samples often requires rule-based methods to be coded as a pre-requisite, which demands significant time and effort. These constraints significantly restrict the deployment of training-based methods in critical Design for Manufacturability (DFM) applications.

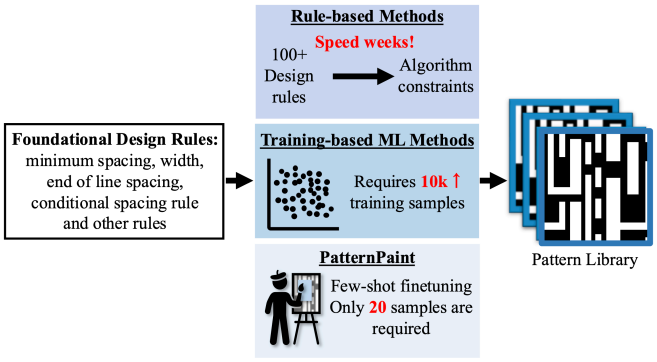


Fig. 1: Comparison between rule-based methods, training-based methods [19], [23], and our PatternPaint for layout pattern generation.

Additionally, these works have been demonstrated only in over-simplified academic design rule settings, rather than being tested in close-to-realistic scenarios. Due to the simplicity of the rule set, they decompose layout generation into generating pattern topologies (a blueprint of a layout pattern consisting only of the shape of patterns) and using a nonlinear solver to convert the topology into DR-clean layout patterns. However, this decomposition becomes unrealistic due to the following reasons. First, the solver's runtime grows exponentially with both the number of designs and pattern size. Second, when the DR set includes discrete constraints, the problem transforms into a mixed integer programming problem, resulting in significantly lower legality rates using the original nonlinear setting.

To address these challenges, we propose PatternPaint, a few-shot inpainting framework capable of generating legal patterns. Our unique advantage is highlighted in Figure 1. Our framework primarily targets single metal layer pattern generation to support DFM tasks, such as pattern feasibility analysis using OPC models. PatternPaint simplifies layout generation into a series of inpainting processes, which naturally exploit design rule information encoded in neighboring regions of existing patterns. PatternPaint operates at the pixel level through a customized template-based denoising scheme, bypassing the need for solver-based legalization.

Our contributions are outlined as follows:

- We present PatternPaint, the first few-shot pattern generation framework that leverages inpainting to drastically reduce training sample requirements for legal pattern generation.
- We decompose layout pattern generation into a series of inpainting processes with a novel template-based denoising scheme specifically designed for layout patterns. Our denoise method outperforms the conventional denoising method [24] by achiev-

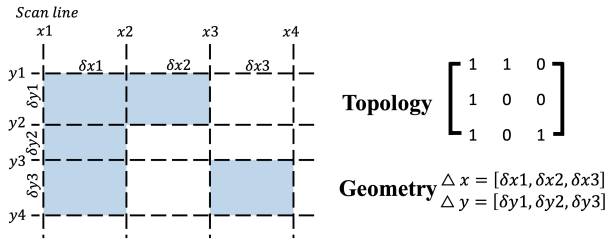


Fig. 2: Squish Pattern Representation.

ing a tenfold increase in legal pattern generation.

- **Validation on industrial PDKs:** PatternPaint is the first ML approach validated on an industrial PDK Intel 18A with full-set sign-off DR settings. Using only 20 starter samples, PatternPaint generates over 4000 DR-clean patterns, while prior ML solutions fail to deliver DR-clean patterns by training with 1k samples.

II. Preliminaries

A. Diffusion model

Diffusion models [25] are generative models that operate through forward and reverse diffusion processes. The forward process gradually adds Gaussian noise to data over T timesteps:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}; \beta_t\mathbf{I}) \quad (1)$$

$$q(x_1, ..., x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \quad (2)$$

where x_0 is the original sample, x_t represents noise-corrupted samples, and β_t controls the noise schedule. As T increases, the data distribution approaches Gaussian:

$$q(x_T) \approx \mathcal{N}(x_t; 0; \mathbf{I}) \quad (3)$$

The reverse process generates samples by learning to denoise:

$$p_\theta(x_0) = \int \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) dx_{1:T} \quad (4)$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t); \sum_\theta(x_t, t)) \quad (5)$$

where θ represents neural network parameters trained to minimize the objective:

$$L = \sum_{t=0}^{T} D_{KL}(q(x_t|x_{t+1}, x_0)||p_\theta(x_t|x_{t+1})), t \in [1, T-1] \quad (6)$$

For inpainting tasks [26], this process is conditioned on known image regions to fill masked areas consistently with the surrounding content. We found this characteristic aligns well with VLSI layout pattern generation, as design rule information is largely encoded in neighboring regions. By conditioning the generation on legal neighboring layouts, our model naturally learns to produce design-rule-compliant patterns. In later sections, we demonstrate how this approach significantly reduces training sample requirements while enriching pattern libraries.

B. Squish representation

Standard layout patterns are typically composed of multiple polygons, presenting a sparse informational structure. To efficiently represent these patterns, majority of existing training-based methods [21], [18], [19], [27], [23] use "Squish" pattern [28], [29] to address these issues by compressing a layout into a concise pattern topology matrix alongside geometric data $(\triangle_x, \triangle_y)$, as illustrated in Fig. 2. This process involves segmenting the layout into a grid framework using a series of scan lines that navigate along the edges of the polygon.

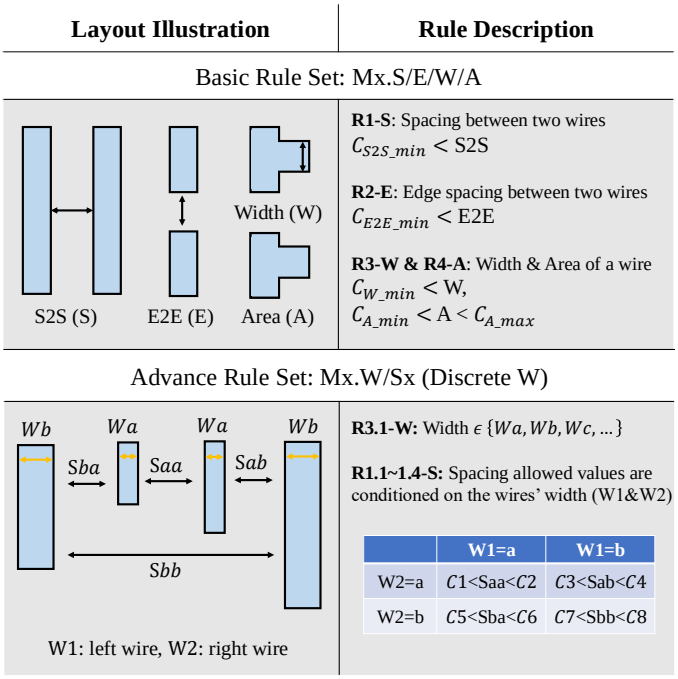


Fig. 3: Illustration of metal layer design rules. A selected set of design rules used in PatternPaint evaluation is shown as the advance rule set. (1) Basic Rule Set: Spacing (R1-S,R2-E)/ Width (R3-W)/ Area (R4-A) of Mx layer metal element. (2) Advance Rule Set: (R3.1-W) Only a set of discrete widths is allowed. (R1.1 1.4-S), the allowed spacing range is different depending on the neighboring metal widths.

The distances between each adjacent pair of scan lines are recorded in the $\triangle$ vectors. The topology matrix itself is binary, with each cell designated as either zero (indicating an absence of shape) or one (indicating the presence of a shape).

Existing training-based methods focus on topology generation, leaving the geometry vector solved by a nonlinear solver. We refer to all the prior approaches that use squish representation as **squish-based** solutions. However, these solvers lack scalability and cannot handle advanced design rules effectively. To overcome these limitations, PatternPaint switches to a simpler **pixel-based** representation where $\triangle x_i, \triangle y_i$ are pre-defined with fixed physical widths (e.g., 1nm × 1nm rectangles per pixel).

C. Related Works

Recent years, the development of training-based ML solutions for layout pattern generation has emerged. DeePattern [18] pioneered this field by using a Variational Autoencoder (VAE) [30] model to generate 1D layout patterns for 7nm EUV unidirectional settings with fixed metal tracks. This method employed a squish representation and a nonlinear solver to solve for the geometry vectors. Then, CUP [21] expanded the approach to 2D pattern generation, creating a large 2D academic layout pattern dataset containing 10k of training samples for a simple design rule setting (minimum width, spacing, and area). Under this dataset, LayouTransformer [20] introduced transformer-based sequential modeling, and DiffPattern [19] employed discrete diffusion methods. Additional explorations include transferability [27], free-size pattern generation [23], and controllable generation [22].

All existing ML works have been demonstrated only in basic DR settings. In contrast, PatternPaint addresses a full set of industrial standard DRs. As shown in Figure 3, the advanced rule set introduces significantly more complex constraints. Under these complex rules, the nonlinear solver-based legalization used in existing state-of-the-art methods [19], [23] becomes unscalable due to the presence of

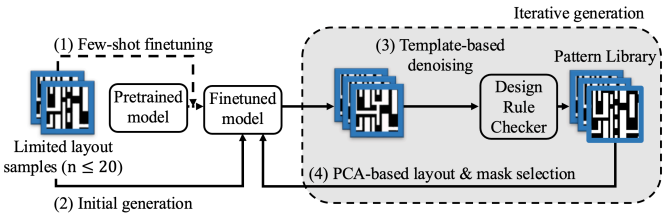


Fig. 4: PatternPaint framework. It consists of: (1) few-shot finetuning, (2) initial generation, (3) template-based denoising for layout refinement, followed by a design rule checking validation, and (4) PCA-based layout & mask selection to select the next inpainting samples for iterative generation. This approach enables efficient pattern generation while ensuring design rule compliance.

discrete widths and upper bounds on spacing. This scalability issue is evident in [23], where lower success rates are observed as pattern sizes increase. PatternPaint's pixel-based approach overcomes these limitations, enabling efficient pattern generation under more realistic and complex design rule constraints. The limitation of the solver is further discussed in Section VI.

III. Problem Formulation

In this section, we formalize the pattern generation problem and its evaluation criteria. The objective is to produce diverse, realistic layout patterns from a small set of existing designs while ensuring compliance with rigorous design rules. We use a variety of metrics for quantitative evaluation.

(1) Legality: A layout pattern is legal iff it is DR-clean.

(2) Entropy H_1: As detailed in [19], the complexity of a layout pattern is quantified as a tuple (C_x, C_y) representing the count of scan lines along the x-axis and y-axis, respectively, each reduced by one. Then, we can obtain

$$H_1 = \sum_{i,j} P(C_{x_i}, C_{y_j}) log P(C_{x_i}, C_{y_j})$$

where $P(C_{x_i}, C_{y_j})$ is the probability of encountering a pattern with complexities C_{x_i} and C_{y_j} within the library. This metric only focuses on topology diversity without considering any geometric information from actual patterns.

(3) Entropy H_2: To further examine the diversity of actual patterns with their geometric information included, we introduce H_2. For each unique combination of $\triangle x$ and $\triangle y$ (defined in Section II-B) presented in the library, we record their probability $P(\Delta_{x_i}, \Delta_{y_j})$ of having a pattern with the same $\Delta_{x,y}$ matrix within the library.

$$H_2 = \sum_{i,j} P(\Delta_{x_i}, \Delta_{y_j}) log P(\Delta_{x_i}, \Delta_{y_j})$$

Since we target on pixel level generation, H_2 serves as the main metric to evaluate generation performance.

Based on the aforementioned evaluation metrics, the pattern generation problem can be formulated as follows.

Problem 1 (Pattern Generation). *Given a set of design rules and existing patterns, the objective of pattern generation is to synthesize a legal pattern library such that H_2 of the layout patterns in the library is maximized.*

IV. PatternPaint

A. Overview

As illustrated in Figure 4, PatternPaint integrates four key components to achieve efficient layout pattern generation: (1) few-shot finetuning, (2) initial generation, (3) template-based denoising and (4) PCA-based layout & mask selection. In the following sections, we detail how these components work together to produce diverse and legal layout patterns.

B. Few-shot Finetuning

PatternPaint adapts a pretrained text-to-image diffusion model to VLSI layouts through few-shot finetuning using the limited available layout samples. During finetuning, we fixed the text encoder and only finetuned on the image diffusion model. This finetuning shares a similar training objective with Equation (6) but starts from the θ' in the pretrained diffusion neural network. The training objectives then become

$$\begin{aligned} L = & D_{KL}(q(x_T|x_0)||p_{\theta'}(x_T)) - log p_{\theta'}(x_0|x_1) \\ & + \sum_{t=2}^{T} D_{KL}(q(x_t|x_{t+1}, x_0)||p_{\theta'}(x_t|x_{t+1})) \\ & + \lambda L_{prior} \end{aligned} \tag{7}$$

where x_0 is selected from the limited n layout patterns at the finetuning stage, L_{prior} represents the prior preservation loss calculated on a set of class-specific images generated before training, and λ is a weighting factor that balances the influence of prior preservation. The L_{prior} helps serve as a regularization term to enable the model to learn very sparse samples while avoiding overfitting. The class-specific images are obtained by giving a fixed prompt to a text-to-image pretrained model. The interested reader is referred to [31], [32] for details.

C. Initial Pattern Generation

After finetuning, PatternPaint begins the initial generation phase using the n starter patterns from finetuning. Unlike prior approaches that generate entire patterns at once, our method decomposes generation into localized inpainting processes, mimicking how human engineers make targeted adjustments while preserving surrounding structures.

The generation process requires two inputs: (1) starter patterns that provide design rule context and (2) mask images that specify regions for variation. A masked image x_0^{masked} is created by applying the mask to a starter pattern, where masked regions (region replaced with Gaussian noise) are set to be predicted while unmasked regions remain unchanged. We provide 10 predefined masks (illustrated in Figure 6), though users can customize masks to target specific regions of interest. For each starter pattern-mask combination, the model generates multiple variations, producing a total of $n \times 10 \times v$ patterns in the initial iteration, where n is the number of starter patterns and v is the number of variations generated during inpainting.

Inpainting. During inpainting, the model predicts the masked regions while conditioning on the known pixels. The reverse diffusion process is modified as:

$$p_\theta(x_{t-1}|x_t, x_0^{masked}) = N(x_{t-1}; \mu_\theta(x_t, x_0^{masked}, t), \sum_\theta(x_t, x_0^{masked}, t)) \tag{8}$$

The mean and covariance now also depend on the original masked image x_0^{masked}, conditioning the reverse process on the known pixels. We also follow the inference scheme mentioned in [32] that only generates masks with about 25% region of its target image size.

D. Template-based Denoising and DR Checking

The inpainting process, while effective for generating a big set of pattern variations, introduces noise along polygon edges due to the lossy nature of latent diffusion models. This edge noise can significantly alter pattern dimensions and lead to design rule violations.

To address this challenge, we propose automated template-matching denoising, listed in Algorithm 1, inspired by the fact that only a sub-region of an image is changed during inpainting and the scan lines of the starter pattern (pre-inpainting) are known. We use the

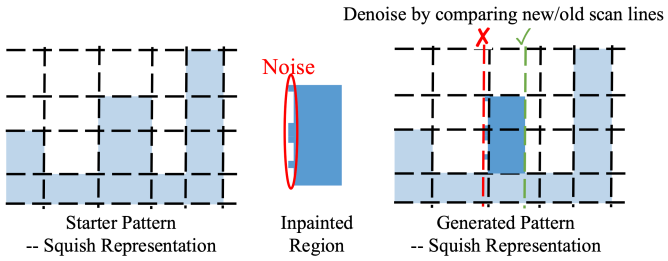


Fig. 5: Illustration of Template-based Denoising. Noise at the edge is reduced by comparing new scan lines with the original scan lines (black). Here, green scan line is preserved since it is larger than a predefined threshold, and red scan line is removed.

Algorithm 1 Template-based Denoising

Input: Generated noisy image I_g, template (noise-free) image I_t, and threshold T
Output: Denoised output image I_o

$L_g \leftarrow$ *extract_squish_lines*(I_g)
$L_t \leftarrow$ *extract_squish_lines*(I_t)
Cluster L_g into subsets $C_1, C_2, \ldots, C_n$ such that for each cluster C_i, $\|L_g(i) - L_g(j)\| \leq T$
for each cluster C_i
    $l_{\text{match}} \leftarrow$ single scan line from L_t that minimizes $\|l - C_i\|$
    if $\|l_{\text{match}} - C_i\| \leq T$
        Replace C_i with l_{match} ▷ Replace cluster with matched scan line from template
    else
        Randomly select $L_{\text{random}} \in C_i$ and replace the cluster with L_{random}
Construct the topology matrix M from the modified L_g
$I_o \leftarrow$ *reconstruct_image*(M, L_g)
return I_o

squish representation mentioned in Section 2.2, where we first extract scan lines from the noisy generated pattern (post-inpainting) and cluster similar lines within a predefined threshold. We then compare them to scan lines from the template (starter pattern). For each cluster, a parent scan line is chosen if available; otherwise, a line is randomly selected from within the cluster. This method is very effective, and we observe that it significantly increases the number of patterns passing DR checks. Figure 5 also gives an intuitive example of denoising is performed by neglecting unnecessary scan lines due to edge noise but still preserving the scan lines. Denoising is performed by extracting the topology matrix using the designated scan lines and reconstructing the pattern again. A quantitative evaluation of this template-based denoising is shown in later Section VI and Table III.

E. PCA-based layout & mask selection

After the initial generation, a vast set of pattern variations is obtained. To produce more new and diverse layout patterns, iterative generation is employed, altering only a sub-region of the image in each iteration. For each iteration, we adopt a PCA-based approach to pick k representative samples from the existing pattern library, followed by a mask selection scheme using two mask sets.

1) PCA-based layout selection: As described in Algorithm 2, we propose a PCA-based layout selection scheme to pick representative layouts for the next iteration generation. PCA reduction provides a qualitative means to illustrate the diversity of a given layout pattern library [14]. The input samples are DR-clean layout clips. We first apply PCA to decompose images into several most representative components. To preserve most of the information in the dataset, we push the PCA to have explained_varaince (0.9, meaning 90% of the explained variance is preserved in the dimension-reduced components. Then, an iterative selection is performed to ensure that diverse samples are extracted from the existing library while meeting density constraints. The constraints can be easily integrated with other requirements such as specific pattern shapes or other interesting features and perform layout pattern generation in a more controlled setting.

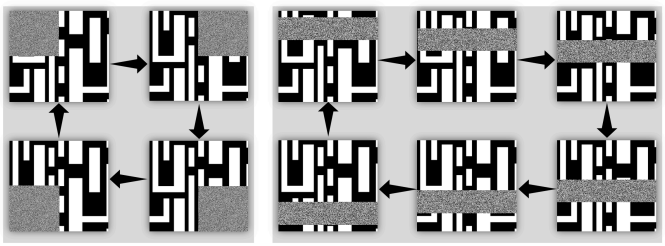


Fig. 6: Predefined mask sets for pattern generation: default masks (left) and horizontal masks (right). Horizontal masks are customized for our dataset since we primarily focus on vertical track layout generation. Mask in each set is selected sequentially during iterative generation.

Algorithm 2 PCA-based Representative Layout Selection

Input: Dataset $X \in \mathbb{R}^{n \times d}$, target samples k, constraints C
Output: Selected samples $S \subset X$

$X_{pca} \leftarrow \text{PCA}(X)$ ▷ Dimensionality reduction
$I_s \leftarrow \{\}$, $I_r \leftarrow \{1, \ldots, n\}$ ▷ Selected and remaining indices
$i_0 \leftarrow \text{random}(I_r)$ ▷ Initial random sample
$I_s \leftarrow I_s \cup \{i_0\}$, $I_r \leftarrow I_r \setminus \{i_0\}$
for $t \leftarrow 1$ to $k-1$
    for $i \in I_r$
        $d_i \leftarrow \sum_{s \in I_s} \|X_{pca}[i] - X_{pca}[s]\|$ ▷ Sum of distances
    $i^* \leftarrow \arg\max_{i \in I_r} d_i$ s.t. $C(X[i])$ ▷ Farthest point
    $I_s \leftarrow I_s \cup \{i^*\}$, $I_r \leftarrow I_r \setminus \{i^*\}$
return $X[I_s]$

2) Mask selection: As illustrated in Figure 6, our framework defined two mask sets (10 masks total) to guide pattern generation. The default mask set enables general pattern variations through targeted modifications, including metal wire modification and inter-track connections. The horizontal mask set is specifically designed for vertical track layouts to enhance exploration of end-to-end design rules and inner-track interactions. For horizontal track layout generation, a vertical mask set shall also be proposed.

For each selected layout, we generate its mask following a predefined sequential schedule within each set. For example, when a pattern undergoes modification in one region (e.g., top-left in the default mask set), subsequent iterations target adjacent regions (e.g., top-right) following the predefined sequence. This sequential approach preserves features generated in previous iterations while providing rich contextual information for the inpainting model through newly generated patterns.

F. Iterative Generation

As illustrated in the grey region of Figure 4, the final iteration generation process then integrates Algorithm 2 to select representative layouts from the existing pattern library with a mask provided by its own mask set. Our framework keeps performing iterative generation until the desired diversity is reached or the sample budget is exceeded. When the iterations are completed, a diverse pattern library within the given DR space can be generated.

V. Experimental Results

A. Experimental Setup

We validate PatternPaint on Intel 18A technology node with all generated patterns verified through industry-standard DR checking. The dataset contains 20 starter patterns.

Model setting: We experiment on two pre-trained models, including stablediffusion1.5-inpaint (PatternPaint-sd1-base) and stablediffusion2-inpaint (PatternPaint-sd2-base) [32].

Finetuning details: We adhered to the procedure described in DreamBooth [31] to finetune the inpainting model with 20 layout patterns. The learning rate is set to 5e-6. For PatternPaint-sd1-base (PatternPaint-sd2-base), we denote its fine-tuned model as PatternPaint-sd1-ft (PatternPaint-sd2-ft). Experiments are performed on one Nvidia A100 GPU and one Intel(R) Xeon(R) Gold 6336Y CPU@2.40GHz. Finetuning time takes around 10 minutes. The average time for generating is 0.81 seconds and 0.21 seconds for denoising per sample.

Baseline methods: We conducted comparisons using two state-of-the-art methods, CUP [21] and DiffPattern [19]. Since 20 patterns in our dataset are not enough to train diffusion-based and VCAE-based solution, we further obtain 1000 samples from a commercial tool with a size of 512 x 512 pixels to train CUP [21] and DiffPattern [19] in squish representation [29]. The topology size for these experiments was set to 128 x 128 pixels. DiffPattern, which employs a non-linear solver-based legalization process, initially supported only three basic design rules. However, this is inadequate for Intel 18A, which includes constraints such as discrete values for certain line widths. We tried our best to improve this solver to accommodate a subset of the design rules that involve max-spacing, max-width, and discrete values for certain line widths. After this improvement, legal layout patterns started to appear. We implemented this nonlinear solver using scipy package, and the maximum iteration count is set to 10^8.

Initial generation: Since previous works are one-time generation methods, to establish a fair comparison, we also present the results of the first stage of PatternPaint, initial generation, into performance comparison. For each initial pattern, each model generated 100 layout patterns per mask. In total, we generate 20,000 patterns. The performance of the initial generation across 4 models is denoted as (model-name)-init.

Iterative generation: Following the initial generation, we created a library of unique patterns with substantial variation. We then perform iterative pattern generation, as described in section IV-E, to check if diversity increased through this process. We designated the unique patterns from experiment 1 (Table 1) as our first iteration. For subsequent iterations, we conducted PCA analysis to select 100 of the most sparse representative samples, with the density constraint set at 40% for the selected patterns. For each iteration, we generated 5000 samples out of the 100 patterns, adding only clean and new samples to our existing pattern library. We performed 6 iterative generations and collected 50000 generated patterns in total. The performance of iterative generation across 4 models is denoted as (model-name)-iter.

B. Comparison of Pattern Generation

The evaluation results of the initial generation are shown in Table I. 10% of the layout patterns generated by PatternPaint are legal and show better H_1 and H_2 than with the starter patterns. Compared with other baselines, CUP is unable to generate legal patterns, and DiffPattern only generated four legal patterns.

The effectiveness of the proposed fine-tuning process is evident when comparing PatternPaint-sd1-base-init (and PatternPaint-sd2-base-init) with their fine-tuned counterparts, PatternPaint-sd1-ft-init (and PatternPaint-sd2-ft-init). Fine-tuned models show improvements in the number of legal patterns, unique patterns, and the main metric H_2. The results of iterative generation are in Figure 7. As iterations proceed, both the unique pattern count and H_2 increase, further highlighting the gap between baseline models (PatternPaint-sd1-base, PatternPaint-sd2-base) and fine-tuned models (PatternPaint-sd1-ft, PatternPaint-sd2-ft), with the latter consistently outperforming the former. This validates that our fine-tuning process demonstrates significant model improvements.

TABLE I: Performance comparison for layout pattern generation.

Method	**Generated Patterns**	**Legal Patterns**	**Unique Patterns**	H_1	H_2
Starter patterns	-	20	20	3.68	4.32
CUP [21]	20000	0	0	0	0
DiffPattern [19]	20000	4	4	2	2
PatternPaint-sd1-base-init	20000	1251	928	5.06	9.78
PatternPaint-sd2-base-init	20000	1479	861	5.15	9.60
PatternPaint-sd1-ft-init	20000	2336	1728	4.65	10.49
PatternPaint-sd2-ft-init	20000	1630	1469	4.96	10.46
PatternPaint-sd1-base-iter	50000	5021	3066	4.31	11.37
PatternPaint-sd2-base-iter	50000	5083	2583	4.19	11.02
PatternPaint-sd1-ft-iter	50000	7229	4458	4.08	11.80
PatternPaint-sd2-ft-iter	50000	5982	4616	4.11	**12.01**

TABLE II: Runtime comparison with our method and DiffPattern.

Method	Avg Runtime (s)
PatternPaint (Inpainting)	0.81
PatternPaint (Denoising)	0.21
DiffPattern	38.04

We observe a slight decrease in H_1 as the iterative process proceeds, which can be attributed to the fact that H_1 primarily focuses on topology diversity. Since our framework alters only a sub-region of a given layout at a time, this results in several replicated topologies with adjustments limited to physical width, leading to the observed decrease in H_1. However, many DFM studies, such as OPC recipe development, benefit not only from topology diversity but also from variations in physical width combined with a given topology. This is captured by our key metric H_2, which considers both topology and physical dimensions. As iterations progress, more patterns with higher diversity are generated, including variations in physical widths and connection types. These diverse patterns can be used in yield analysis of metal patterns as well as finetuning OPC recipes, addressing the critical needs for real-world DFM applications.

Figure 8 visually represents the variations generated by our proposed methods. The starter pattern is depicted in (a), (b-f) show the generated patterns. We observed that the proposed methods explored a wide range of variations, demonstrating the models' awareness of tracks. For example, in (f), the model attempts to disconnect from an adjacent thick track and establish a connection with a farther one. In (e), more complex changes were made, forming connections with even farther tracks and upper objects. These alterations enrich the pattern library and represent a unique feature of the ML-based method. Achieving such inter-track alternations with a rule-based method would require significant engineering effort, making it nearly impossible without advanced techniques coded for the given DR set.

Table II exhibits the runtime comparison with DiffPattern. Note that we omit the comparison with CUP because it is unable to give a feasible solution. The runtime for DiffPattern is 30× longer than PatternPaint due to the time-consuming solver-based legalization process. Overall, these results demonstrate the effectiveness of PatternPaint to generate legal patterns and indicate that the solver-based solution is not suitable for industrial DR settings.

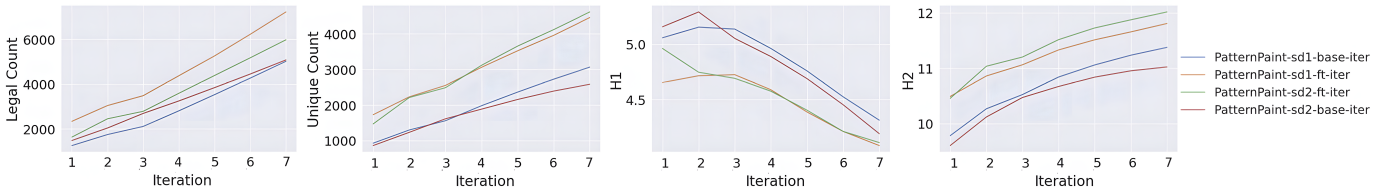
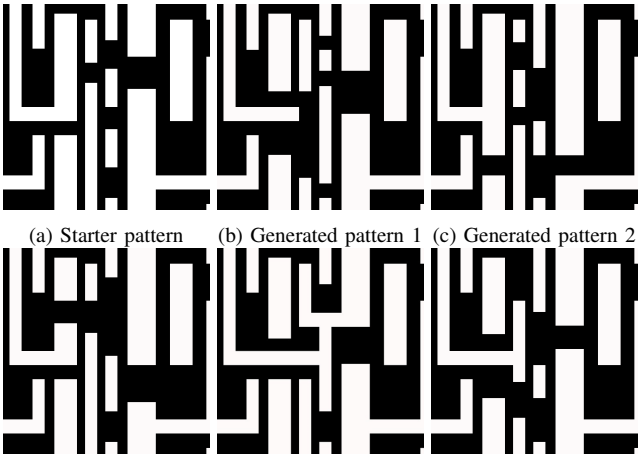


Fig. 7: Experimental results for iterative generation process using four metrics: legal pattern counts, unique pattern counts, H_1, and H_2.



(a) Starter pattern (b) Generated pattern 1 (c) Generated pattern 2

(d) Generated pattern 3 (e) Generated pattern 4 (f) Generated pattern 5

Fig. 8: Generated variations from a starter pattern.

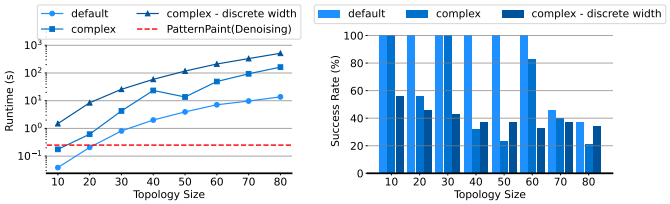


Fig. 9: Runtime and success rate analysis of non-linear solver under three design rule settings: default, complex, and complex-discrete width. Results show exponential runtime growth and declining success rates with increasing topology size.

VI. Ablation Study

We conduct comprehensive ablation studies to validate two key aspects: (1) the limitations of solver-based approaches with increasing design rule complexity, and (2) the effectiveness of our template-based denoising scheme.

1) Impact of Design Rule Complexity: We evaluate three progressively design rule settings to illustrate the solver limitations as shown in Figure 9. The *default* setting follows the academic design rule set of [19], including basic constraints such as minimum width/spacing and area checks. The *complex* setting extends this by differentiating horizontal and vertical directions for width and spacing checks, including their minimum and maximum values. The *complex-discrete* setting further restricts width values to discrete sets. We observe two critical limitations of this non-linear solver. First, the solver's runtime increases significantly from default to complex-discrete settings, showing exponential growth with topology size, significantly exceeding PatternPaint's denoising time. Second, despite the existence of legal solutions, the solver's success rate deteriorates with rule complexity. For topologies larger than 60×60, all settings achieve less than 50% success rate. This scalability issue is also evident in [23], where lower success rates are observed as pattern sizes increase.

TABLE III: Comparison of the pattern generation success rate (S%) using PatternPaint with different denoising schemes: our template-based denoising, OpenCV non-local means filter [24], and without denoising.

Method	W/ Template-based Denoise (S↑)	W/ OpenCV Denoise Filter [24] (S↑)	W/o Denoise (S↑)
PatternPaint-sd1-base	6.25	0.12	0
PatternPaint-sd1-ft	11.68	1.04	0
PatternPaint-sd2-base	7.40	0.24	0
PatternPaint-sd2-ft	8.15	0.76	0
Average	**8.37**	**0.86**	**0**

2) Effectiveness of Template-based Denoising: Table III evaluates the effectiveness of our template-based denoising scheme in the PatternPaint framework. We compare the template-based denoising scheme with a widely used denoising filter, the non-local means filter [24] implemented in OpenCV. We also show the DRC results without any denoising activity. The generation success rate is calculated by legal patterns divided by total generated patterns. The results show that no patterns can be directly used without denoise. Our template-based denoising significantly outperforms the OpenCV non-local means filter [24], with an average of 9.7x generation success rate improvement. The fine-tuned versions of PatternPaint achieve the highest success rates when combined with template-based denoising, reaching 11.68%. These findings validate the effectiveness of the template-based denoising scheme in maximizing pattern generation efficiency, especially when combined with fine-tuning techniques.

VII. Conclusion

In this paper, we propose PatternPaint, an automated few-shot pattern generation framework using diffusion-based inpainting. We develop our own unique template-based denoising scheme to tackle noise and propose a PCA-based sample selection scheme for iterative pattern generation. In the initial round of the iterative generation process, thousands of DR clean layouts are generated on the latest Intel PDK and checked through an industry-standard DR checker. In later iterations, by measuring entropy, we observed that pattern diversity improves. Our work, PatternPaint, has its unique benefits with little to no human effort in loop and is the first can perform pattern generation in a few-shot learning scenario. In future work, we will improve PatternPaint to support larger size pattern generation and explore further finetuning the pre-trained models using legal samples collected from PatternPaint enriched pattern library. We also plan to evaluate the explored design rule space against product-level layout patterns and demonstrate the application of PatternPaint-generated patterns on yield learning test chips for future PDK development and DFM studies on Intel silicon.

VIII. Acknowledgement

This work is partially supported by SRC 3104.001 and NSF 2106828.

REFERENCES

[1] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proceedings of the Design Automation Conference (DAC)*, pp. 1–6, 2014.

[2] Y. Jiang, F. Yang, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via neural architecture search," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, June 2022.

[3] B. Jiang, H. Zhang, J. Yang, and E. F. Y. Young, "A fast machine learning-based mask printability predictor for opc acceleration," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 412–419, 2019.

[4] J. Kuang, W.-K. Chow, and E. F. Y. Young, "A robust approach for process variation aware mask optimization," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1591–1594, 2015.

[5] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Y. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proceedings of the Design Automation Conference (DAC)*, pp. 1–6, 2018.

[6] A. Hamouda, M. Bahnas, D. Schumacher, I. Graur, A. Chen, K. Madkour, H. Ali, J. Meiring, N. Lafferty, and C. McGinty, "Enhanced opc recipe coverage and early hotspot detection through automated layout generation and analysis," in *Optical Microlithography XXX*, vol. 10147, p. 101470R, SPIE, 2017.

[7] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[8] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *Proceedings of the Design Automation Conference (DAC)*, pp. 1–6, 2019.

[9] J. Pan, C.-C. Chang, Z. Xie, J. Hu, and Y. Chen, "Robustify ml-based lithography hotspot detectors," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2022.

[10] H. Zhang, B. Yu, and E. F. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.

[11] H. Yang, Y. Lin, B. Yu, and E. F. Y. Young, "Lithography hotspot detection: From shallow to deep learning," in *Proceedings of the 30th IEEE International System-on-Chip Conference (SOCC)*, pp. 233–238, 2017.

[12] A. Kabeel, S. Kim, Y. G. Park, D. Kim, J. Kwan, S. Rizk, K. Madkour, M. Shafee, and J. Kim, "Design rule manual and drc code qualification flows empowered by high coverage synthetic layouts generation," in *DTCO and Computational Patterning II*, vol. 12495, pp. 415–428, SPIE, 2023.

[13] H. Li, E. Zou, R. Lee, S. Hong, S. Liu, J. Wang, C. Du, R. Zhang, K. Madkour, H. Ali, *et al.*, "Design space exploration for early identification of yield limiting patterns," in *Design-Process-Technology Co-optimization for Manufacturability X*, vol. 9781, 2023.

[14] G. R. Reddy, M.-M. Bidmeshki, and Y. Makris, "VIPER: A versatile and intuitive pattern generator for early design space exploration," in *Proceedings of the IEEE International Test Conference (ITC)*, pp. 1–7, 2019.

[15] G. R. Reddy, C. Xanthopoulos, and Y. Makris, "Enhanced hotspot detection through synthetic pattern generation and design of experiments," in *Proceedings of IEEE VLSI Test Symposium (VTS)*, pp. 1–6, 2018.

[16] J. Pan, G. Zhou, C.-C. Chang, I. Jacobson, J. Hu, and Y. Chen, "A survey of research in large language models for electronic design automation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, Feb. 2025.

[17] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[18] H. Yang, S. Li, W. Chen, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Deepattern: Layout pattern generation with transforming convolutional auto-encoder," *IEEE Transactions on Semiconductor Manufacturing*, vol. 35, no. 1, pp. 67–77, 2022.

[19] Z. Wang, Y. Shen, W. Zhao, Y. Bai, G. Chen, F. Farnia, and B. Yu, "Diff-pattern: Layout pattern generation via discrete diffusion," in *Proceedings of the Design Automation Conference (DAC)*, pp. 1–6, 2023.

[20] L. Wen, Y. Zhu, L. Ye, G. Chen, B. Yu, J. Liu, and C. Xu, "LayouTransformer: Generating layout patterns with transformer via sequential pattern modeling," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2022.

[21] X. Zhang, J. Shiely, and E. F. Young, "Layout pattern generation and legalization with generative learning models," in *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[22] Q. Wang, X. Zhang, M. D. Wong, and E. F. Young, "Controlayout: Conditional diffusion for style-controllable and violation-fixable layout pattern generation," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLS)*, p. 511–515, 2024.

[23] Z. Wang, Y. Shen, X. Yao, W. Zhao, Y. Bai, F. Farnia, and B. Yu, "ChatPattern: Layout pattern customization via natural language," in *Proceedings of the Design Automation Conference (DAC)*, pp. 1–6, 2024.

[24] "Image denoising." https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html, 2019. Accessed: 2024-09-28.

[25] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 6840–6851, 2020.

[26] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11451–11461, 2022.

[27] X. Zhang, H. Yang, and E. F. Young, "Attentional transfer is all you need: Technology-aware layout pattern generation," in *Proceedings of the Design Automation Conference (DAC)*, pp. 169–174, 2021.

[28] F. E. Gennari and Y.-C. Lai, "Topology design using squish patterns," *U.S. Patent 8832621B1*, 2014.

[29] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multi-layer layout hotspots with adaptive squish patterns," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, p. 299–304, 2019.

[30] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[31] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein, and K. Aberman, "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22500–22510, 2023.

[32] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022.

A Scalable and Robust Compilation Framework for Emitter-Photonic Graph State

Xiangyu Ren[†*], Yuexun Huang[‡], Zhiding Liang[§*], Antonio Barbalace[†]
[†] University of Edinburgh, Edinburgh, Scotland, UK
[‡] University of Chicago, Chicago, IL, USA
[§] Rensselaer Polytechnic Institute, Troy, NY, USA

***Abstract*—Quantum graph states are critical resources for various quantum algorithms, and also determine essential interconnections in distributed quantum computing. There are two schemes for generating graph states — *probabilistic scheme* and *deterministic scheme*. While the all-photonic *probabilistic scheme* has garnered significant attention, the emitter-photonic *deterministic scheme* has been proved to be more scalable and feasible across several hardware platforms.**

This paper studies the GraphState-to-Circuit compilation problem in the context of the *deterministic scheme*. Previous research has primarily focused on optimizing individual circuit parameters, often neglecting the characteristics of quantum hardware, which results in impractical implementations. Additionally, existing algorithms lack scalability for larger graph sizes. To bridge these gaps, we propose a novel compilation framework that partitions the target graph state into subgraphs, compiles them individually, and subsequently combines and schedules the circuits to maximize emitter resource utilization. Furthermore, we incorporate *local complementation* to transform graph states and minimize entanglement overhead. Evaluation of our framework on various graph types demonstrates significant reductions in CNOT gates and circuit duration, up to 52% and 56%. Moreover, it enhances the suppression of photon loss, achieving improvements of up to $\times$1.9.

I. Introduction

Quantum information processing has the potential to revolutionize computation, communication, cryptography, sensing, and imaging. Within quantum information, entanglement plays a vital role as it serves as the foundation for interconnections between qubits. *Graph state* – a highly entangled quantum state, is the critical resource for quantum error correction (QEC) [1], measurement-based quantum computing (MBQC) [2], and various other applications. Moreover, it enables distributed quantum computing [3], [4] and quantum networking [5], acting as a paradigm for non-local qubit interactions. For instance, the *Bell state* is a special case of a graph state.

Before graph states can be utilized for various applications, they must first be generated on quantum computers or quantum network nodes. Graph state generation has predominantly focused on the *probabilistic scheme* [6], which relies on a sequence of *fusion gates* and photon interference. This all-photonic approach is widely studied for its relevance in optical quantum computing. Several previous compilation studies [7], [8], [9] for photonic MBQC are rooted in the *probabilistic scheme*. However, such approach suffers from an exponential demand for resources, severely limiting the scalability of the target graph states. Additionally, the randomness and uncertainty associated with fusion gates significantly increase the complexity of the compilation algorithms.

An alternative hardware solution is the *deterministic scheme*, where *photon qubits* are directly generated by emitters [10]. In this scheme, a set of interacting *emitter qubits* is employed to induce entanglement among photon qubits as they are emitted. Owing to better scalability and advancements in engineering, the *deterministic scheme* is attracting increasing attention. Emitter-based graph states have been demonstrated across various hardware platforms, including silicon quantum dots [11], [12], nitrogen vacancy (NV) color centers [13], [14], and Rydberg atoms [15], achieving scales of up to 14 qubits [16].

The generation process of an emitter-based graph state could be formalized by a quantum circuit model. Constructing the quantum circuit for the generation of a specific graph state can be referred to as the **GraphState-to-Circuit compilation** — which is the problem we focus on in this paper. Such a compilation task has distinct constraints compared to previous quantum compilers designed for superconducting [17], [18], [19], [20], [21], [22], trapped-ion[23], all-photonic [7], Bosonic quantum [24], or neutral atom [25] platforms. Shown in Figure 1.a, the circuit model for graph state generation [26], [27] contains two types of qubits: *emitter qubit* and *photon qubit*. Entanglement is created via emitter-emitter interactions (e.g., emitter-emitter CNOTs), and the entanglement connections are transduced to photons through the photon emission process. However, for the purpose of being *deterministic*, photon-photon interactions are not allowed, which significantly differentiates this approach from the all-photonic *probabilistic scheme*. Moreover, the gates for different types of qubits are implemented based on completely different physical operations, leading to significantly varying gate durations. These particular constraints necessitate a specific compiler.

A series of research efforts attempt to address the compilation problem for emitter-based graph state generation. Li et al. [26] propose a method to minimize the emitter resources required for graph state generation. However, their approach does not consider circuit depth and suffers from photon loss issues due to the prolonged hardware operations of emitter CNOTs. Kaur et al. [28] explore loss-aware generation for repeater graph states (RGS), but their method does not fully exploit resource utilization for arbitrary graph states. Ghanbari et al. [27] leverage *local complementation* to optimize graph states and reduce generation overhead. Nevertheless, finding the optimal *local complementation* is #P-complete [29], which limits the scalability of this optimization. Furthermore, none of these works provide a feasible compilation scheme for large-scale graphs. Hindered by high complexity and an exponential solution space, optimal solutions for random graph states can only be achieved at the scale of a few qubits.

To address the above limitations, we propose a novel compilation framework for the graph state generation task, adopting a divide-and-conquer strategy to tackle the scalability problem. First, we partition the graph state into subgraphs, enabling optimal compilation for each subgraph with low overhead, while integrating *local complementation* for optimization. Next, we compile the subgraphs based on a comprehensive objective function that accounts for the characteristics of actual quantum computing hardware. Finally, during subgraph recombination, we design a circuit scheduling scheme to exploit emitter reuse among subgraphs, thereby maximizing qubit

* Corresponding to: Zhiding Liang (liangz9@rpi.edu) and Xiangyu Ren (xiangyu.ren@ed.ac.uk).

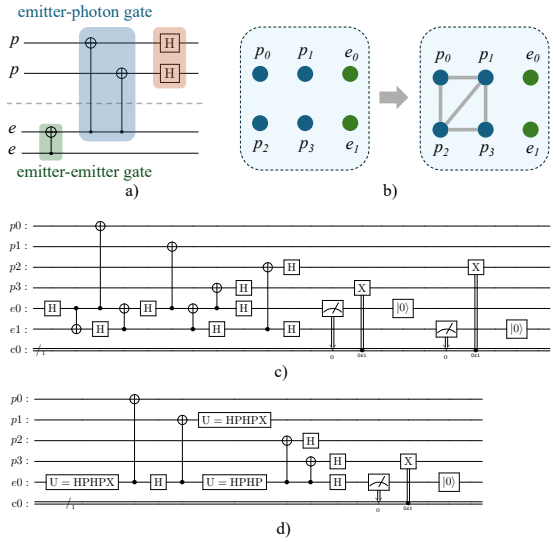


Fig. 1: The background on GraphState-to-Circuit Compilation. a) The constraints of a *deterministic* graph state generation, with p as photon and e as emitter. For deterministic purpose, only emitter-emitter CNOT in green and emitter-photon CNOTs (emissions) in blue are allowed. b) Generation of a graph state on photon qubits p_0 - p_3. c) The circuit for generating the graph state in b), as a result of compilation. d) An optimized circuit for the same graph state, with fewer emitters utilization and fewer CNOTs engaged compared to c).

resource utilization and reducing photon loss accumulation during generation. Our framework is evaluated on various types of graph states and compared with the best-known method [30]. It outperforms the baseline, achieving on average a 30% (up to 52%) reduction in emitter-emitter CNOTs, a 38% (up to 56%) reduction in circuit duration, and a suppression of photon loss by $\times$1.4 (up to $\times$1.9).

Our key contributions are as follows:

1) We are the first to study emitter-based graph state generation from a quantum compiler perspective. We formulate the compilation constraints, perform a comprehensive analysis of previous methods, and identify the objectives to consider based on realistic hardware implementations.
2) We propose a scalable compilation method by partitioning the target graph state into subgraphs and compiling these subgraphs. This method incorporates *local complementation* to reduce the critical overhead – emitter-emitter CNOT.
3) We design a scheduling scheme that maximizes the utilization of emitter qubit resources. It minimizes circuit duration, thereby reducing the accumulation of photon loss.

II. Background

A. Quantum Graph State

A quantum graph state $|G\rangle$ is a specific type of multipartite entanglement state that is intrinsically associated with a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges connecting these vertices. In graph G, a vertex v represents a physical qubit, whereas an edge e indicates the entanglement between the two qubits. The formal definition of a graph state $|G\rangle$ is given by:

$$|G\rangle = \prod_{(i,j)\in E} CZ_{(i,j)} \, |+\rangle^{\otimes |V|} \tag{1}$$

Here, $|+\rangle^{\otimes |V|}$ denotes the tensor product of $|V|$ copies of $|+\rangle$ state at all the vertices of G, and $CZ_{(i,j)}$ is the controlled-Z gate operation applied to the pair of qubits corresponding to the edge $(i, j) \in E$.

Typically, the generation of a graph state is achieved through two steps: 1) Prepare a set of qubits in initial states, corresponding to all the vertices in V; 2) For each pair of qubits where the entanglement is desired, apply a CNOT/CZ interaction. Note that for a graph state, all CNOT/CZ gates are commutable, allowing them to be arranged in arbitrary sequences or applied in parallel, providing abundant flexibility for circuit scheduling. Here we provide an example of graph state, shown on the right side of Figure 1.b), representing the entanglement among photon qubits p_0, p_1, p_2, p_3.

B. Emitter-Based Graph State Generation

Problem Formulation. Here we provide a description of the circuit model for emitter-based graph state generation, including all elementary operations required for the process, as illustrated in Figure 1.a-c, and an example of optimization in Figure 1.d:

1) Two types of qubits are involved: *emitter qubit* and *photon qubit*. Emitter qubits are initialized to state $|0\rangle$, while photon qubits do not exist physically before emission and are represented as $|0\rangle$.
2) Entanglement is induced by emitter-emitter CNOT, and the entanglement connection are transduced to emitted photon via emission process.
3) The emission of a photon is modeled by an emitter-photon CNOT between the emitter and emitted photon, being always the first gate applied on each photon qubit.
4) For *deterministic* purpose, the photon-photon interactions are not allowed. Only single-qubit gates and measurement can be applied on emitted photon qubits.
5) The final result should be a graph state entangled across n photons (n = size of the graph), while all emitters return to $|0\rangle$ state upon completion.

Hardware Characteristic. The emission and entanglement operations are implemented differently across hardware platforms. Here we use the quantum dots platform [10], [31] as an example. The emitter-emitter CNOT is realized through electron exchange coupling between quantum dots, with a gate time of 1-10 ns and a fidelity $\geq$0.99. In contrast, the emitter-photon CNOT (emission) has a significantly shorter duration of $\sim$0.1 ns. Given this distinction in gate time, circuit scheduling for emitter-based graph state generation becomes a non-trivial problem.

C. Time-Reversed Model for Graph State Generation

Our compilation algorithm is built upon a **time-reversed model** introduced in previous work [26]. In this time-reversed context, photon emission is replaced with a hypothetical "photon absorption", while emitter-emitter interaction correspond to "disentangling" between emitters. Using these reversed operations, we can derive

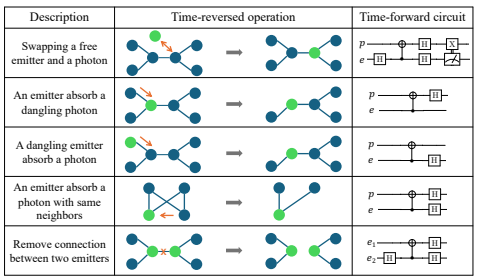


Fig. 2: Operations of the time reversed graph reduction model.

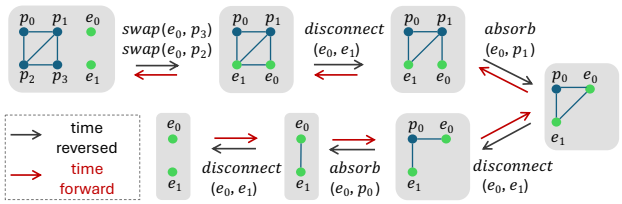


Fig. 3: An example of the time-reversed reduction sequence (black arrows), corresponding to the circuit in Figure 1.c (red arrows).

a virtual sequence that reduces a target graph state to initial state. Hence, reversing this sequence yields the required circuit to generate target state from initial state. We specify the following reversed operations [28], which are also illustrated in Figure 2:

a) Swapping with free emitter: An arbitrary photon in the graph state can be replaced by a free emitter in $|0\rangle$ state.

b) Photon absorption: If an emitter is entangled with a photon in the graph, it can absorb the photon under certain conditions. The specific conditions for photon absorption are illustrated in Figure 2.

c) Emitter disentangle: Remove the entanglement connection (represented as an edge in the graph) between two emitters.

Based on these operations, our compiler determines the optimal time-reversed reduction sequence of target graph state (e.g. Figure 3), then reverses this sequence to generate the desired circuit. We recommend Kaur et al. [28] for a more comprehensive background.

D. Optimizing Graph State via Local Complementation

By applying local Clifford (LC) gates to a graph state, it can be transformed into a set of alternative graph states that are LC-equivalent with original graph state. Hence, generation of the target graph state can be redefined as generating an LC-equivalent graph state. If the LC-equivalent graph has fewer edges, this transformation can reduce the entanglement overhead, with the only cost being the addition of single-qubit gates to the circuit.

Finding LC-equivalent graph state can be formalized as applying *local complementation* to the graph [27], [29]. Specifically, local complementation of a vertex v in the graph modifies the edges within the neighborhood on v. In the neighborhood of v, edges that are present will be removed, and edges that are missing will be added (Figure 4). A proper local complementation of target graph state can significantly reduce the entanglement overhead. However, determining the optimal local complementation has been proven to be #P-complete [32], making it infeasible to solve in polynomial time.

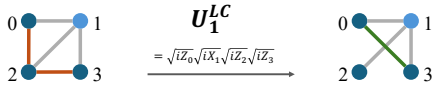


Fig. 4: Leveraging local complementation to optimize the graph state. In this example, a Clifford unitary is applied on photon 1 (U_1^{LC}), leading to a local complementation of the corresponding vertex 1, acting on neighborhood vertices 0, 2, 3.

III. Motivation

A. Challenges and Solution

Given the constraints and optimization opportunities, GraphState-to-Circuit compilation is a complex and non-trivial task. In this section, we discuss the associated challenges and propose our solution.

Challenge 1: There is an exponential overhead in searching the optimal circuit of graph state generation. Additionally, finding the best local complementation for target graph state is a #P-complete problem. As a result, the compilation is **not scalable**.

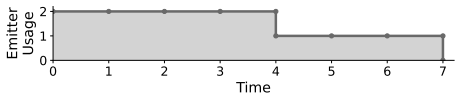


Fig. 5: The emitter usage number over time (timescale simplified), for a graph state generation circuit.

As we mentioned in Section II.A, the commutation of CNOT/CZ in graph state generation allows photon emission in an arbitrary sequence. As a result, the compilation overhead grows exponentially when exhaustively searching all possible circuits. For the state-of-the-art compiler GraphiQ [30], the runtime exceeds 10^3 seconds when solving a linear cluster state with more than 10 qubits. Furthermore, when realistic optimization targets such as circuit duration and loss rate are considered instead of simply minimizing the #CNOT, the runtime becomes even worse.

Solution: We partition the target graph state into several subgraphs, each with a feasible size for finding the optimal generation circuit. These subgraphs are treated as *leaves* and are compiled separately with designated optimization. Meanwhile, those inter-subgraph entanglements are treated as *stems*, compiled into an independent circuit and integrated into the final result.

Challenge 2: In hardware-aware scenarios, there are **various factors affecting the quality** of compiled circuits.

Based on the hardware characteristic of quantum dots emitter in Section II.B, several optimization targets must be considered: 1) A limited number of emitters, leading to prolonged circuit execution; 2) Accumulation of photon loss over time, which may render photons unusable; 3) Imperfect emitter-emitter CNOTs, affecting fidelity of the target state. Previous research has only considered one of these factors [26], resulting in impractical outcomes.

Solution: We construct a cost function that combines multiple optimization objectives. Specifically, we extract the gates duration based on hardware settings, and use them to calculate the photon loss duration of generation circuit as the cost function.

Challenge 3: When an emitter is entangled in the graph state, it may remain idle due to the lack of available operations. This results in **insufficient utilization of emitters**, thereby increasing the circuit duration.

Figure 5 provides an example of the emitter usage curve of a circuit, illustrating the number of utilized emitters during the generation process. It can be observed that, for certain parts of the circuit, the utilization of emitters does not reach the maximum capacity. In other words, the emitter resources are not fully utilized. This under-utilization becomes more pronounced as the scale of the target graph state increases, resulting in a generation circuit with significantly longer duration.

Solution: We schedule the subgraphs circuits by arranging them on the timeline. Based on the emitter utilization curve of each subgraph, we can share emitter qubits among different subgraphs, ensuring maximum utilization of emitters at every time slot. Briefly, we exploit the circuit-level parallelism and emitter reuse.

B. Insight: Divide-and-Conquer

The framework we propose adopts a *divide-and-conquer* strategy, inspired by the natural *stem-leaf* structure of a tree graph, but also applies to arbitrary graph structure. A straightforward illustration of this *stem-leaf* structure can be find in the left part of Figure 6. By partitioning the target graph state into several subgraphs [33],

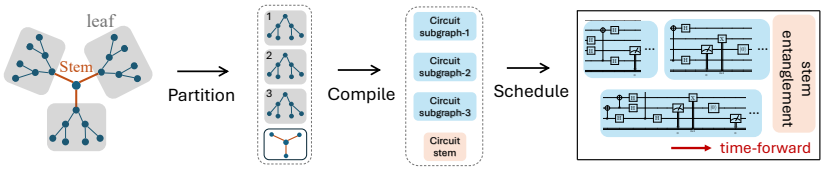


Fig. 6: The overview of our framework.

we treat the entanglements within each *leaf* as sub-problems, and finally recombine them through the compilation of inter-subgraph entanglements as the *stem*. The overview of framework is in Figure 6.

Our insight within this strategy is that finding the global optimal solution of graph state generation circuit is highly complex. However, optimizing each sub-problem individually can reduce overhead and achieve a local-minimum. Another challenge is determining an appropriate partitioning scheme for any graph state, which will be discussed in Section IV.A. Additionally, our circuit scheduling approach reduces redundancy by treating each subgraph as a coarse-grained basic unit.

IV. Framework Design & Implementation

A. Graph State Partitioning

We present the design of the first part in our compilation framework, where the target graph state is partitioned into subgraphs. To find an appropriate partitioning scheme, the following considerations should be addressed:

Reduce interconnections among subgraphs. The target graph state is partitioned into subgraphs, and each subgraph is compiled with lower runtime overhead. After obtaining the optimal generation circuit for each subgraph, they need to be connected via inter-subgraph entanglements (edges). In the resulting circuit that combines all subgraphs, these inter-subgraph entanglements will be compiled into emitter-emitter CNOTs, which are expensive operations on the hardware. Thus, the objective of graph state partitioning is to minimize the number of interconnection edges among different subgraphs.

Optimize graph partitioning with local complementation. Taking the graph state in Figure 7 as an example, normally at least 5 interconnecting edges must be cut to divide it into two subgraphs, shown in (a). However, by applying local complementation on vertices α and β, only 1 interconnecting edge remains for the partitioning, shown in (b). Thus, finding the appropriate local complementation is crucial not only for reducing the overall number of edges, but also for minimizing the partitioning cost. Due to the exponential search space for local complementations, we limit the number of LC operations to l, resulting in a depth-limited search. This restriction not only simplifies the search process, but also limits the additional circuit depth introduced by the local complementation operations.

Graph state partition searcher with depth-limited LC. We leverage a mix integer programming (MIP) solver to simultaneously find the best local complementation and graph partitioning solution. Our MIP model is described as follows:

a) Constant Parameters: Given the target graph state $G = (V, E)$, the vertices are represented as $v \in V$ and the edges as $e \in E$. Assuming the number of local complementation (LC operations) is limited to l, we define a sequence $1, 2, \ldots, l$ to mark the status of the graph state at each step. The subgraphs partitioned from G are represented as the set $\{g\}$, with size limitation $|g| \leq g_{max}$. Additionally, the total number of subgraphs is given by $\lceil |G|/g_{max} \rceil$.



Fig. 7: LC optimization for graph state partitioning.

b) Variables: First, we define our edge status in each step t by $e_{v_1,v_2,t}$ $(t \in 1, \ldots, l)$, and vertices in each subcircuit by $y_{v,g}$:

$$e_{v_1,v_2,t} = \begin{cases} 1, \text{ if vertex } v_1, v_2 \text{ are adjacent at step } t \\ 0, \text{ otherwise} \end{cases}$$

$$y_{v,g} = \begin{cases} 1, \text{ if vertex } v \text{ is partitioned in subgraph } g \\ 0, \text{ otherwise} \end{cases}$$

Then, we set the local complementation in each step as $x_{v,t}$.

$$x_{v,t} = \begin{cases} 1, \text{ if local complementation applied on } v \text{ at step } t \\ 0, \text{ otherwise} \end{cases}$$

c) Constraints: For local complementation, we only apply it on single vertex at each step (or skip this step), thus we have:

$$\sum_{v \in V} x_{v,t} \leq 1, \quad \forall t \in \{1, ..., l\} \tag{2}$$

Most importantly, the local complementation at step t will add or remove the edges, resulting in a change at step $t+1$. Suppose the local complementation is applied to vertex v_0 ($x_{v_0,t} = 1$), and v_1, v_2 being two of its adjacent vertices ($e_{v_0,v_1,t} = e_{v_0,v_2,t} = 1$). Then, there should be a change to the edge e_{v_1,v_2} from step t to $t+1$:

$$x_{v_0,t} * e_{v_0,v_1,t} * e_{v_0,v_2,t} \leq |e_{v_1,v_2,t} - e_{v_1,v_2,t+1}| \tag{3}$$

In addition, the size of each subgraph is restricted by:

$$\sum_{v \in V} y_{v,g} \leq g_{max}, \quad \forall g \in \{g\} \tag{4}$$

d) Objective Function: Since we hope to minimize the interconnecting edges among subgraphs, our objective function is the number of such edges, represented by K. We use $\sum_{\forall g \in \{g\}} y_{v_1,g} * y_{v_2,g}$ to determine if v_1, v_2 are in the same subgraph, then we have:

$$K = \sum_{\forall v_1, v_2 \in V} e_{v_1,v_2,l} * (1 - \sum_{\forall g \in \{g\}} y_{v_1,g} * y_{v_2,g}) \tag{5}$$

Overall, the MIP model can be formulated as

$$minimize\ objective\ \ K(Eq.\ 5)$$
$$s.t.\ constraint\ Eqs.\ (2,\ 3,\ 4)$$

B. Optimizing Circuit Depth for Subgraph Generation

Optimizing Targets. In this section, we introduce our compilation method for the subgraphs. Unlike previous works that focus on optimizing either the minimal emitter resource [26] or #CNOT [30], we propose a novel compilation method emphasizing the suppression of photon loss, informed by the practical constraints of current hardware platforms. Our optimization targets are outlined below:

a) Number of emitter-emitter CNOTs: Considering gates durations on hardware, emitter-emitter CNOTs are typically more challenging to implement than other operations. As discussed in the quantum dot example in Section II.B, the emitter-emitter CNOT time is more than $10\times$ longer than that of the emitter-photon CNOT, constituting the primary contributor to the overall circuit duration.

b) Photon loss duration: Another critical problem in graph state generation is the loss of photon qubits. From the moment a photon is emitted until the entire graph state is fully generated, the photon accumulates loss, reducing the fidelity of the final graph state. Minimizing the existing time of photons can suppress the probability of photon loss. Specifically, given two circuits with the same depth (duration), we prefer the circuit that delays photon emission operations as much as possible.

Compilation Strategy. Based on the time-reversed model, our compilation of each subgraph involves the following steps:

① We search for the graph state reduction sequence, utilizing the reduction operations detailed in Section II.C, considering the maximum number of available emitter ne_{limit}. A heuristic-based depth-first searching is performed to find reduction sequences requiring the minimal number of *emitter disentangle* operations, which corresponds to circuits with minimal emitter-emitter CNOTs. The degree of vertices is used as heuristic, prioritizing the reduction of lower-degree vertices to ensure that more entanglements are created by photon emissions rather than emitter-emitter CNOTs. Additionally, we truncate the operation sequences with a high number of emitter-emitter CNOTs to reduce the searching overhead.

② After obtaining a set of candidate sequences with minimal number of emitter-emitter CNOTs, these sequences are translated into circuits, and the one with the minimal photon loss duration is selected. Specifically, the average photon loss duration ($\overline{T}_{loss}$) is calculated by:

$$\overline{T}_{loss} = \frac{1}{n} \sum_{p=p_1,\ldots,p_n} M_{circ_end} - M_{emit}(p)$$

where $M_{\text{circ_end}}$ represents the circuit ending time, and $M_{\text{emit}}(p)$ represents the emission time of photon p.

Flexible Resource Constraint. We adopt a flexible resource constraint, enabling the generation of optimal circuits under varying numbers of available emitters. For a subgraph g, the minimal number of required emitters, ne_{min}, is calculated using entanglement entropy theory [26]. The subgraph is then compiled with different available emitter limits: $ne_{\text{limit}} = ne_{\text{min}}, ne_{\text{min}}+1, ne_{\text{min}}+2$, respectively, to generate circuits for each scenario. This approach enhances parallelism within the subgraph when additional emitter qubits are available, facilitating qubit reuse, which will be discussed further in the next subsection.

C. Subgraph Recombination & Circuit Scheduling

Finally, we combine the generation circuits corresponding to each subgraph and schedule these subcircuits. In this step, inter-subgraph entanglements are added to the overall circuit to connect all components. This process is illustrated in Figure 8.

Circuit Scheduling Strategy. Our scheduling scheme adopts the *as-late-as-possible* strategy from Qiskit [34], appending subgraph circuits starting from the latest possible time and working gradually backward. When selecting circuits to schedule, we prioritize subcircuits with more photon qubits and shorter execution times (placing them later), and deprioritize subcircuits with fewer photon qubits and longer execution times (placing them earlier). The priority coefficient for a subgraph circuit is calculated as: $P_c = \frac{n_p^c}{T_c}$, where n_p^c is the number of photons in it, and T_c is its duration. This strategy ensures that more photons are emitted as late as possible, minimizing the time lasting for photon loss.

Emitter Qubit Reuse. The timestamp for a subgraph circuit c is determined based on its emitter qubit usage curve. When inserting c into the timeline of overall circuit C, we shift c as late as possible, while ensuring that the total emitter usage in C does not exceed the maximum emitter limit Ne_{limit} at any moment. This approach is analogous to playing Tetris, where the pieces (subcircuits c) are shifted downward (later in time) as far as possible, while being constrained by the walls (the Ne_{limit}), as shown in Figure 8.

Full Utilization of Emitter Qubits. To maximize the circuit parallelism in Ne_{limit} constraint, we aim to utilize any remaining (unused) emitter qubits at each time slot. This approach works in conjunction with the *flexible resource constraint* policy in Section IV.B. For example, if no additional subgraph circuit c can fit within a given time period, but the total emitter usage does not fully reach Ne_{limit}, we relax the ne_{limit} constraint from ne_{min} to $ne_{\text{min}}+1$ for the circuits already scheduled in that period. This ensures full utilization of emitter qubit resources, ultimately reducing circuit duration.

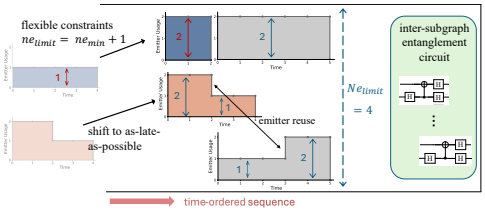


Fig. 8: Our scheduling scheme. The blocks are emitter usage curve for each subgraph, and the width is restricted by total emitters Ne_{limit}.

V. Evaluation

We evaluate our framework on various benchmarks using several hardware-realistic metrics and compare our results with the state-of-the-art graph state generation framework, GraphiQ [30].

A. Experimental Setup

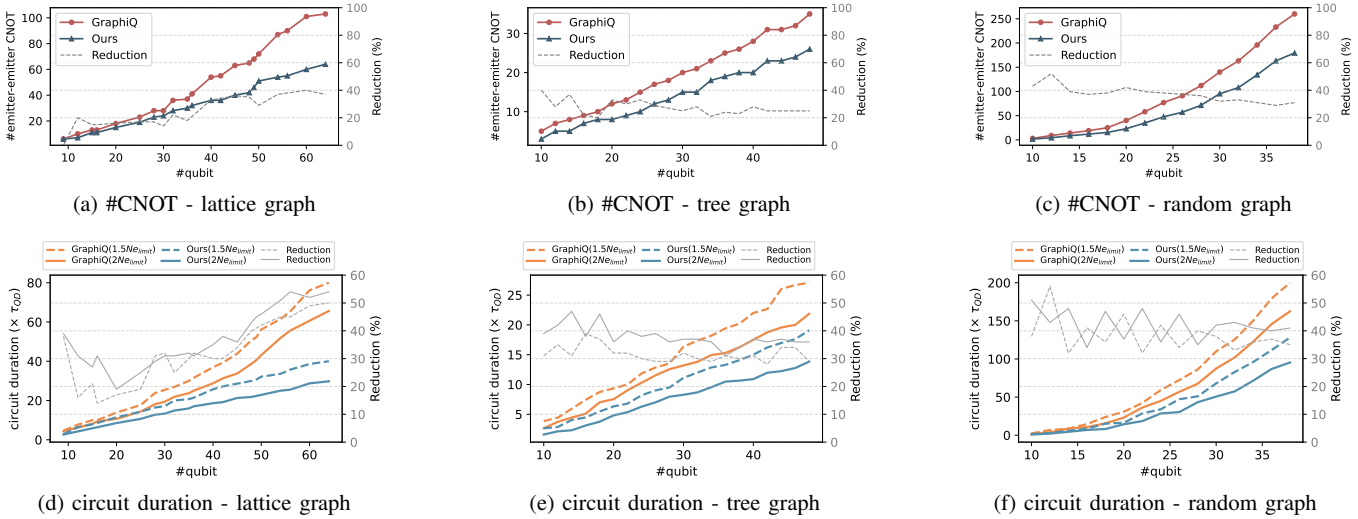
Benchmarks. The graph states we tested (Figure 9) are selected from popular quantum applications: 1) *Lattice graph* is a two-dimensional (2D) square grid structure, which is basic element for MBQC [8]; 2) *Tree graph* is a connected acyclic graph, which is particularly useful as it constitutes the quantum routers in quantum random access memory (QRAM) [35], while it's also vital to the tree code [36] in quantum error correction; 3) *Random graph* is generated by the Waxman graph structure [37], which covers most of the possible communication topologies for distributed quantum computing and quantum network [38].

Experiment Platform. All compilation and simulation are executed on a CentOS 7 server featuring dual Intel Xeon 8255C CPUs (48 logical cores) and 384GB memory. The Python version is 3.10.0. Our framework is built based on GraphiQ[30] and Qiskit [34].

Hardware Model. We select the silicon quantum dot (QD) emitter [10] as our hardware model for simulation. However, it can be easily adapted to other hardware platform, e.g. SiV color centers [39], NV color centers [14] and Rydberg atoms [15], just by changing the configurations of gate characteristic. In QDs hardware, an emitter-emitter CNOT is realized by electron exchange coupling between two QDs, with coefficient J as the exchange interaction strength. Within a time period $\tau_1 = \pi/2J$, the interaction Hamiltonian yields a $\sqrt{SWAP}$ gate, enabling the entangling between emitters. Then, a emitter-emitter CNOT consists of two $\sqrt{SWAP}$ interleaved by two $R_x \otimes R_x$, while each of them take τ_1 as evolution time. Hence, we



Fig. 9: The types of graph in our benchmarks.



(a) #CNOT - lattice graph (b) #CNOT - tree graph (c) #CNOT - random graph

(d) circuit duration - lattice graph (e) circuit duration - tree graph (f) circuit duration - random graph

Fig. 10: Evaluation: reduction of emitter-emitter CNOTs and circuit duration.

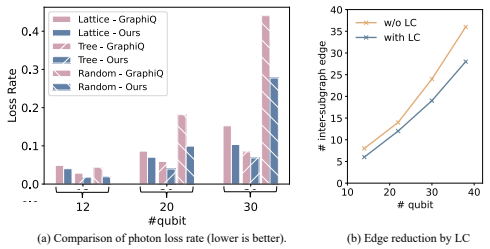
represent the period of emitter-emitter CNOT as $\tau_{QD} = 4\tau_1 = 2\pi/J$. The timescale of τ_{QD} depends on J in hardware system, while it is commonly set to $2\pi \times$ 1GHz and thus $\tau_{QD} =$ 1ns. On the other hand, we set the time period for photon emission as $0.1\tau_{QD}$ based on cavity enhancement [40].

Algorithm Configuration & Baseline. In our framework, we use Gurobi [41] as the MIP solver for graph state partitioning. For the graph partitioning search algorithm in Section IV.A, we set a 20-minute timeout for MIP solver, using the best solution found within this time. In MIP model, we limit the size of subgraph $g_{max} = 7$ and maximum length of LC sequence to $l = 15$. Note that the compilation runtime for each subgraph is relatively low (< 100 seconds), owing to the constrained subgraph size. For baseline GraphiQ, we select its *AlternateTargetSolver* and modify it to include a 30-minute timeout, instead of searching exhaustively.

B. Result

1) # Emitter-emitter CNOTs: As the emitter-emitter CNOT being a critical component of graph state generation circuit, we compare its count in the compilation of graph state in varying sizes. The results are shown in Figure 10.(a)-(c) for three types of graphs. Our compilation framework consistently reduces the #CNOT compared to GraphiQ across all graph states, with average reductions of 25%, 28%, 37% (maximum 40%, 39%, 52%) for lattice, tree, and random graphs, respectively. Furthermore, as the size of the graph state increases, the reduction rate grows for lattice graphs and remains steady for the others, demonstrating the scalability of our method.

2) Circuit Duration: The time unit we use for calculating duration is τ_{QD} introduced in last subsection, with its realistic value dependent on hardware. We compare circuit durations under two emitter resource settings: $Ne_{limit} = 1.5Ne_{min}$ and $Ne_{limit} = 2Ne_{min}$ (Ne_{min} is the minimal #emitter required for target graph).



(a) Comparison of photon loss rate (lower is better). (b) Edge reduction by LC

Fig. 11: Evaluation: photon loss suppression and LC optimization.

Our results in Figure 10.(d)-(f) show that our compilation framework reduces circuit duration significantly. For $Ne_{\text{limit}} = 1.5Ne_{\text{min}}$, the average reduction percentages are 33%, 32%, and 39% (maximum 50%, 39%, and 56%). For $Ne_{\text{limit}} = 2Ne_{\text{min}}$, the average reductions are 38%, 38%, and 43% (maximum 54%, 47%, and 51%).

For the same graph state, circuit duration reduction rates are consistently higher than #CNOT reduction rates. Additionally, the duration reduction rate generally increases when the emitter resource setting Ne_{limit} is raised from $1.5Ne_{\text{min}}$ to $2Ne_{\text{min}}$. This highlights the benefits of circuit parallelism, with its impact becoming more pronounced as more emitter resources are made available.

3) Photon Loss: We assess the photon loss rate of final graph state generated by circuit, based on realistic hardware settings. Given the electron spin coherence time $T_2 \sim$ 1s, we set the photon loss rate to 0.5% per time period τ_{QD}. Our simulations ($Ne_{limit} = 1.5Ne_{min}$) are performed using GraphiQ, and the results are given in Figure 11.(a). Our framework improves the baseline by $\times$1.3, $\times$1.4, and $\times$1.9 on average for three types of graph, respectively.

4) Local Complementation: We compare the average number of inter-subgraph edges with and without local complementation optimizing in Figure.11.(b) (by setting LC sequence $l = 15$ or 0), to illustrate the benefits of local complementation. In this experiment, we use random (Waxman) graphs as the benchmark.

VI. Conclusion

We propose a novel compilation framework for emitter-photonic graph state generation, leveraging a divide-and-conquer strategy. The framework partitions the graph state into subgraphs, compiles them separately, and then recombines them using circuit scheduling. Tested on various graph states, our framework shows reductions in #CNOT and circuit duration, taking hardware constraints into account.

Acknowledgment

We thanks Liang Jiang for helpful discussion and we acknowledge support from the ARO MURI (W911NF-21-1-0325) and NSF (ERC-1941583). Also, we thanks the reviewers of DAC25 for their invaluable feedback.

References

[1] D. Schlingemann and R. F. Werner, "Quantum error-correcting codes associated with graphs," Physical Review A, vol. 65, no. 1, p. 012308, 2001.

[2] R. Raussendorf and H. J. Briegel, "A one-way quantum computer," Physical review letters, vol. 86, no. 22, p. 5188, 2001.

[3] K. Liu, Y. Zhou, H. Luo, L. Xiong, Y. Zhu, E. Casey, J. Cheng, S. Y.-C. Chen, and Z. Liang, "Ecdqc: Efficient compilation for distributed quantum computing with linear layout," arXiv preprint arXiv:2410.23857, 2024.

[4] D. Ferrari and M. Amoretti, "A simulation framework for distributed quantum computing," arXiv preprint arXiv:2306.11539, 2023.

[5] C. D. Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller et al., "Design and demonstration of an operating system for executing applications on quantum network nodes," arXiv preprint arXiv:2407.18306, 2024.

[6] D. E. Browne and T. Rudolph, "Resource-efficient linear optical quantum computation," Physical Review Letters, vol. 95, no. 1, p. 010501, 2005.

[7] H. Zhang, J. Ruan, H. Shapourian, R. R. Kompella, and Y. Ding, "Oneperc: A randomness-aware compiler for photonic quantum computing," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 738–754. [Online]. Available: https://doi.org/10.1145/3620666.3651372

[8] Y. Li, A. Pawar, Z. Mo, Y. Zhang, J. Yang, and X. Tang, "Minimizing photonic cluster state depth in measurement-based quantum computing," arXiv preprint arXiv:2312.10865, 2023.

[9] Z. Mo, Y. Li, A. Pawar, X. Tang, J. Yang, and Y. Zhang, "Fcm: A fusion-aware wire cutting approach for measurement-based quantum computing," in Proceedings of the 61st ACM/IEEE Design Automation Conference, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657346

[10] A. Russo, E. Barnes, and S. E. Economou, "Photonic graph state generation from quantum dots and color centers for quantum communications," Physical Review B, vol. 98, no. 8, p. 085303, 2018.

[11] Y. Arakawa and M. J. Holmes, "Progress in quantum-dot single photon sources for quantum information technologies: A broad spectrum overview," Applied Physics Reviews, vol. 7, no. 2, 2020.

[12] D. Cogan, Z.-E. Su, O. Kenneth, and D. Gershoni, "Deterministic generation of indistinguishable photons in a cluster state," Nature Photonics, vol. 17, no. 4, pp. 324–329, 2023.

[13] K. Nemoto, M. Trupke, S. J. Devitt, A. M. Stephens, B. Scharfenberger, K. Buczak, T. Nöbauer, M. S. Everitt, J. Schmiedmayer, and W. J. Munro, "Photonic architecture for scalable quantum information processing in diamond," Physical Review X, vol. 4, no. 3, p. 031022, 2014.

[14] H. Choi, M. Pant, S. Guha, and D. Englund, "Percolation-based architecture for cluster state creation using photon-mediated entanglement between atomic memories," npj Quantum Information, vol. 5, no. 1, p. 104, 2019.

[15] C.-W. Yang, Y. Yu, J. Li, B. Jing, X.-H. Bao, and J.-W. Pan, "Sequential generation of multiphoton entanglement with a rydberg superatom," Nature Photonics, vol. 16, no. 9, pp. 658–661, 2022.

[16] P. Thomas, L. Ruscio, O. Morin, and G. Rempe, "Efficient generation of entangled multiphoton graph states from a single atom," Nature, vol. 608, no. 7924, pp. 677–681, 2022.

[17] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems, 2019, pp. 1001–1014.

[18] S. S. Tannu and M. Qureshi, "Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 253–265.

[19] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems, 2019, pp. 987–999.

[20] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via maxsat," in 2022 55th IEEE/ACM international symposium on Microarchitecture (MICRO). IEEE, 2022, pp. 1078–1091.

[21] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1031–1044. [Online]. Available: https://doi.org/10.1145/3297858.3304018

[22] X. Ren, T. Zhang, X. Xu, Y.-C. Zheng, and S. Zhang, "Invited: Leveraging machine learning for quantum compilation optimization," in Proceedings of the 61st ACM/IEEE Design Automation Conference, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3663510

[23] W.-H. Tseng, Y.-W. Chang, and J.-H. R. Jiang, "Satisfiability modulo theories-based qubit mapping for trapped-ion quantum computing systems," in Proceedings of the 2024 International Symposium on Physical Design, 2024, pp. 245–253.

[24] J. Zhou, Y. Liu, Y. Shi, A. Javadi-Abhari, and G. Li, "Bosehedral: Compiler optimization for bosonic quantum computing," in 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), 2024, pp. 261–276.

[25] H. Wang, P. Liu, D. B. Tan, Y. Liu, J. Gu, D. Z. Pan, J. Cong, U. A. Acar, and S. Han, "Atomique: A quantum compiler for reconfigurable neutral atom arrays," in 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA). IEEE, 2024, pp. 293–309.

[26] B. Li, S. E. Economou, and E. Barnes, "Photonic resource state generation from a minimal number of quantum emitters," npj Quantum Information, vol. 8, no. 1, p. 11, 2022.

[27] S. Ghanbari, J. Lin, B. MacLellan, L. Robichaud, P. Roztocki, and H.-K. Lo, "Optimization of deterministic photonic graph state generation via local operations," arXiv preprint arXiv:2401.00635, 2024.

[28] E. Kaur, A. Patil, and S. Guha, "Resource-efficient and loss-aware photonic graph state preparation using an array of quantum emitters, and application to all-photonic quantum repeaters," arXiv preprint arXiv:2402.00731, 2024.

[29] T. Ji, J. Liu, and Z. Zhang, "Distributing arbitrary quantum cluster states by graph transformation," arXiv preprint arXiv:2404.05537, 2024.

[30] J. Lin, B. MacLellan, S. Ghanbari, J. Belleville, K. Tran, L. Robichaud, R. G. Melko, H.-K. Lo, and P. Roztocki, "Graphiq: Quantum circuit design for photonic graph states," arXiv preprint arXiv:2402.09285, 2024.

[31] M. Gimeno-Segovia, T. Rudolph, and S. E. Economou, "Deterministic generation of large-scale entangled photonic cluster state from interacting solid state emitters," Physical review letters, vol. 123, no. 7, p. 070501, 2019.

[32] A. Dahlberg, J. Helsen, and S. Wehner, "Counting single-qubit clifford equivalent graph states is# p-complete," Journal of Mathematical Physics, vol. 61, no. 2, 2020.

[33] X. Ren, M. Zhang, and A. Barbalace, "A hardware-aware gate cutting framework for practical quantum circuit knitting," arXiv preprint arXiv:2409.03870, 2024.

[34] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

[35] S. Xu, C. T. Hann, B. Foxman, S. M. Girvin, and Y. Ding, "Systems architecture for quantum random access memory," in Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023, pp. 526–538.

[36] K. Azuma, K. Tamaki, and H.-K. Lo, "All-photonic quantum repeaters," Nature communications, vol. 6, no. 1, pp. 1–7, 2015.

[37] B. M. Waxman, "Routing of multipoint connections," IEEE journal on selected areas in communications, vol. 6, no. 9, pp. 1617–1622, 1988.

[38] Y. Huang, X. Ren, B. Li, Y. Wong, and L. Jiang, "Space-time peer-to-peer distribution of multi-party entanglement for any quantum network," arXiv preprint arXiv:2412.14757, 2024.

[39] P.-J. Stas, Y. Q. Huan, B. Machielse, E. N. Knall, A. Suleymanzade, B. Pingault, M. Sutula, S. W. Ding, C. M. Knaut, D. R. Assumpcao et al., "Robust multi-qubit quantum network node with integrated error detection," Science, vol. 378, no. 6619, pp. 557–560, 2022.

[40] Y. A. Kelaita, K. A. Fischer, T. M. Babinec, K. G. Lagoudakis, T. Sarmiento, A. Rundquist, A. Majumdar, and J. Vučković, "Hybrid metal-dielectric nanocavity for enhanced light-matter interactions," Optical Materials Express, vol. 7, no. 1, pp. 231–239, 2016.

[41] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

Power-Constrained Printed Neuromorphic Hardware Training

Tara Gheshlaghi[1]♣, Haibin Zhao[1]♣, Priyanjana Pal[1]♣, Michael Hefenbrock[2], Michael Beigl[1], and Mehdi B. Tahoori[1]
[1]*Karlsruhe Institute of Technology,* [2]*RevoAI GmbH*
[1]{tara.gheshlaghi, haibin.zhao, priyanjana.pal, michael.beigl, mehdi.tahoori}@kit.edu, [2]michael.hefenbrock@revoai.de

***Abstract*—With the rising demand for ultra-low-cost and flexible electronics in applications like smart packaging and wearable health monitoring, printed electronics provide an affordable, adaptable, and customizable alternative to conventional silicon. However, these systems often rely on printed batteries or energy harvesters with limited power capacity, making strict power budgets critical. Printed neuromorphic circuits (pNCs) are promising for their analog signal processing, reduced circuit complexity, and energy efficiency in low-power environments. Nonetheless, maintaining robust performance under strict power constraints remains challenging, necessitating advanced optimization techniques. In this work, we propose an augmented Lagrangian approach to enforce task-specific power constraints in pNCs, validated across 13 benchmark datasets. Our method preserves accuracy within strict power budgets while achieving Pareto-optimal power-accuracy trade-offs in a single training run. In contrast, the penalty-based method, which serves as the baseline, requires up to 150 runs per dataset to generate the Pareto front. For low-power scenarios ($\approx 20\%$ of the original power), our method demonstrates a $52\times$ improvement in accuracy-to-power ratio over the baseline. At higher power budgets ($\approx 80\%$ of the original power), it achieves a $59\times$ improvement, maintaining competitive performance. Experimental results demonstrate that our approach achieves 81.82% accuracy with p-tanh activation function (AF) at high power budgets and excels with p-Clipped_ReLU AF under low power constraints. This highlights the computational efficiency and effectiveness of our approach for power-constrained circuit design.**

I. Introduction

Although silicon-based electronics have witnessed continual advancements in power efficiency and transistor density, their adoption in numerous consumer edge applications such as smart packaging [1], smart bandages [2, 3], wearable devices, and other disposable electronics for consumer use [4]–[7] (as shown in Fig. 1[1]) remains a challenge. Even the most affordable traditional silicon-based electronics, such as Application-Specific Integrated Circuits (ASICs) [8] and microcontrollers [9], struggle to achieve the desired cost efficiency and adaptability for these resource-constrained applications. In this context, printed electronics (PE) emerge as a highly flexible [10, 11], biodegradabele [12] and cost-effective solution, through its additive manufacturing processes, thus enabling highly bespoke circuits at ultra-low costs. Furthermore, many of these applications must operate within a prescribed power budget to ensure a sufficient battery life, making energy efficiency [13] as crucial as cost and flexibility. So, by carefully selecting materials, substrates, and various optimization strategies [13], printed devices can also exhibit high flexibility [10, 11] and biocompatibility [12], making them ideal for constrained environments.



Fig. 1. Target application domains of printed electronics: (a) smart fruit package, (b) smart milk carton, (c) smart food packaging, (d) smart bandages.

Incorporating printed computing circuits is essential to equip printed devices with the basic capacity to process sensor data, such as performing classification tasks. Among various computing paradigms, analog computing is often chosen for PE due to its ultra-low cost and high flexibility in these applications. This approach allows for a reduction in device components, thus avoiding the expensive analog-digital converters (ADCs), and simplifies the execution of multiply-accumulate (MAC) operations. Analog neuromorphic circuits [14] stand out as an effective choice due to their expressiveness [15], using simplified circuit primitives, and benefit from machine learning (ML)-driven off-device design and optimization, making them highly suitable for resource-limited PE applications [13]. Moreover, the agile manufacturing process of PE allows highly tailored circuit designs that can meet the specific functional and physical requirements of target applications, reducing material waste, minimizing design iterations, and enabling rapid prototyping, making it ideal for ultra-low-cost, flexible solutions in constrained environments. A key advantage of this approach is the ability to treat critical components of printed analog neuromorphic circuits (pNCs), such as AFs, as learnable parameters, allowing the adaptation of both the shape and hardware realization of AFs, directly linking ML-optimized parameters to physical circuit characteristics.

Low-power electronics design remains a critical focus area due to its potential to extend device runtime and minimize its uses on high-capacity power sources. This is particularly important for portable and edge devices. While traditional electronics has extensively explored low-power solutions [13, 16, 17] these methods generally operate under soft power constraints, where reducing power consumption is desirable but not mandatory. In contrast, many target applications, especially power harvesters, wearable health monitoring devices, operate within strict power and energy envelopes that demand hard constraints arising

♣Authors contributed equally to this work.
[1]Images are generated by the DALL-E AI tool.

from different physical limitations like weight, cost, reliability over extended periods etc. This distinction is pivotal, as soft constraint solutions often fail to meet the rigid power requirements of such applications. In real-world scenarios, strict power constraints are crucial. For example, wearable health devices, such as ECG or glucose monitors, must operate continuously on compact batteries with limited capacity, requiring precise power management to ensure reliable functionality over their intended lifespan. Disposable medical sensors designed for short-term vital monitoring must similarly adhere to minimal power budgets. Supply chain IoT smart labels, tasked with tracking perishable goods' conditions like temperature over extended transit times, rely on small, non-rechargeable batteries. In outdoor applications such as GPS trackers powered by flexible solar panels, strict power limits must be maintained to guarantee performance in intermittent sunlight conditions.

In this work, we introduce an approach for designing pNCs under strict power constraints at the algorithmic level while ensuring high accuracy. Leveraging the augmented Lagrangian approach together with a surrogate power model of the circuit, we integrate strict power budgets directly into the optimization process, ensuring power consumption stays within pre-defined limits. In short, the contributions of this work are:

- We propose an algorithmic framework based on the augmented Lagrangian method that enables training under strict power constraints while improving classification accuracy in a single training run, in contrast to penalty-based methods, which require multiple runs to approximate Pareto-optimal results.
- We developed data-driven surrogate power models for four different AFs, demonstrating that the choice of activation function and its learnable parameters have a significant impact on power consumption in pNCs.

Experiments show high accuracy within strict power budgets: p-tanh reaches 81.82% accuracy at 80% power, p-Clipped_ReLU maintains 74.65% at 20%, and p-ReLU reduces device count by up to 37% at low power budgets, highlighting its efficiency in power-constrained environments. Unlike the penalty-based approach, which requires numerous training runs to approximate the Pareto front, the augmented Lagrangian method achieves competitive results within a single training run, significantly reducing computational time and effort. This efficiency makes our approach more practical for circuit design applications.

The rest of this paper is structured as follows: Sec. II introduces PE, printed neuromorphic circuits (pNCs), and related works. Sec. III describes the development of power models and their integration into the design objective of the pNCs. In Sec. IV, the proposed approach is evaluated and discussed. Finally, Sec. V concludes this work and discusses possible future works.

II. Preliminaries

A. Printed Electronics

Printed solution-processed electronics (PE) is an additive manufacturing approach that requires less printing process steps

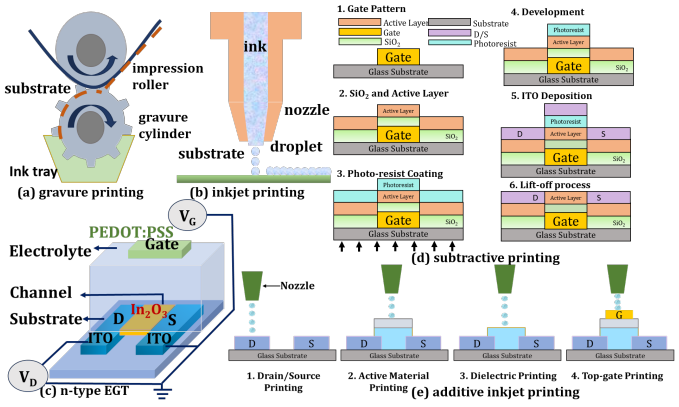


Fig. 2. Schematic of typical printing technologies: (a) gravure printing and (b) inkjet printing (c) a printed nEGT (d) Subtractive and (e) Additive printing process of PE.

than traditional silicon electronics, resulting in lightweight, ultra-low cost and flexible circuits. By selecting appropriate printing technologies, PE devices can be tailored to different production volumes and precision requirements. Furthermore, a wide range of printing materials enables important features for next-generation electronics, such as flexibility [10] and biocompatibility [12]. These makes PE a competitor in IoT applications like wearables [18], RFID [19], disposable electronics [20], energy harvesters, and implantable sensors [21].

Although vacuum-deposited methods achieve the best performance, simpler solution processes, such as spin-coating and inkjet printing are gaining popularity for their cost-effectiveness. PE technologies are categorized as either replication printing (e.g., gravure) for high-volume needs or jet printing (e.g., inkjet) for low-volume, bespoke fabricated circuits.

Most printed FETs are organic in nature and uses P-type materials that require high voltages (≥ 25V) [22]. This restricts their use in low-power applications. In contrast, inorganic N-type Electrolyte-Gated Transistors (nEGTs) offer higher electron mobility, enabling sub-1V operation [23], making nEGTs ideal for low-power devices reliant on small batteries or energy harvesters.

B. Printed Analog Neuromorphic Circuits

In the analog computing paradigm, neuromorphic computing has emerged as a preferred solution and is largely driven by advancements in artificial intelligence (AI), where artificial neural networks (ANNs) have outperformed in solving complex tasks [24].

1) Hardware Primitives: Fig. 3 (a) illustrates the circuit schematic of a pNC with 4-3-3 topology. Fig. 3(b) shows a 3-input, 3-output printed neuron, composed of a crossbar array and printed activation circuits (indicated in red). Additional circuits (highlighted in blue) are incorporated to handle negative weights, $\text{neg}(\cdot)$, by connecting resistors to either V_{in}^i or $\text{neg}(V_{\text{in}}^i)$, based on the sign of the corresponding weights. Figures 3(c)-(f) illustrate the schematics of various printed activation circuits. In the following sections, we provide a comprehensive overview of these circuit primitives.

a) Crossbar array: The array in Fig. 3 (b) illustrates a basic resistor crossbar architecture in pNCs, emulating weighted-sum operations of ANNs [14]. Using Kirchhoff's law [25] and

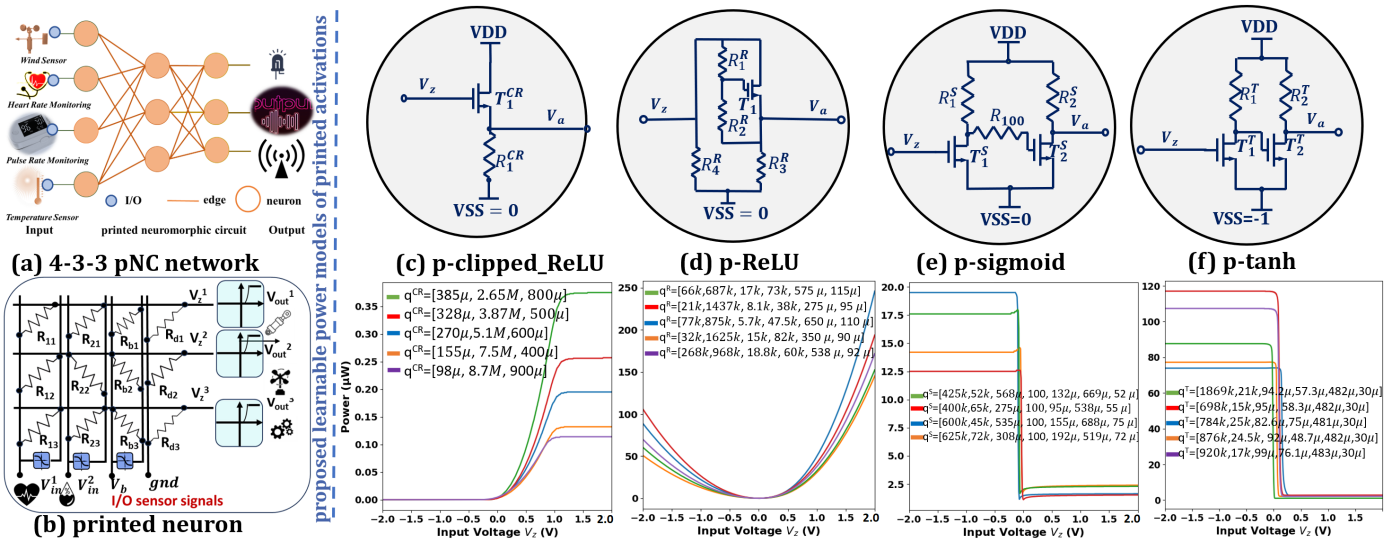


Fig. 3. Schematic of printed neuromorphic circuits. (a) Example of a 4-3-3 pNC network, (b) 3-input-3-output printed neuron based on crossbar array; surrogate power models of: (c) p-clipped_ReLU, (d) p-ReLU, (e) p-sigmoid and (f) p-tanh activation circuit.

expressing resistance as conductance ($g = 1/R$), the output voltage V_z^1 is calculated as

$$V_z^1 = \sum_j \frac{g_{j1}^C}{G_1} V_{in}^j + \frac{g_{b1}^C}{G_1},$$

where G_1 is the summed conductance of the first row in the crossbar, i.e., $\sum_i g_{i1}^C + g_{b1} + g_{d1}$. Here, V_z^1 represents a weighted sum of input voltages V_{in}^j, with conductance values defining weights and bias, enabling control over weights and biases by adjusting these conductance values.

The resistive nature of the crossbar array enables a direct analytical power model using the electronic power formula. For each resistor, the power is calculated as

$$P = \frac{\Delta V^2}{R} = \Delta V^2 \cdot g,$$

where ΔV is the potential difference across the resistor. Therefore, the power consumption of the crossbar, excluding negative weight circuits, is expressed as

$$\boldsymbol{P}^C = ((\tilde{\boldsymbol{V}}_{in} \odot \mathbb{1}_{\{\boldsymbol{\Theta} \geq 0\}} + \text{neg}(\tilde{\boldsymbol{V}}_{in}) \odot \mathbb{1}_{\{\boldsymbol{\Theta} < 0\}}) - \tilde{\boldsymbol{V}}_z)^2 \odot |\boldsymbol{\Theta}|,$$

Here, $\boldsymbol{\Theta} \in \mathbb{R}^{(M+2)\times N}$ is a matrix of learnable parameters for the crossbar array, where M and N denote the number of inputs and outputs, respectively. Each element in $\boldsymbol{\Theta}$ represents a surrogate conductance θ, where the absolute value $|\theta|$ encodes the physical conductance g of the corresponding resistor. Meanwhile, the sign of θ specifies whether a negative weight circuit must be pre-connected to emulate a negative weight. The operation $(\cdot)^2$ denotes an element-wise square, and $\mathbb{1}_{\{\cdot\}}$ is the element-wise indicator function that outputs 1 for true conditions and 0 otherwise. Each element in the matrix $\boldsymbol{P}^C$ represents the power of the corresponding resistor. Thus, the total power of the crossbar is given by

$$\mathcal{P}^C = \mathbf{1}_{M+2}^\top \cdot \boldsymbol{P}^C \cdot \mathbf{1}_N,$$

where $\mathbf{1}_{M+2}$ and $\mathbf{1}_N$ are vectors of ones.

b) Printed non-linear transformation circuits: In crossbar resistor arrays, conductances represent only positive weights. To incorporate negative weights, some resistors are paired with inverter-based circuits [13], as shown in Fig. 3(b) (indicated by blue boxes), which emulate negative weights by inverting the input voltages V_i. After passing through the crossbar, signals are processed by printed AFs, shown in Fig. 3 (c-f) (top). These printed AFs simulate the standard AFs in ANNs, introducing the non-linearity necessary for modeling complex functions. It should also be noted that a significant portion of the overall power is often consumed by AFs, whose non-linear behaviors make the modeling and optimization more complex. The power consumption of these circuits depends on physical parameters like transistor size, resistor values, and input voltage, requiring surrogate models ($\mathcal{P}^N$ and $\mathcal{P}^T$) trained on the SPICE simulation data [13]. This is further complicated as AFs are learnable parameters, meaning that their shapes are adapted during optimization, influencing both accuracy and power.

C. Power-Constrained Electronic Design

While traditional neuromorphic computing has made significant strides in low-power design [26], most existing methods rely on best-effort power optimization with soft constraints, where minimizing power is desirable but not mandatory [27, 28]. However, many real-world applications require strict (hard) power constraints to ensure reliable operation within predefined energy budgets. For example, wearable health devices, disposable medical sensors, and IoT smart labels depend on limited power sources like small batteries or energy harvesters, making it critical to enforce precise power budgets.

Our work directly addresses this gap by modeling and optimizing strict power constraints during the training of pNCs. Unlike soft constraints, where achieving optimal power-performance trade-offs is the goal, we enforce hard constraints to achieve the best accuracy and efficiency without exceeding the predefined power budget. This approach integrates AFs as learnable parameters and uses SPICE simulation for accurate

power estimation, ensuring a robust and practical methodology for power-constrained electronic design.

III. Methodology

To enable training under strict power constraints, we propose a training framework based on the augmented Lagrangian method. The primary objective is to maximize classification accuracy given the circuit parameters $\boldsymbol{\theta}$ and $\boldsymbol{q}$, while ensuring that power consumption below a predefined limit. Formally, this can be described by

$$\underset{\boldsymbol{\theta},\boldsymbol{q}}{\text{minimize}}\ \mathcal{L}(\mathcal{D},\boldsymbol{\theta},\boldsymbol{q})\ \text{s.t.}\ c(\boldsymbol{\theta},\boldsymbol{q}) = P(\boldsymbol{\theta},\boldsymbol{q}) - \overline{P} \leq 0, \quad (1)$$

where $\mathcal{L}(\mathcal{D},\boldsymbol{\theta},\boldsymbol{q})$ denotes the cross-entropy loss on the training data $\mathcal{D}$, which ensures high classification accuracy, and $c(\boldsymbol{\theta},\boldsymbol{q}) = P(\boldsymbol{\theta},\boldsymbol{q}) - \overline{P}$ represents the constraint function, ensuring that the estimated power consumption $P(\boldsymbol{\theta},\boldsymbol{q})$ remains within the strict task-specific upper limit $\overline{P}$. Evaluating the power contraint $c(\boldsymbol{\theta},\boldsymbol{q})$ requires accurate power estimation. Furthermore, since we will require derivatives through the contraint in the training procedure described later, we require a differentiable power model $P(\boldsymbol{\theta},\boldsymbol{q})$.

A. Surrogate power Modeling for Power Estimation

To simplify power analysis for resistor crossbar arrays, we adopt analytical solutions derived in [13]. However, modeling power consumption for nonlinear circuits, such as printed activation circuits (AFs), is more complex. We address this by approximating power consumption through SPICE simulations using the pPDK [29]. For each AF (p-ReLU, p-clipped ReLU, and p-sigmoid), we run 10,000 SPICE simulations, revealing distinct power behaviors as illustrated in Fig. 3 (c)-(f) (bottom). For instance, in p-clipped_ReLU, power spikes near a threshold as transistors conduct more current, then stabilizes due to the clipping effect; p-ReLU exhibits a smooth increase in power with input voltage, reflecting its unbounded nature; and p-sigmoid shows asymmetric power patterns due to higher current demands at negative voltages. Thus, the choice of AFs and their learnable parameters $q^{\mathrm{AF}} = [\mathbf{R}, \mathbf{W}, \mathbf{L}]$ (representing vectors of resistance, transistor widths, and transistor lengths) significantly impacts power consumption in pNCs.

To model learnable AFs, we define a feasible design space, $\mathbb{Q}^{\mathrm{AF}}$, with specified bounds for each parameter $\boldsymbol{q}^{\mathrm{AF}}$. We sample 10,000 circuit configurations using a Sobol sequence and simulate their power consumption using SPICE.

Based on this data, we train a 15-layer ANN for each AF as a surrogate model to map the corresponding physical variables $\boldsymbol{q}^{\mathrm{AF}}$ to power consumption, $\mathcal{P}^{\mathrm{AF}}(\boldsymbol{q}^{\mathrm{AF}})$. To enhance accuracy, data normalization and hyperparameter tuning are applied. This surrogate model provides efficient power estimation, enabling the total power calculation and analysis of the impact of AF configurations on power.

B. Power Estimation for a Printed Neuron

We estimate the power consumption of each printed neuron using the power of the crossbar ($\mathcal{P}^{C}$) and surrogate power models for its components: $\mathcal{P}^{\mathrm{AF}}$ and $\mathcal{P}^{\mathrm{N}}$, corresponding to the AF and negation circuits, respectively. To calculate the total power consumption, we determine N^{AF} and N^{N} [13], which represent the counts of AFs and negation circuits, respectively.

a) Counting Activation Circuits.: Initially, N^{AF} is calculated using an indicator function:

$$N^{\mathrm{AF}} = \mathbf{1}_N^{\top} \cdot \text{row max}\left\{\mathbb{1}_{\{|\boldsymbol{\theta}|>0\}}\right\}, \quad (2)$$

where $\mathbf{1}_N \in \mathbb{R}^N$ is a vector of all ones, $\mathbb{1}_{\{|\boldsymbol{\theta}|>0\}}$ assigns 1 to active circuits and 0 otherwise, and row max$(\cdot)$ returns the row-wise maximum values, as each row in $\boldsymbol{\theta}$ corresponds to a single activation circuit. If all elements in a row of surrogate conductances are zero, the corresponding activation circuit does not need to be printed or activated. This calculation ensures that any row with at least one non-zero conductance is counted only once. However, this piecewise constant function is non-differentiable, limiting its utility for gradient-based optimization.

b) Soft Count for Differentiability: For optimization, we replace $\mathbb{1}_{\{|\boldsymbol{\theta}|>0\}}$ in Eq. (2) with a sigmoid function $\sigma(|\boldsymbol{\theta}|)$ for gradient computation, defining $N_{\text{soft}}^{\mathrm{AF}}$ as

$$N_{\text{soft}}^{\mathrm{AF}} = \mathbf{1}_N^{\top} \cdot \text{row max}\left\{\sigma(|\boldsymbol{\theta}|)\right\}.$$

This relaxation allows gradient-based updates for N^{AF} by providing smooth gradients with respect to $\boldsymbol{\theta}$, as was previously done for N_{soft}^{N} in [13].

c) Final Power Estimation.: During the backward pass, the soft count facilitates gradient-based optimization. However, the final power estimation for a neuron is determined using the indicator function, as shown below:

$$\mathcal{P} = \mathcal{P}^{C} + N^{N} \cdot \mathcal{P}^{N} + N^{\mathrm{AF}} \cdot \mathcal{P}^{\mathrm{AF}}.$$

Power estimation in pNCs is influenced directly by the learnable parameters $\boldsymbol{q}^{\mathrm{N}}$, $\boldsymbol{q}^{\mathrm{AF}}$, and $\boldsymbol{\theta}$. Additionally, $\boldsymbol{\theta}$ indirectly affects the number of active elements in the circuit, which further impacts the total power consumption $P(\boldsymbol{\theta},\boldsymbol{q})$. In multi-neuron configurations, the output voltage of each neuron serves as the input for the subsequent neuron, with the final neuron's output representing the overall pNC output. The total power consumption is determined by summing the contributions from all neurons.

C. Power-constrained Training

To respect the constraints in training with Eq. (1), we require an adapation of the classic training procedure towards methods that can handle constraints. One such method that integrates particularly well backpropagation-based training schemes is the augmented Lagrangian method, see e.g. [30, 31]. It can be motivated by solving a sequence of unconstrained problems that converge to the solution of the constrained problem. Specifically, the method alternates between solving

$$\underset{\boldsymbol{\theta},\boldsymbol{q}}{\text{minimize}} \quad \max_{\lambda \geq 0}\ \mathcal{L}(\mathcal{D},\boldsymbol{\theta},\boldsymbol{q}) + \lambda \cdot c(\boldsymbol{\theta},\boldsymbol{q}) - \frac{1}{2\mu}(\lambda - \lambda')^2, \quad (3)$$

and updates to the Lagrange multiplier estimate λ' by

$$\lambda' \leftarrow \max\{0, \lambda' + \mu \cdot c(\boldsymbol{\theta},\boldsymbol{q})\}. \quad (4)$$

The parameter $\mu \in \mathbb{R}^{+}$ is a hyperparameter that controls the speed of convergence and influences the stability of the method.

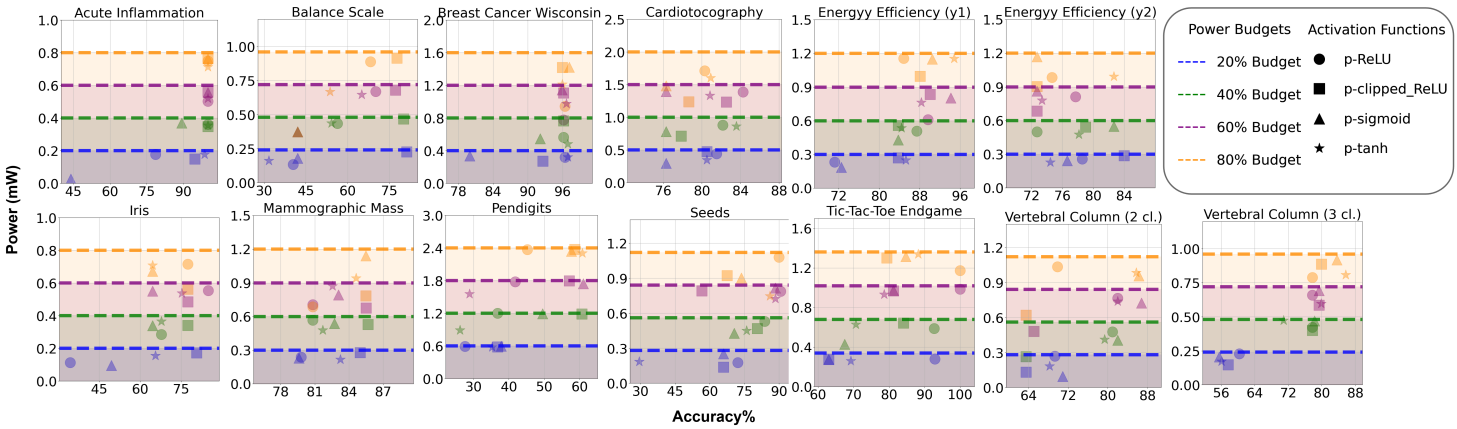


Fig. 4. Scatter plot showing the accuracy and power consumption for 13 datasets using four AFs (p-ReLU, p-clipped ReLU, p-sigmoid, p-tanh) under defined power budgets (20%, 40%, 60%, 80%). Each marker represents an AF, and dashed lines indicate the power budget thresholds. The goal is to achieve high accuracy while ensuring that power consumption stays below the specified budget constraints.

The minimization in Eq. (3) may be performed using classic backpropagation based training. The inner maximization over λ can be solved analytically for a given λ', details see [32]. After solving Eq. (3), the update of λ' in Eq. (4) adjusts the estimate for the Lagrange multiplier, encouraging the constraints to hold for the next minimization attempt. To save computation time, $\boldsymbol{\theta}$ and $\boldsymbol{q}$ should be warmstarted with the last solution obtained. The alternations of these steps can be stopped when a sufficiently good feasible solution is obtained.

IV. Evaluation

A. Experiment Setup

1) Dataset and Training Setup: To evaluate the proposed power-constrained training approach, we implemented the proposed method[2] using PyTorch [33] and tested it on 13 benchmark datasets used in prior studies [13, 34, 35]. The datasets were split into training (60%), validation (20%), and test (20%) sets to ensure consistent evaluation. Training used full-batch gradient descent with the Adam optimizer [36], starting with an initial learning rate of 0.1. Early stopping was applied to prevent overfitting by halving the learning rate after 100 epochs without improvement on the validation set and stopping training if the power constraint was violated. For the selection of μ, RayTune [37] was used. All pNCs were trained with a consistent topology (#$inputs$-3-#$outputs$) and randomly initialized parameters for each AF.

Following the primary training phase, a fine-tuning step was conducted to enhance accuracy while strictly adhering to power constraints. During this process, masks m^C were generated to deactivate inactive components, such as resistors with zero conductance and unused AFs. Similarly, masks m^N were introduced for negation circuits to enforce positive weights where necessary. These masks effectively pruned redundant hardware components, reducing power consumption. The model was then retrained using cross-entropy loss, optimizing accuracy without violating the power constraints. This mask-based adjustment ensured a trade-off between accuracy and power, even under stringent budget limitations.

[2]Code is available at https://github.com/KIT-Neuromorphic-Computing/Power-Constrained-PNC.

2) Benchmark Circuit Design Setup: All experiments at the simulation level were conducted using pPDK [29], while the functionality of printed neuromorphic hardware has been validated in previous work [14, 38].

3) Algorithm Benchmarking: To benchmark the proposed method, we compared its results to a Pareto front generated by traditional penalty-based optimization with p-tanh AFs [13]. For the baseline, we uniformly sampled 50 μ values in the range $[0, 1]$ and trained each model 10 times (using 10 different seeds) to generate the Pareto front. For the augmented Lagrangian approach, power constraints were set at 20%, 40%, 60%, and 80% of the maximum power observed during unconstrained training. This setup allowed for a comprehensive evaluation of the trade-offs between power and classification accuracy.

B. Result

To evaluate the effectiveness of our power-constrained training approach, we compared the results with a penalty-based method [13]. Four neural networks, each using a different AF (p-ReLU, p-Clipped_ReLU, p-sigmoid, and p-anh), were trained under four power budgets: 20%, 40%, 60%, and 80% of the maximum power consumption derived from unconstrained power training. The top three models per dataset were selected based on test accuracy. As shown in Fig. 4, each point represents classification accuracy and power consumption for a dataset. Marker shapes indicate AFs (circles for p-ReLU, squares for p-Clipped_ReLU, triangles for p-sigmoid, and stars for p-tanh), and marker colors represent power budgets, aligned with horizontal dashed lines for power thresholds. The x-axis shows accuracy (percentage), and the y-axis shows power (milliwatts).

Tab. I summarizes the average performance metrics across 13 datasets, comparing power consumption, accuracy, and device count among the proposed AFs (p-ReLU, p-Clipped_ReLU, p-sigmoid, and p-tanh). On the right side of the table, the results for the penalty-based baseline method [13] are presented, showing accuracy and power under different scaling factors α (0.25, 0.5, 0.75, and 1), which indicate the trade-off between power and accuracy. Additionally, for the baseline method, we generated a Pareto front (pink curve) using 500 penalty-based runs, varying the scaling factor α over the range $[0, 1]$. Fig. 5

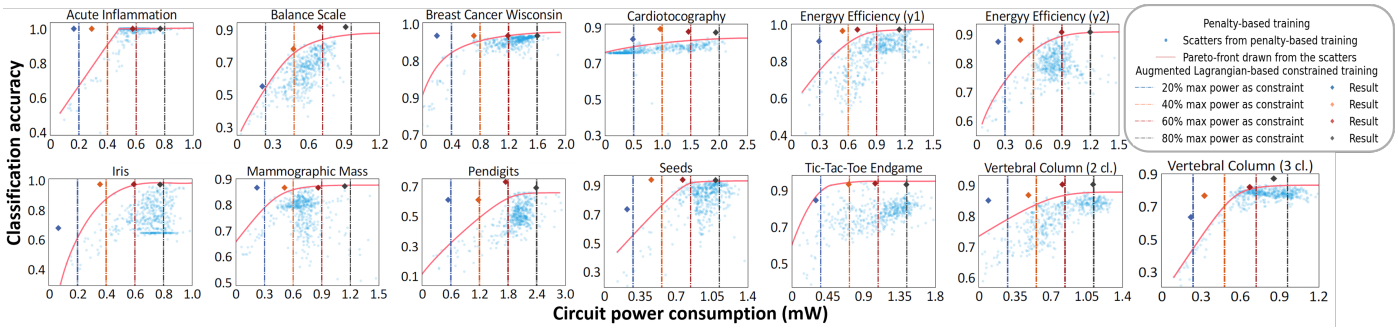


Fig. 5. Classification accuracy and power consumption of pNCs using exemplary p-tanh AF from penalty-based and Augmented Lagrangian-based training approaches. The blue scatters are the results of penalty-based training, while the pink curves are Pareto fronts drawn from the scatters. The vertical lines indicate the power constraints in the augmented Lagrangian approach, whereas the rhombus with the same color refers to the results from the augmented Lagrangian

compares the optimal solutions found by our method in a single training run against this Pareto front. Vertical dashed lines highlight power constraints, demonstrating that our method efficiently achieves power-constrained optima, often matching or surpassing the Pareto front with significantly fewer training runs.

TABLE I
AVERAGED PERFORMANCE METRICS ACROSS 13 DATASETS: COMPARISON OF METRICS (POW: POWER (mW), ACC: ACCURACY, DEV: DEVICE COUNT) ACROSS AFS AT DIFFERENT POWER BUDGETS AND PENALTY-BASED BASELINE

	Metric \ ACT	p-ReLU	p-clipped_ ReLU	p-sigmoid	p-tanh	Baseline	
20%	Pow	0.27	0.26	0.23	0.25	10.8	$\alpha = 1$
	Acc	67.94	**74.56**	62.60	67.29	54.9	
	#Dev	**17**	27	27	19	-	
40%	Pow	0.56	0.56	0.53	0.53	44.1	$\alpha = 0.75$
	Acc	78.11	**80.51**	73.96	74.60	83.6	
	#Dev	**28**	36	41	30	-	
60%	Pow	0.85	0.84	0.88	0.84	55.6	$\alpha = 0.5$
	Acc	**82.76**	78.54	78.89	78.45	86.3	
	#Dev	**34**	47	54	40	-	
80%	Pow	1.10	1.05	1.12	1.09	68.8	$\alpha = 0.25$
	Acc	80.42	78.84	78.05	**81.82**	87.5	
	#Dev	**37**	45	57	58	-	

C. Discussion

Our experimental results demonstrate the effectiveness of the augmented Lagrangian approach in achieving high accuracy while adhering to defined power constraints across various datasets. As shown in Fig. 4, all results lie below the defined power levels, confirming the method's capability to enforce power limits effectively. Notably, at 20% of the power budget, accuracy decreases compared to higher power budgets, highlighting the trade-off between power and performance. As shown in Tab. I, the baseline demonstrates high accuracy at $\alpha = 0.25$ but with higher power consumption, while $\alpha = 1$ reduces power at the cost of accuracy. In contrast, the proposed method achieves a balanced trade-off between power and accuracy across power budgets in a single training run. Furthermore, it delivers results comparable to or better than the Pareto front obtained from penalty-based objectives (Figure 5) without the need for extensive hyperparameter tuning.

The success of this approach is due to two main factors. First, by directly incorporating power constraints into the training objective, the augmented Lagrangian method avoids the instability often caused by high power penalties in penalty-based methods. Second, this explicit formulation focuses on achieving the best feasible accuracy for each specified power budget without unnecessarily minimizing power further.

Compared to penalty-based methods, which often create ill-conditioned optimization problems and fail to align with optimal power-accuracy trade-offs, the augmented Lagrangian method dynamically enforces constraints while maintaining accuracy. This ensures a balanced solution on the Pareto front with fewer computational resources.

Analyzing individual AFs reveals trade-offs between accuracy and device count. For example, p-tanh achieves up to 81.82% accuracy at an 80% power budget, but requires an average of 58 devices, whereas p-ReLU achieves 80.42% accuracy with only 37 devices—a 36% reduction. This makes p-ReLU particularly suitable for hardware-limited applications. These findings demonstrate that such trade-offs persist even under strict power constraints, with reduced device counts contributing to lower power consumption.

V. CONCLUSION

In this work, our objective is to introduce a methodology for power-constraint training of analog printed neuromorphic circuits. This is implemented by an augmented Lagrangian approach which leveraged the surrogate power models for activation function circuit primitives. The approach achieved en par to superior performance to penalty based methods across 13 benchmark datasets, while requiring fewer training time to find suitable pNCs. We additionally developed power models for four printed AFs, enabling analysis of power and accuracy trade-offs. While p-tanh achieves higher accuracy, p-ReLU minimizes device count, proving beneficial for resource-limited applications. Due to the flexibility of the proposed approach, future works may explore its applicability to additional circuit components and constraints.

ACKNOWLEDGMENT

This work has been partially supported by the European Research Council (ERC) and the Carl-Zeiss-Foundation as part of "stay young with robots" (JuBot) project.

REFERENCES

[1] A. U. Alam *et al.*, "Fruit Quality Monitoring with Smart Packaging," *Sensors*, vol. 21, no. 4, p. 1509, 2021.

[2] Q. Sun *et al.*, "Smart Band-Aid: Multifunctional and Wearable Electronic Device for Self-Powered Motion Monitoring and Human-Machine Interaction," *Nano Energy*, vol. 92, p. 106840, 2022.

[3] H. Zhao *et al.*, "Towards Temporal Information Processing–Printed Neuromorphic Circuits with Learnable Filters," in *Proceedings of the 18th ACM International Symposium on Nanoscale Architectures*, 2023, pp. 1–6.

[4] E. Shirzaei Sani *et al.*, "A Stretchable Wireless Wearable Bioelectronic System for Multiplexed Monitoring and Combination Treatment of Infected Chronic Wounds," *Science Advances*, vol. 9, no. 12, p. 7388, 2023.

[5] W. S. Wong and A. Salleo, *Flexible electronics: materials and applications*. Springer Science & Business Media, 2009, vol. 11.

[6] W. Gao, H. Ota, D. Kiriya, K. Takei, and A. Javey, "Flexible electronics toward wearable sensing," *Accounts of chemical research*, vol. 52, no. 3, pp. 523–533, 2019.

[7] H. Zhao, T. Röddiger, and M. Beigl, "Aircase: Earable Charging Case with Air Quality Monitoring and Soundscape Sonification," in *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*, 2021, pp. 180–184.

[8] Sigenics, "Sigenics - custom integrated circuits," https://sigenics.com/, accessed: Nov. 19, 2024.

[9] I. Ayyub, "Top 10 cheapest programmable microcontrollers," 2018, accessed: Nov. 19, 2024. [Online]. Available: https://pic-microcontroller.com/world-top-10-cheapest-microcontrollers-mcus/

[10] I. I. Labiano *et al.*, "Flexible Inkjet-printed Graphene Antenna on Kapton," *Flexible and Printed Electronics*, vol. 6, no. 2, p. 025010, 2021.

[11] J. Chang *et al.*, "Challenges of Printed Electronics on Flexible Substrates," in *2012 IEEE 55th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2012, pp. 582–585.

[12] A. Kaidarova *et al.*, "Wearable Multifunctional Printed Graphene Sensors," *NPJ Flexible Electronics*, vol. 3, no. 1, pp. 1–10, 2019.

[13] H. Zhao *et al.*, "Power-Aware Training for Energy-Efficient Printed Neuromorphic Circuits," in *42nd IEEE/ACM International Conference on Computer-Aided Design*, 2023.

[14] D. D. Weller *et al.*, "Realization and Training of an Inverter-based Printed Neuromorphic Computing System," *Scientific reports*, vol. 11, no. 1, pp. 1–13, 2021.

[15] Z. Yu *et al.*, "An Overview of Neuromorphic Computing for Artificial Intelligence Enabled Hardware-based Hopfield Neural Network," *IEEE Access*, vol. 8, pp. 67 085–67 099, 2020.

[16] M. Alioto, "Ultra-low Power VLSI Circuit Design Demystified and Explained: A Tutorial," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 1, pp. 3–29, 2012.

[17] S. Yan and E. Sanchez-Sinencio, "Low Voltage Analog Circuit Design Techniques: A Tutorial," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 83, no. 2, pp. 179–196, 2000.

[18] J. C. Costa *et al.*, "Flexible Sensors — From Materials to Applications," *Technologies*, vol. 7, no. 2, 2019.

[19] S. Kim, "Inkjet-Printed Electronics on Paper for RF Identification (RFID) and Sensing," *Electronics*, vol. 9, no. 10, p. 1636, 2020.

[20] R. Martins, I. Ferreira, and E. Fortunato, "Electronics with and on Paper," *physica status solidi (RRL)–Rapid Research Letters*, vol. 5, no. 9, pp. 332–335, 2011.

[21] R. Sobot, "Implantable Technology : History, Controversies, and Social Implications," *IEEE Technology and Society Magazine*, vol. 37, no. 4, pp. 35–45, 2018.

[22] S. K. Garlapati *et al.*, "Ink-Jet Printed CMOS Electronics from Oxide Semiconductors," *Small*, vol. 11, no. 29, pp. 3591–3596, 2015.

[23] F. Rasheed and M. Tahoori, *Compact Modeling and Physical Design Automation of Inkjet-Printed Electronics Technology*. Karlsruhe Institute of Technology (KIT), 2020.

[24] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in Artificial Neural Network Applications: A Survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[25] G. Kirchhoff, "LXIV. On a Deduction of Ohm's Laws, in connexion with the Theory of Electro-statics," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 37, no. 252, pp. 463–468, 1850.

[26] Y.-H. Chen *et al.*, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[27] B. Li *et al.*, "Build Reliable and Efficient Neuromorphic Design with Memristor Technology," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 224–229.

[28] A. Basu *et al.*, "Low-Power, Adaptive Neuromorphic Systems: Recent Progress and Future Directions," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 6–27, 2018.

[29] F. Rasheed *et al.*, "Variability Modeling for Printed Inorganic Electrolyte-gated Transistors and Circuits," *IEEE transactions on electron devices*, vol. 66, no. 1, pp. 146–152, 2018.

[30] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

[31] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[32] J. K. H. Franke, M. Hefenbrock, G. Koehler, and F. Hutter, "Improving deep learning optimization through constrained parameter regularization," 2024. [Online]. Available: https://arxiv.org/abs/2311.09058

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An Imperative Style, High-performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[34] H. Zhao *et al.*, "Aging-Aware Training for Printed Neuromorphic Circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22)*, 2022.

[35] ——, "Highly-Bespoke Robust Printed Neuromorphic Circuits," in *Design, Automation and Test in Europe (DATE)*. IEEE, 2023.

[36] D. P. Kingma *et al.*, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," 2018, accessed: 2024-07-11. [Online]. Available: https://docs.ray.io/en/latest/tune.html

[38] S. A. Singaraju *et al.*, "Artificial Neurons on Flexible Substrates: A Fully Printed Approach for Neuromorphic Sensing," *Sensors*, vol. 22, no. 11, p. 4000, 2022.

Graph in the Vault: Protecting Edge GNN Inference with Trusted Execution Environment

Ruyi Ding*, Tianhong Xu*, Aidong Adam Ding, Yunsi Fei
Northeastern University
{ding.ruy, xu.tianh, a.ding, y.fei}@northeastern.edu

***Abstract*—Wide deployment of machine learning models on edge devices has rendered the model intellectual property (IP) and data privacy vulnerable. We propose GNNVault, the first secure Graph Neural Network (GNN) deployment strategy based on Trusted Execution Environment (TEE). GNNVault follows the design of "partition-before-training" and includes a private GNN rectifier to complement with a public backbone model. This way, both critical GNN model parameters and the private graph used during inference are protected within secure TEE compartments. Real-world implementations with Intel SGX demonstrate that GNNVault safeguards GNN inference against state-of-the-art link stealing attacks with a negligible accuracy degradation ($< 2\%$).**

I. Introduction

On-device machine learning has emerged as an important paradigm for tasks requiring low latency and high privacy [1]. Applications like Apple's CoreML [2] and Huawei's MindSpore [3] enable user personalization on local devices. This trend has also extended to Graph Neural Networks (GNNs) [4], [5], ensuring the privacy of user data during inference for tasks such as community detection [6], e-commerce personalization [7], and recommender systems [8]. However, local GNN inference grants users significant privileges to local models and data, introducing additional security vulnerabilities [9].

There are two primary security threats to inference on local devices: model intellectual property (IP) theft and data privacy breach. Valuable model IP includes the hyper-parameters, trained parameters, and model's functionality [10]. Data privacy issues involve attacks such as membership inference [11] and link stealing attack [12], which can exploit the white-box accessibility of local models. Existing defenses, including watermarking [13], non-transferable learning [14], and homomorphic encryption [15], are often passive, inaccurate, or computation-expensive. Recently, studies have begun leveraging Trusted Execution Environments (TEEs) to protect local models and data [16], [17], by partitioning them into slices and deploying critical components within secure compartments.

However, current TEE-based secure deployment mainly targets datasets for tasks such as computer vision and large language models, which is not applicable to GNN inference. Specifically, GNN operations require updating the graph node representations (i.e., embeddings) based on information from their neighboring data points. Thus, *the entire graph data, including node features and their connectivity (adjacency matrix), has to be stored locally during inference*, posing additional challenges for data privacy. Moreover, TEEs have limited memory capacities, while large graphs require significant storage for node features. To bridge this gap, we propose a new method for local GNN deployment, GNNVault, which effectively preserves both critical model parameters and the most important structural information (i.e., edges) using TEE.

GNNVault employs the "partition-before-training" strategy [16], where the GNN model is designed with a public backbone and a private GNN rectifier. Specifically, the public backbone is computation-intensive but inaccurate, which is trained with a substitute graph and deployed in the untrusted environment. The GNN rectifier leverages the real private graph structure to rectify the backbone's node embeddings. It is designed to be much smaller and resides within the TEE. Our experiments show that GNNVault achieves performance comparable to the original unprotected GNN while preventing information leakage of private parameters and edges. We also evaluate our design on a real system using Intel SGX, demonstrating the low overhead of GNNVault.

Our Contributions: We propose GNNVault, the first secure deployment framework designed specifically for graph neural networks. Our work makes the following contributions:

1) We propose various communication schemes between public backbones and private rectifiers in GNNVault, and evaluate the performance and implementation cost of rectifiers.
2) Software evaluation demonstrates that GNNVault achieves a high inference accuracy with a small rectifier inside TEE while protecting critical model parameters and edge data.
3) We implement a real-world deployment using Intel SGX to run GNNs locally. Experimental results show that the introduction of the enclave increases a small inference overhead and negligible accuracy degradation (less than 2%), demonstrating the effectiveness of GNNVault.
4) We conduct a security analysis with link stealing attacks on the intermediate data communicated from the public backbone to the rectifier, showing that no private edge information is leaked in the untrusted environment.

II. Background

A. Graph Neural Network

GNNs process graph-structured data, which contains relational information between nodes (or edges). Without loss of generality, we follow the common GNN definition [18]: a graph $G = (V, E)$ consists of the node set V with n nodes, each with a feature represented by a d-dimensional vector,

denoted as $X \in \mathbb{R}^{n \times d}$. The set of edges E is depicted by an adjacency matrix $A \in \{0,1\}^{n \times n}$, indicating the existence of connections between nodes in G. A GNN model is a function $f(X, A)$ to summarize global information in the graph for downstream tasks including node classification [19], community detection [6], and recommender systems [20]. The forward propagation of GNN layer k can be written as

$$H^{(k)} = \sigma(\hat{A}H^{(k-1)}W^{(k)}) \tag{1}$$

where $H^{(k)} \in \mathbb{R}^{N \times d^{(k)}}$ is the matrix of node embedding at layer k, $\hat{A}$ is the adjacency matrix with self-loops normalized with degree matrix, $W^{(k)} \in \mathbb{R}^{d^{(k-1)} \times d^{(k)}}$ is the learnable weight matrix, and σ is an activation function. Note that GNN operations require $\hat{A}$ through message passing during the inference process, which causes edge data privacy concerns.

B. Model Deployment in Trusted Execution Environments

A TEE, such as Intel SGX or ARM TrustZone, protects code and data within it, ensuring their confidentiality and integrity with hardware support against strong adversaries including a malicious host operating system. The increasing demand for edge AI applications, e.g., those that require real-time responses or rely on local data or private data, has led to a trend of using TEEs for IP protection and privacy preservation. Previous works focus on securely deploying DNNs by isolating the non-linear layers [21] or by partitioning the model and placing the final dense layers inside the TEE [22]. Recently, Zhang et.al [16] also consider the privacy of training data and propose a "partition-before-training" strategy against membership inference attack [11]. Additionally, Liu et al. [17] design a deployment strategy that only allows single-directional data flow from the untrusted environment to the enclave. However, all previous works only focus on local DNN's IP and data privacy, which are not applicable to the secure deployment of a GNN on the edge. Our proposed method, GNNVault, bridges this gap by protecting the security of trained GNNs and private edge data using TEEs.

III. Problem Statement

A. Deploying GNN locally Causes Vulnerabilities

Deploying GNNs on local devices requires access to graph data in addition to the trained model, which introduces unique security and privacy challenges. Similar to DNN deployment, the IP of the well-performed local model, including its trained weights and biases, is valuable asset that must be protected against model extraction attacks. Beyond the model IP, local GNN inference raises additional privacy concerns due to the nature of GNN architecture. Specifically, during the message-passing phase of GNN inference, target nodes aggregate information from neighboring nodes to update their embeddings. This process involves accessing sensitive edge data, such as user-product interactions in recommender systems. In our work, we will address the GNN IP infringement and edge data breach vulnerability during GNN deployment.

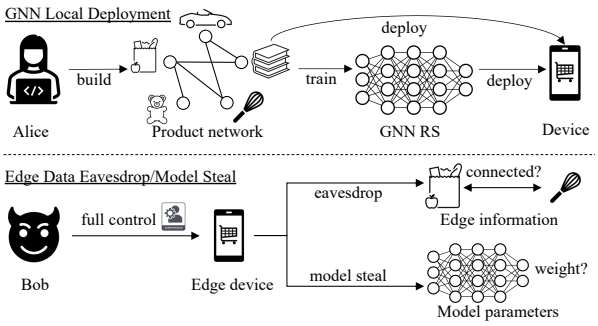


Fig. 1. **Motivation Example:** Alice (victim) builds a graph of products and trains a GNN RS. She deploys both edge data and RS on local devices. Bob (attacker) accesses this device and steals the edge data and model parameters.

B. Edge Privacy is Valuable

Membership inference attack is the most common data privacy threat to machine learning models [11], where the goal is to determine whether a given data point belongs to the training set. However, in the context of GNNs, edge data raises additional privacy concerns. Link stealing attacks [12], [23] aim to infer the connectivity between any pair of given nodes. In this work, we focus on the adjacency information (edges), while considering the node features as public. A real-world example is illustrated in Fig. 1, where Alice (victim) deploys a recommender system (RS) on local edge devices. In such a product graph, the node features are public attributes of the products—such as price, user reviews, or categories—that are available to any user. However, the internal relationships between products require intensive learning from user behavior data, which is valuable IP for the model vendor. Therefore, safeguarding the node connectivity information during GNN local inference is of great importance.

C. TEE Has Memory Restrictions

The introduction of TEE greatly enhances data security and privacy with secure compartments. However, TEE platforms face significant memory limitations, a critical constraint for secure computation. For instance, for Intel SGX trusted enclaves, the physical reserved memory (PRM) is limited to 128MB, with 96 MB of it allocated to the Enclave Page Cache (EPC) [24]. Excessive memory allocation will lead to frequent page swapping between the unprotected main memory and the protected enclave, which can cause high overhead and additional encryption/decryption to ensure data integrity [25]. This memory constraint poses a significant challenge for deploying GNN models and the entire graph (including node features and adjacency information) within the secure enclave, which often far exceed the PRM limitation of enclaves.

IV. Our Approach: GNNVault

A. Threat Model

Our work aims to protect both GNN model parameters and the edge data during inference, specifically using the TEE of Intel SGX. Unless specified, we consider node classification as the primary downstream task.

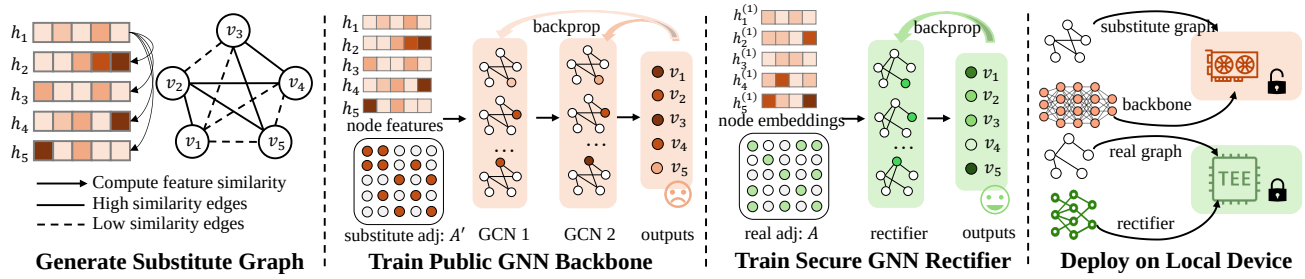


Fig. 2. **Overview of GNNVault.** GNNVault can be summarized into four steps: 1. generate a substitute graph based on the similarity between the node features; 2. train the GNN backbone with the substitute adjacency matrix; 3. freeze the backbone and train the secure recalibration rectifier using real adjacency information; 4. deploy the backbone and substitute graph in untrusted environments (in brown) and deploy the real graph and rectifiers on enclaves (in green).

Attacker's Goal. We assume there are "honest-but-curious" attackers such as users of local ML systems. Specifically, these passive attackers are curious about sensitive data but do not intend to corrupt the system's integrity and functionality. Their objective is to extract confidential graph structural information, represented by the adjacency matrix, and to steal the parameters of highly accurate GNNs present on the device.

Attacker's Capability. The attacker has full control over the victim system, enabling them to query the GNN model with any chosen node to obtain the model output results. Additionally, they can observe any computation and intermediate results in the untrusted environment, such as the public part of the GNN model or the intermediate node embeddings. However, any data and operations within the enclaves remain sealed and inaccessible to the attacker. Other attacks against TEE, e.g., side-channel attacks, are out of the scope of this work.

Attacker's Knowledge. We assume the attacker lacks knowledge of the adjacency matrix, which represents confidential relationships between any two nodes in the graph (e.g., hidden correlations between customers and products in recommender systems or private connections between users in social networks). The node features, however, are considered public knowledge such as product attributes or user profiles.

Defender's Goal. The defender, a model provider such as Apple or Amazon, is responsible for the design and deployment of the GNN model on the local device, aiming to protect both the highly accurate GNN model and the private adjacency matrix information. This involves designing the GNN structure and establishing the secure deployment of the trained model and graph data on the inference platform.

B. Overview of GNNVault

GNNVault employs the design strategy known as *partition-before-training* [16], to completely avoid using the private edge data during training of the public backbone. To maintain the model performance, a GNN rectifier is designed to use the real adjacency matrix, which is lightweight and securely protected within the TEE. Furthermore, to prevent information leakage through intermediate data transfer, GNNVault allows only one-way communication from the untrusted environment to the enclave. As illustrated in Fig. 2, GNNVault can be summarized in four steps: (1) generate substitute graph(s); (2) train the public backbone; (3) train the private rectifier; (4) securely deploy the data and models.

C. Public GNN Backbone

The backbone is trained with public data, aiming to provide sufficient pre-computation during the inference process. Unlike the backbones for DNNs or LLMs in [16], which are open-source pre-trained models, GNNs are more dataset-specific and cannot directly leverage such models. Therefore, our GNNVault uses a pre-trained neural network backbone that utilizes public node features. We generate a substitute adjacency matrix based on node similarity and train this backbone for the target tasks (node classification).

Substitute Adjacency Matrix: To enable the backbone model to capture message passing between nodes, we design it as a GNN with a substitute adjacency matrix. This substitute adjacency is computed based on the similarity between nodes:

$$A'(i,j) = \begin{cases} 1 & \text{if } F(i,j) \geq \tau \text{ for each } j \neq i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Here, $F(i,j) = \text{sim}(x_i, x_j)$ represents the similarity measure computed between two node features, such as cosine similarity $\frac{x_i \cdot x_j}{|x_i||x_j|}$. Based on this similarity, we determine the connected edges using a threshold τ. Alternatively, connectivity can be computed using the k-nearest neighbors, where we connect the top-k similar nodes to construct the adjacency matrix.

Training the Public Backbone: The public backbone consists of GCN layers trained with the substitute adjacency matrix A' and the public node features X. Each GCN layer k projects the high-dimensional features (embeddings) from $\mathbb{R}^{N \times d^{(k)}}$ to a lower-dimensional space $\mathbb{R}^{N \times d^{(k+1)}}$ through the message passing procedure. However, due to the loss of spatial information from the real adjacency matrix, this public backbone often exhibits lower accuracy. Therefore, the other component, a private rectifier, aims to recalibrate the embeddings in $\mathbb{R}^{N \times d^{(k+1)}}$ with real adjacency to achieve higher accuracy.

D. Secure GNN Rectifier

With the public backbone trained using the substitute adjacency matrix, the node embeddings are passed from the backbone to the GNN rectifier to rectify the misinformation from A'. The rectifier is designed with GCN layers that take the outputs of the backbone as inputs, along with the true adjacency matrix for rectification. Notably, the embeddings are often in a lower-dimensional latent space compared to the original node features. Therefore, a rectifier layer will be

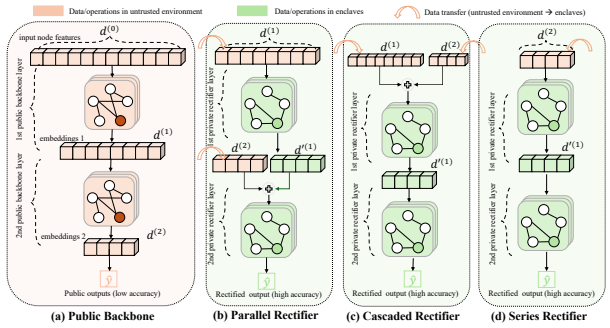


Fig. 3. **Backbone and GNN Rectifier Designs:** (a). Public GNN backbone; (b). Parallel rectifier; (c). Cascaded rectifier; (d) Series rectifier.

smaller than the corresponding backbone layer, so that it can fit into the enclave. During the training process of the rectifier, we freeze the pre-trained GNN backbone and adjust the rectifier parameters using cross-entropy loss for node classification.

How to process node embeddings from the backbone will result in various rectifier sizes and rectification performances. We consider three communication schemes (data flows) and the corresponding rectifiers, as shown in Fig. 3:

Parallel Rectifier: Layers of the public backbone and layers of the rectifier run layer-by-layer in parallel. We aim to rectify the node embeddings right after each message-passing with the substitute adjacency matrix. Our experiment shows that the parallel rectifiers have the best accuracy (Sec. V-B1).

Cascaded Rectifier: This design allows the backbone to run first, and then we concatenate all GCN layer outputs from the untrusted environment as input to the rectifier, which provides the rectifier a global view of the node embeddings but introduces additional computational requirements.

Series Rectifier: In this design, we utilize only the final layer from the backbone as input to the rectifier. This approach uses the smallest input space and results in lower memory consumption and faster response time.

E. Deployment of GNN on TEE

As shown in Fig. 2, we deploy the public backbone and substitute graph in the untrusted environment, where the model can benefit from acceleration from GPUs. We safeguard the real adjacency information and GNN rectifiers inside the secure enclaves to prevent edge privacy breaches and model stealing attacks. The true adjacency matrix will be saved in the Coordinate (COO) format, which maintains only the non-zero elements with their indices for memory efficiency, with the pre-computed degree matrix, to accelerate the normalization process. During the inference phase, each layer of the rectifier's output will be stored internally within the enclave, preventing intermediate data from leaking to the untrusted environment. Logits have been shown to contain additional privacy information such as data membership and link connectivity [12], [11]. Therefore, we also require the final output to be label-only, i.e., the logits are kept inside the enclave and only the output class is available to users.

V. Experimental Validation

A. Experiment Setup

Datasets: We focus on semi-supervised learning tasks for node classification with GNNs on multiple datasets at different scales, as shown in Table I. Specifically, we evaluate GNNVault on three standard datasets in [26], two larger graphs *Amazon Computer* and *Photo* in [27], and *CoraFull* [28] which has many classes (70). Table I shows the memory requirement for a dense adjacency matrix. To train the GNN, we follow the common practice: using 20 labeled node per class [27], and the unlabeled nodes are used as the testing set. If not specified, we use KNN with k=2 as substitute graphs in backbones, which are chosen based on the ablation study in V-B4.

TABLE I
Dataset Used in GNNVault Validation

Dataset	# Node	# Edge	# Feature	# Class	Dense A (MB)
Cora [26]	2,708	10,556	1,433	7	167.85
Citeseer [26]	3,327	9,104	3,703	6	253.35
Pubmed [26]	19,717	88,648	500	3	8898.01
Computer [27]	13,752	491,722	767	10	4328.56
Photo [27]	7,650	238,162	745	8	1339.47
CoraFull [28]	19,793	126,842	8,710	70	8966.74

Models: We evaluate GNNVault on three different GNN structures based on dataset features space size: $\mathcal{M}_1$ has as a 3-layer GCN backbone with output channel size as $(128, 32, \mathcal{C})$ and a rectifier $(128, 32, \mathcal{C})$, which is designed for a smaller dataset including Cora, Citeseer, Pubmed, where $\mathcal{C}$ is the label space size. $\mathcal{M}_2$ has wider output channels (256) for both the backbone and the rectifier, which is used for the case when the model has a high-class number, such as CoraFull. We also test a larger and deeper design $\mathcal{M}_3$ with a backbone $(256, 64, 32, 16, \mathcal{C})$ and rectifier $(64, 32, \mathcal{C})$. Different rectifiers may have various input channel sizes, as shown in Fig. 3.

Metrics: We first compute the accuracy of the original GNN model, which has the same architecture as the backbone but uses the real adjacency matrix and therefore has different parameters, denoted as p_{org}. The difference between the classification accuracy of the public backbone in the normal world, denoted as p_{bb}, and the accuracy of the GNN rectifier, denoted as p_{rec}, measures the *protection performance*. We also compare the size of the public backbone and private rectifier in terms of the number of model parameters, denoted as θ_{bb}, and θ_{rec}. To show accuracy degradation, we utilize the difference between p_{rec} and p_{org} and the silhouette score to show the node's clustering performance [29].

Platform: We run the training procedure and the software evaluation process on Ubuntu 18.04.6 with NVIDIA TITAN RTX. We also follow a similar setting to [16] as a real-world case study on an SGX-enabled Intel Core i7-7700, running Ubuntu 20.04.6 and utilizing Intel SGX SDK and PSW.

B. Software-based Validation

1) Evaluation of GNNVault Performance: The performance of GNN deployment using TEE on a local device can be evaluated using the following attributes: 1. Accuracy degradation (p_{org} - p_{rec}) (lower is better); 2. Protection performance $\Delta p = p_{\text{rec}} - p_{\text{bb}}$ (higher is better); 3. The model size inside

TABLE II
GNNVAULT PERFORMANCE WITH KNN GRAPH ($k = 2$), **BOLD INDICATES THE BEST RECTIFIER**; UNDERLINE IS THE SECOND BEST.

	Original / Backbone			Parallel GNNVault			Series GNNVault			Cascaded GNNVault		
Dataset	$p_{org}(\%)$	θ_{bb} (M)	p_{bb} (%)	p_{rec} (%)	Δp (%)	θ_{rec} (M)	p_{rec} (%)	Δp (%)	θ_{rec} (M)	p_{rec} (%)	Δp (%)	θ_{rec} (M)
Cora [26]	80.4	0.188	60.2	**78.8**	18.6	0.022	78.2	18.0	0.0088	77.6	17.4	0.027
Citeseer [26]	65.2	0.479	60.3	**70.1**	9.80	0.022	68.7	8.40	0.0087	69.0	8.70	0.026
Pubmed [26]	77.1	0.068	66.6	**75.2**	8.60	0.022	75.1	8.50	0.0085	73.6	7.00	0.025
Computer [27]	75.5	0.216	56.6	77.6	21.0	0.021	**78.2**	21.6	0.0039	77.4	20.8	0.027
Photo [27]	83.7	0.210	68.3	84.9	16.6	0.021	84.2	15.9	0.0037	**85.1**	16.8	0.026
CoraFull [28]	59.5	2.27	43.1	57.8	14.7	0.051	**58.0**	14.9	0.050	55.8	12.7	0.060

TABLE III
COMPARE VARIOUS BACKBONE DESIGNS

	DNN		random		cosine		KNN	
Dataset	p_{bb}	p_{rec}	p_{bb}	p_{rec}	p_{bb}	p_{rec}	p_{bb}	p_{rec}
Cora	54.4	76.8	17.2	51.5	55.3	**79.1**	60.2	78.8
Citeseer	53.9	64.6	18.9	38.3	46.2	64.3	66.6	**70.1**
Pubmed	71.9	73.9	34.5	52.1	72.1	**76.0**	66.6	75.2
Computer	52.6	73.6	7.16	28.9	44.6	76.7	56.6	**77.6**
Photo	64.3	83.4	30.4	52.8	69.1	**84.9**	68.3	84.9
CoraFull	43.9	57.7	2.69	27.3	40.1	55.6	43.1	**57.8**

the enclave θ_{rec} (smaller is better). In Table II, we present the performance of GNNVault on three rectifier designs, following the experimental setups described in Section V-A. Among the three rectifier designs, the parallel rectifier achieves the highest accuracy on three datasets and ranks second-best in others, and it also offers the highest protection. The cascaded rectifier has the largest model size but shows lower accuracy, as the input GCN layer may not handle all complex inputs together well. The series rectifier has the smallest size inside the TEE, as it only requires the final node embedding from the backbone as input, and yet it still achieves good performance overall.

2) Evaluation of Different Backbones: GNNVault's backbones show low accuracy when using the substitute graph, indicating that our design effectively prevents malicious attackers from accessing any high-precision models in untrusted environments. We then compare the performance of GNNVault using different types of backbones: the DNN backbone (an MLP using only node features), and the GNN backbones with three different ways to generate the substitute graph: random (the graph is randomly generated), cosine (the graph is generated based on cosine similarity), and KNN (the graph is generated using the KNN algorithm). For the GNN-based backbones, we sample the graph density to match that of the original graph. Among all four types of backbones, the cosine and KNN methods demonstrate the best performance, while the GNN with a random graph shows the lowest accuracy. Specifically, the random graph incorporates too much misinformation, degrading the embedding performance of the backbone model and making rectification difficult. Moreover, compared with the DNN model, the GNN backbones using cosine similarity and KNN substitute graphs enable the backbone not only to extract information from the node features but to gather additional information from similar nodes via message passing. This additional information enhances the rectification process when using the correct adjacency information.

3) Rectifier Interpretation: To illustrate the rectification process, we visualize the node latent space of a parallel GNNVault applied to the Cora dataset, which contains 7 classes. In Fig. 4, we illustrate how the latent space changes layer-by-layer between the backbone and the rectifier using t-SNE [30]. The rectifier layer significantly improves the clustering performance by gradually incorporating the real adjacency matrix, ultimately approaching the original model's performance. Additionally, we plot the silhouette score of each layer's embeddings to numerically assess their clustering performance. The results show that the rectifier's scores (green line) become close to those of the original GNN (gray line), while the backbone model's scores (orange line) remain low.

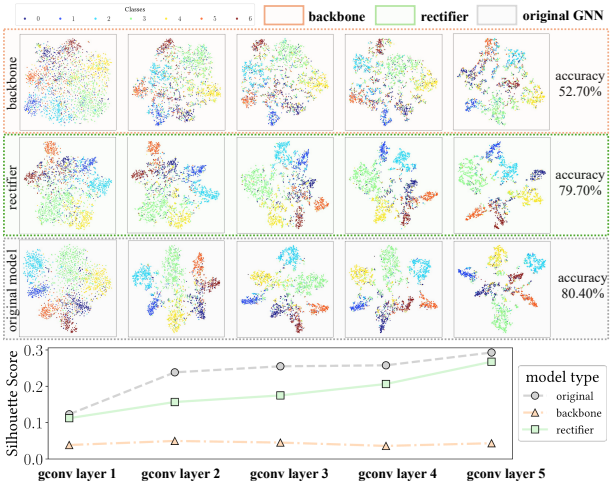


Fig. 4. **Visualization of the node embeddings.** We visualize the embedding latent space for parallel GNNVault on Cora dataset using the same structure for the original GNN, backbone and rectifier as $\mathcal{M}_2$. The line chart shows the Silhouette Score of the clustering performance**(the higher the better).**

4) Ablation Study: We evaluate the impact of different hyperparameters on GNNVault's performance, as shown in Fig. 5. Specifically, we vary the number of neighbors in the KNN-based substitute graph, adjust the cosine similarity threshold for the cosine similarity graph, and change the percentage of random edges (relative to the number of real edges) for the random substitute graph. For the KNN method, the performance remains stable because the algorithm consistently connects similar nodes based on their features. Increasing k primarily affects the density of the substitute graph without significantly impacting performance. We select k=2 as the number of edges for it is close to the real graph for most of the datasets. In contrast, selecting a low cosine similarity threshold for the cosine similarity graph can connect unrelated nodes, which adversely affects GNNVault's performance (when the cosine similarity threshold is $\leq$ 0.2). As we increase the

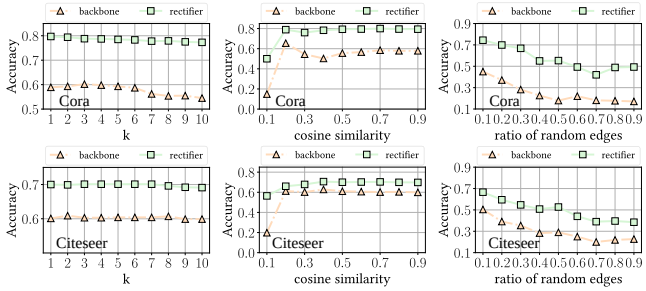


Fig. 5. Impact of hyperparameters of different substitute graphs

number of random edges in the substitute graph, the performance of both the backbone and the rectifier decreases as more structural noise is introduced. Notably, when we use an extremely small number of edges, random GNNVault's accuracy approaches that of the DNN backbone.

C. Real-world Implementation

1) Implementation Details: We implemented GNNVault on a desktop PC equipped with an Intel Core i7-7700 (3.60GHz) processor. The public backbone in the untrusted environment is developed using PyTorch 2.4.1, while the rectifier is implemented in C++ with Intel SGX SDK 2.25. To optimize the computation within the rectifier, we leveraged the Eigen linear algebra library to efficiently handle matrix operations, achieving approximately 20% speed up for large models [31].

2) Inference Overhead and Memory Usage: The inference time of GNNVault comprises of the backbone execution time, data transfer time, and the rectifier execution time. In Fig. 6 (top), we present a breakdown of the inference time, for three rectifier designs and different GNN models: $\mathcal{M}_1$ (Cora), $\mathcal{M}_2$ (CoraFull), and $\mathcal{M}_3$ (Amazon Computer). The parallel and cascaded rectifiers transfer all intermediate node embeddings from the normal world to the secure world, resulting in higher latency due to data transfer and execution inside the TEE. The series rectifier exhibits the smallest overhead since it only requires the last layer's embedding, leading to an overhead of approximately 52% to 131% compared with running an unprotected GNN with CPU. Note that the backbone can be accelerated using GPUs to further increase the inference speed.

We also present the enclave runtime memory usage required for different GNNVault configurations, as shown in Fig. 6 (bottom). As mentioned before, the memory allocated in the trusted enclave is limited by SGX's PRM and EPC. Therefore, it is essential to ensure that the rectifier operates effectively within the enclave's secure boundaries and does not exceed the recommended limits. The enclave memory usage is primarily for each layer's input features, adjacency matrix, and model parameters. Notably, the highest memory usage across all models and rectifier designs remains at only 41.6MB, well within the 96MB EPC threshold. This confirms the feasibility of running these configurations securely within the enclave. In contrast, the backbone's runtime memory usage for the three models significantly exceeds 128MB, underscoring the impracticality of executing a complete GNN model within the enclave due to memory constraints.

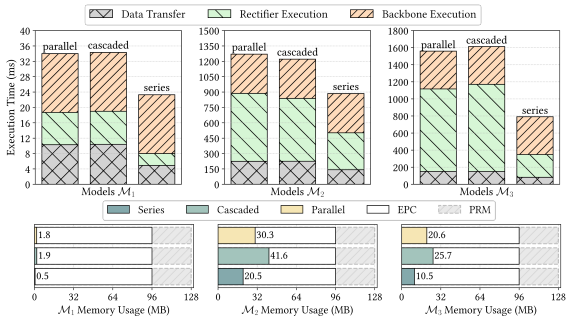


Fig. 6. GNNVault Inference Time Breakdown and Memory Usage

D. Security Analysis using Link Stealing Attacks

TABLE IV
LINK STEALING ATTACK PERFORMANCE ON GNNVAULT

Dataset	$\mathcal{M}_{org}$	$\mathcal{M}_{gv}$	$\mathcal{M}_{base}$	$\mathcal{M}_{org}$	$\mathcal{M}_{gv}$	$\mathcal{M}_{base}$	$\mathcal{M}_{org}$	$\mathcal{M}_{gv}$	$\mathcal{M}_{base}$
	Euclidean			Correlation			Cosine		
Cora	0.844	**0.702**	0.715	0.903	**0.735**	0.720	0.972	**0.765**	0.754
Citeseer	0.915	**0.750**	0.731	0.912	**0.778**	0.752	0.987	**0.807**	0.790
	Chebyshev			Braycurtis			Canberra		
Cora	0.847	**0.661**	0.691	0.902	**0.696**	0.693	0.933	**0.741**	0.717
Citeseer	0.908	**0.711**	0.698	0.953	**0.751**	0.732	0.976	**0.785**	0.746

Link stealing attacks [12], [23] aim to infer the adjacency matrix by exploiting the similarity between node embeddings, based on the assumption that GNN layers produce more similar embeddings for connected nodes than for unconnected ones. GNNVault isolates the private graph within the secure world and ensures one-directional communication from the normal world to the secure world. To verify the effectiveness of this isolation, we conduct link stealing attacks using six different similarity metrics and present the attack ROC-AUC scores in Table IV using all intermediate embeddings. A higher AUC score indicates greater leakage of private graph information. We use the attack performance on a DNN model with pure node features as the baseline ($\mathcal{M}_{\text{base}}$). The unprotected GNN ($\mathcal{M}_{org}$) shows very high AUC scores using all similarity metrics on both datasets, but with our GNNVault ($\mathcal{M}_{gv}$), the attack performance decreases to the level of the $\mathcal{M}_{base}$.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel secure GNN deployment strategy called GNNVault. Our approach involves strict isolation, training a public backbone model using only public data, and training a rectifier on private data to rectify the results from the backbone. The two spited models and their corresponding graph data are deployed in the untrusted environment and secure enclave, respectively. This method effectively safeguards sensitive adjacency information during GNN inference on edge devices. Given the growing trend of GNN local deployment, ensuring data privacy in such settings is of paramount importance. As part of our future work, we plan to explore and implement our strategy with additional GNN architectures, including GraphSAGE and GAT, to further enhance the security and broaden applicability.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under grants SaTC-1929300 and CNS-2212010.

REFERENCES

[1] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, and M. Shah, "A survey of on-device machine learning: An algorithms and learning theory perspective," *ACM Transactions on Internet of Things*, vol. 2, no. 3, pp. 1–49, 2021.

[2] A. Inc., "Core ml," https://developer.apple.com/documentation/coreml, 2023, accessed: Nov 07, 2024.

[3] L. Huawei Technologies Co., "Mindspore," https://www.mindspore.cn/, 2023, accessed: Nov 07, 2024.

[4] L. Zeng, C. Yang, P. Huang, Z. Zhou, S. Yu, and X. Chen, "Gnn at the edge: Cost-efficient graph neural network processing over distributed edge servers," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 3, pp. 720–739, 2022.

[5] J. Shao, H. Zhang, Y. Mao, and J. Zhang, "Branchy-gnn: A device-edge co-inference framework for efficient point cloud processing," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 8488–8492.

[6] O. Shchur and S. Günnemann, "Overlapping community detection with graph neural networks," *arXiv preprint arXiv:1909.12201*, 2019.

[7] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: A comprehensive graph neural network platform," *arXiv preprint arXiv:1902.08730*, 2019.

[8] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[9] M. Xue, J. Wang, and W. Liu, "Dnn intellectual property protection: Taxonomy, attacks and evaluations," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 455–460.

[10] C. Gongye, Y. Luo, X. Xu, and Y. Fei, "Side-channel-assisted reverse-engineering of encrypted dnn hardware accelerator ip and attack surface exploration," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4678–4695.

[11] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[12] X. He, J. Jia, M. Backes, N. Z. Gong, and Y. Zhang, "Stealing links from graph neural networks," in *30th USENIX security symposium (USENIX security 21)*, 2021, pp. 2669–2686.

[13] J. Guo and M. Potkonjak, "Watermarking deep neural networks for embedded systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[14] L. Wang, S. Xu, R. Xu, X. Wang, and Q. Zhu, "Non-transferable learning: A new approach for model ownership verification and applicability authorization," *arXiv preprint arXiv:2106.06916*, 2021.

[15] P. Xie, B. Wu, and G. Sun, "Bayhenn: Combining bayesian deep learning and homomorphic encryption for secure dnn inference," *arXiv preprint arXiv:1906.00639*, 2019.

[16] Z. Zhang, C. Gong, Y. Cai, Y. Yuan, B. Liu, D. Li, Y. Guo, and X. Chen, "No privacy left outside: On the (in-) security of tee-shielded dnn partition for on-device ml," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3327–3345.

[17] Z. Liu, Y. Luo, S. Duan, T. Zhou, and X. Xu, "Mirrornet: A tee-friendly framework for secure on-device dnn inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[18] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[19] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng, "Meta-gnn: On few-shot node classification in graph meta-learning," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2357–2360.

[20] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 346–353.

[21] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, "Shadownet: A secure and efficient on-device model inference system for convolutional neural networks," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1596–1612.

[22] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: towards model privacy at the edge using trusted execution environments," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 161–174.

[23] R. Ding, S. Duan, X. Xu, and Y. Fei, "Vertexserum: Poisoning graph neural networks for link inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4532–4541.

[24] I. Corporation, "Enclave memory measurement tool for intel® software guard extensions (intel® sgx) enclaves," https://www.intel.com/content/dam/develop/external/us/en/documents/enclave-measurement-tool-intel-sgx-737361.pdf, 2017, white Paper.

[25] V. Costan, "Intel sgx explained," *IACR Cryptol, EPrint Arch*, 2016.

[26] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International conference on machine learning*. PMLR, 2016, pp. 40–48.

[27] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[28] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," *arXiv preprint arXiv:1707.03815*, 2017.

[29] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*. IEEE, 2020, pp. 747–748.

[30] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[31] B. Jacob, G. Guennebaud, and et al., "Eigen: a c++ template library for linear algebra, matrices, vectors, numerical solvers, and related algorithms," https://eigen.tuxfamily.org.

Ensembler: Protect Collaborative Inference Privacy from Model Inversion Attack via Selective Ensemble

Dancheng Liu, Chenhui Xu, Jiajie Li, Amir Nassereldine, Jinjun Xiong
Department of Computer Science and Engineering
University at Buffalo, NY, USA
{dliu37, cxu26, jli433, amirnass, jinjun}@buffalo.edu

***Abstract*—For collaborative inference through a cloud computing platform, it is sometimes essential for the client to shield its sensitive information from the cloud provider. In this paper, we introduce Ensembler, an extensible framework designed to substantially increase the difficulty of conducting model inversion attacks by adversarial parties. Ensembler leverages selective model ensemble on the adversarial server to obfuscate the reconstruction of the client's private information. Our experiments demonstrate that Ensembler can effectively shield input images from reconstruction attacks, even when the client only retains one layer of the network locally. Ensembler significantly outperforms baseline methods by up to 43.5% in structural similarity while only incurring 4.8% time overhead during inference.**

I. Introduction

Deep learning models have achieved exceptional performance across critical domains [1], but their increasing size poses challenges for computationally constrained edge devices like mobile phones [2, 3]. When the model's size and complexity exceed the computational ability of the edge devices, a prevalent solution to reduce the workload on these resource-limited edge devices is through collaborative inference (CI) [4], as shown in the left side of Fig. 1. During inference, the client (edge device) computes the first few layers of the network and sends the intermediate features to the server (cloud), where the server computes the computationally expensive layers and returns the features or the results back to the client. However, this raises concerns about the privacy of sensitive client data, especially when the cloud server may be adversarial and tasks involve sensitive information like medical or facial data [5]. Consequently, there is a growing need for secure, accurate, and efficient machine learning frameworks that enable privacy-preserving collaboration between edge devices and cloud servers.

During CI, a classical yet powerful attack on the client's sensitive data is through model inversion attacks (MIA). As shown in the right side of Fig. 1, an adversarial server aims to inverse the client's private network to reconstruct the private input through intermediate features. We will elaborate on the details of MIA in Section II. He et. al. show that the client's data is especially vulnerable when the client retains only one

The research was supported in part by the National AI Institute for Exceptional Education (NSF Award #2229873), Center for Early Literacy and Responsible AI (IES Award #R305C240046), FuSe-TG (NSF Award #2235364) and SaTC (NSF Award #2329704). The opinions expressed are those of the authors and do not represent views of any sponsors.

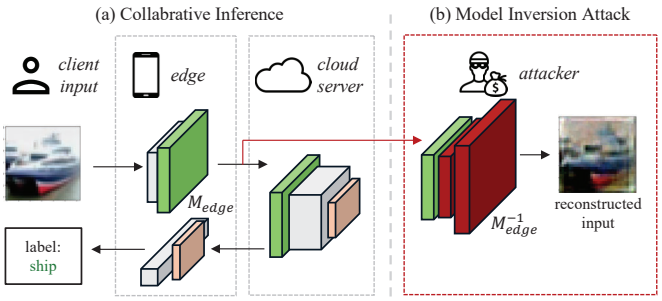


Fig. 1. An illustration of (a) collaborative inference, where the client offloads computation to the server; and (b) model inversion attack, where the adversarial server builds a decoder M_{edge}^{-1} to decode the client's input from the intermediate features computed by the client's private network M_{edge}.

single convolutional layer [4]. While the client can potentially keep more privacy as it keeps more layers, such a method is less practical in the real world due to the limitations of the computational power of edge devices. Shredder [4] uses a noise injection layer before the client sends out computed results to reduce mutual information between client and server while maintaining good classification accuracy. Nevertheless, Preva [5] demonstrates that Shredder falls short in safeguarding facial images from recovery. Through experimentation with the noise injection layer proposed by Shredder on a shallow layer, we observed significant accuracy drops with combined multiplicative and additive noise. On the other hand, simple additive noise with about a 3% drop in accuracy failed to protect images from recovery.

The failure of Shredder [6] could be attributed to two factors: the client's computational limit and the server's knowledge of the partial weights. To ensure the best defense, Shredder requires the noise injection layer to be very deep, but that results in the client taking heavy computation work and is less practical. Meanwhile, a shallow noise injection layer leaves most of the weights to the server. With rich information about public model weights available, the server can learn a strong and accurate representation of the private network held by the client, which leads to a successful MIA. Thus, the key challenge behind a practical CI framework is to defend against an adversarial server's reconstruction ability when the client only holds a minimal portion of the network.

To address the current deficiencies in privacy-preserving edge-cloud collaborative inference scenarios, in this work,

we propose a novel mechanism of "selective ensemble." Our intuition is as follows. Since the reconstruction ability of the adversarial server is powerful, it may be wiser to hide the client's private model than to defend it directly against the model reconstruction attack. We propose to leverage model ensemble on the cloud and secret private masking on the user side to achieve a "Schrödinger's model," where a reconstruction attack can lead to a successful, but different, shadow network than the client's true network.

Based on the above analysis, we propose *Ensembler*, a novel and secure CI framework designed to substantially increase the adversary's effort required to recover the client's input. The proposed architecture consists of multiple parallel networks, but only a subset of them is actually activated by the client during inference. An existing MIA on the network will lead to a seemingly successful reconstruction, but such reconstruction does not represent the client's real network weights. We make the following contributions in this paper:

- We identify and address the key challenge in practical private collaborative inference, i.e., when the model split location between the client and the server is shallow, a direct defense fails to work.
- We reveal that a secret selection of a subset of the networks in an ensemble architecture can be useful for defending against model inversion attacks.
- We propose *Ensembler*, a standalone framework that can not only significantly increase the adversary server's reconstruction difficulty but also be seamlessly integrated with existing complex algorithms to construct practical and private inference architectures.

To verify the effectiveness of *Ensembler*, we conduct experiments on multiple datasets. Compared to models without using model ensemble, when defending against MIA, *Ensembler* achieves up to 43.5% decrease in structural similarity and 40.5% decrease in peak signal noise ratio. In addition, compared to the previous popular approach, Shredder [6], *Ensembler* better protects the client's sensitive information with a 10.7% decrease in peak signal noise ratio, highlighting its effectiveness even as a standalone framework.

II. Background

A. Collaborative Machine Learning

The development of mobile graphic processing units (GPUs) has ushered in a new era where machine learning tasks are increasingly deployed with a portion of the computation being handled by edge devices. Related areas include federated learning, where multiple edge devices jointly train a DL model [7, 8]; split learning, where a DL model is split into two or more parts, and the client and server jointly train it [9]; and collaborative inference, where a DL model is split, with only a portion deployed on the server to provide services [4, 10]. In this paper, we will focus on the inference part and assume that the training phase of DL models is secure. Though the training phase is sometimes also susceptible to adversarial attacks aimed at stealing sensitive information [11, 12], private inference is still more prevalent in most practical scenarios.

There have been multiple works addressing this formidable challenge of safeguarding the client's sensitive information in collaborative inference scenarios. [13, 14] provides an extensive discussion on different algorithmic and architectural choices and their impacts on privacy protection. For simplicity and without loss of generalization, we will group existing approaches into two categories in this paper: encryption-based algorithms that guarantee privacy at the cost of thousands of times of time efficiency [15, 16, 17, 18], and perturbation-based algorithms that operate on the intermediate layers of a DL architecture, introducing noise to thwart the adversary's ability to recover client input [6, 10, 5, 19]. Since perturbation-based algorithms directly operate on the intermediate outputs from the client, they incur minimal additional complexity during the inference process. As opposed to guaranteed privacy provided by encryption-based algorithms, perturbation-based algorithms suffer from the possibility of privacy leakage, meaning sensitive private information may still be recoverable by the adversarial server despite the introduced perturbations, but they are nonetheless more practical to use due to the less additional costs associated with the methods.

B. Threat model

In this paper, we consider the collaborative inference task between the client and the server, which acts as a semi-honest adversarial attacker that aims to steal the raw input from the client. Formally, we define the system as a collaborative inference on a pre-trained DNN model, $\mathbf{M}(x, \theta)$, where the client holds the first and the last few layers (i.e. the "head" and "tail" of a neural network), denoted as $\mathbf{M}_{c,h}(x, \theta_{c,h})$ and $\mathbf{M}_{c,t}(x, \theta_{c,t})$. The rest of the layers of DNN are deployed on the server, denoted as $\mathbf{M}_s(x, \theta_s)$. θ is the trained weights of M, where $\theta = \{\theta_{c,h}, \theta_s, \theta_{c,t}\}$. The complete collaborative pipeline is thus to make a prediction of incoming image x with $\mathbf{M}_{c,t}[\mathbf{M}_s[\mathbf{M}_{c,h}(x)]]$.

During the process, the server has access to θ_s and the intermediate output $\mathbf{M}_{c,h}(x)$. In addition, we assume that it has a good estimate of the DNN used for inference. That is, it has auxiliary information on the architecture of the entire DNN, as well as a dataset in the same distribution as the private training dataset used to train the DNN. However, it does not necessarily know the hyper-parameters, as well as the engineering tricks used to train the model. Since the server is a computation-as-a-service (CaaS) provider, it is assumed to have reasonably large computation resources. While it is powerful in computing, the server is restricted from querying the client to receive a direct relationship between raw input x and intermediate output $\mathbf{M}_{c,h}(x)$.

To reconstruct raw input x from the intermediate output $\mathbf{M}_{c,h}(x)$, the server adopts a common model inversion attack [4, 5, 20]. To obtain x's reconstruction, MIA aims to obtain the inverse of $\mathbf{M}_{c,h}$, which will be denoted as $\mathbf{M}_{c,h}^{-1}$. The server constructs a shadow network $\tilde{\mathbf{M}}(x, \tilde{\theta_{c,h}}, \theta_s, \tilde{\theta_{c,t}})$: $\{\tilde{\mathbf{M}_{\mathbf{c,h}}}, \mathbf{M}_s, \tilde{\mathbf{M}_{\mathbf{c,t}}}\}$ such that $\tilde{\mathbf{M}}$ simulates the behavior of $\mathbf{M}$. After training $\tilde{\mathbf{M}}$, the adversarial server is able to obtain a representation $\tilde{\mathbf{M}_{\mathbf{c,h}}}$ such that $\tilde{\mathbf{M}_{\mathbf{c,h}}}(x) \sim \mathbf{M}_{c,h}(x)$. As the

next step, with the help of a decoder, $\tilde{\mathbf{M}}_{\mathbf{c,h}}^{-\mathbf{1}} \sim \mathbf{M}_{c,h}^{-1}$, to reconstruct the raw image from intermediate representation, it is able to reconstruct the raw input from $\mathbf{M}_{c,h}(x)$.

C. Assumptions of other related works

In this section, we provide an overview of various attack models and the assumptions adopted in other works related to collaborative inference (CI) under privacy-preserving machine learning (PPML). Since different works potentially use different collaboration strategies between the client and the server, we will use the generic notation, where $\mathbf{M}_c$ is held by the client, and $\mathbf{M}_s$ is held by the server. Generally, the attacks from the server will fall into three categories:

Membership Inference Attacks that try to predict if certain attributes, including but not limited to individual samples, distributions, or certain properties, are a member of the private training set used to train $\mathbf{M}(x, \theta)$. If successful, the privacy of the training dataset will be compromised. We refer readers to the survey by [21] and [22] for more details.

Model Inversion Attacks that try to recover a particular input during inference when its raw form is not shared by the client. For example, in an image classification task, the client may want to split $\mathbf{M}$ such that it only shares latent features computed locally to the server. However, upon successful model inversion attacks, the server will be able to generate the raw image for classification tasks based on the latent features. It is important to note that, in this paper, we adopt the same definition of model inversion attacks as of [4]. This term also refers to attacks that reconstruct the private training dataset in other works. We will focus on reconstructing private raw input for the rest of this paper.

Model Extraction Attacks that try to steal the parameters and even hyper-parameters of $\mathbf{M}$. This type of attack compromises the intellectual property of the private model and is often employed as sub-routines for model inversion attacks when the server lacks direct access to the private model's parameters.

Different works also make different assumptions about the capability of the server. First, it is widely accepted that the server has sufficiently yet reasonably large computing power and resources, as its role is often providing ML service. Regarding the auxiliary information on $\mathbf{M}$, they generally fall into three levels:

White Box assumes that the server has full access to architecture details of $\mathbf{M}$ such as the structure and parameters [23]. Different definitions also add different auxiliary available information such as training dataset [23], corrupted raw input [24], or a different dataset [25]. This setting is often associated with attacks that try to reconstruct private training datasets [25, 24, 26].

Black Box assumes that server does not have any information on both $\mathbf{M}$ and the training dataset. However, it is allowed to send unlimited queries to the client to get $\mathbf{M}_c(x)$ [27, 28].

Query-Free restricts the server from querying $\mathbf{M}_c$. While such an assumption greatly limits the reconstruction ability of the adversarial party, there are no limitations on auxiliary information available to the server besides the actual weights of $\mathbf{M}_c$. [4, 29] have both shown that $\mathbf{M}_c$ is still vulnerable to leaking private information of its input when the server has information on the model architecture and training dataset. Our work uses this setting.

III. *Ensembler* ARCHITECTURE

While it is possible to protect sensitive data by increasing the depth of the client network, such depth is often impractical for edge devices due to the computational demands involved. In this section, we present *Ensembler*, a framework that augments the privacy of intermediate information sent by the client without requiring extra computation efforts of the client during inference. *Ensembler* is highly extensible, and it is compatible with existing works that apply noises and perturbation during both DNN training and inference. We will go over the inference pipeline of *Ensembler* in Section III-A, the training stage of this new framework in Section III-C, an intuitive explanation of why *Ensembler* works in Section III-B, and finally the time complexity analysis in Section III-D.

A. Inference Pipeline

As illustrated in Figure 2, *Ensembler* leverages model ensemble on the server to generate a regularized secret $\mathbf{M}_{c,h}$ that is hard to be reconstructed by the adversarial server. It comprises three parts: standard client layers, N different server nets, and a Selector. During the CI pipeline, the client computes the addition of $\mathbf{M}_{c,h}(x)$ and a predefined noise, and transmits the intermediate output to the server. The server then feeds the intermediate output through each of the $\mathbf{M}_s^i$ and reports the output of each $\mathbf{M}_s^i$ to the client. The client then employs a user-defined private selector to perform a selection of the feedback from the server, which activates the results of P out of N nets and combines them. As a final step, it computes the last few layers ($\mathbf{M}_{c,t}(x)$) to classify the input image. We will introduce these separately in this section.

1) Client Layers: During collaborative inference, the client runs a part of the DNN. Under the proposed framework, the client is responsible for running the first h layers $\mathbf{M}_{c,h}$ and the last t layers $\mathbf{M}_{c,t}$. These layers are the same as the client part of a typical collaborative inference framework. $\mathbf{M}_{c,h}$ takes the raw input (often an image) and outputs the intermediate result, whereas $\mathbf{M}_{c,t}$ takes the output from the server as input and outputs the likelihood of each class.

2) Server Nets: On the server side, the network consists of N copies of DNN, with each $M_s^i \in M_s$ corresponding to what the server would normally process in a typical CI pipeline. That is, each $M^i : \{M_{c,h}, M_s^i, M_{c,t}\}$ is a standard pipeline for the inference task. Upon receiving $\mathbf{M}_{c,h}(x) + N(0, \sigma)$, which is the intermediate features generated by the client, the server shall feed this input into each of M_s^i. It outputs N representations of hidden features used for classification and sends all of them back to the client. We denote those outputted feature maps as $M_s(x') = \{M_s^i(x')\}$.

3) Selector and $\mathbf{M}_{c,t}$: To increase the difficulty of the server reconstructing the model and recovering the raw input, a user-defined private selector is applied to the outputs returned

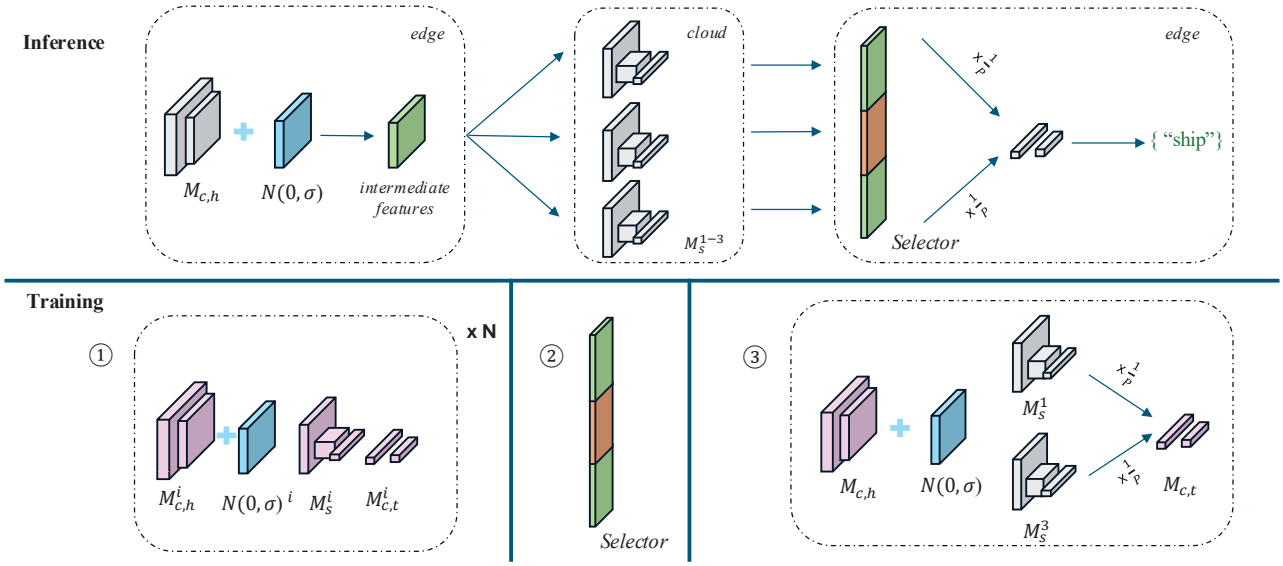


Fig. 2. Illustration of the proposed architecture, *Ensembler*, during inference and training. Unlike traditional CI pipelines, during inference, *Ensembler* deploys N neural networks on the server and uses a private selector to activate P of the N nets, making the reconstruction of the client network infeasible. During training, *Ensembler* follows a three-stage training procedure, and the purple elements are trainable parameters of each stage.

by the server before the client runs $\mathbf{M}_{c,t}$. The selector is a secret activation function, activating a subset of $M_s(x')$. We denote this subset of activated feature maps as $M_s(x')_p$, and there are in total P activated feature maps. Then, the Selector is defined by Eq. 1, where S_i is the activation from the Selector, and $\odot$ is the element-wise multiplication. For simplicity, we consider $S_i = 1/P$. In other words, the client will normalize P of the N feature maps returned by the adversarial server and concatenate them as the input features to $\mathbf{M}_{c,t}$ while keeping those selections private to the server.

$$Sel[M_s(x)] = Concat[S_i \odot f \quad \forall f \in M_s(x')_p] \tag{1}$$

B. Intuition behind Ensembler

In this section, we provide an intuitive explanation of *why* the proposed architecture can defend the client's private information against MIA through a simulated MIA performed by the adversarial server. As defined by the threat model in Section II-B, the server has access to the training dataset, model architecture, and M_s. To perform a MIA, it will construct $\tilde{\mathbf{M}}_{\mathbf{c,h}}$ and $\tilde{\mathbf{M}}_{\mathbf{c,t}}$. Now, let's further assume that each of $M_s \in \{M_s^i\}$ has different weights (how to achieve this will be shown in the next subsection). With brute-forcing all combinations being exponential and unpractical, and since the adversarial server does not know the user-defined selector, there remain two possible MIA constructions. The server could either construct the shadow network with a random subset of $\{M_s^i\}$, or it could use the entire $\{M_s^i\}$. We will show that both methods result in $\tilde{\mathbf{M}}_{\mathbf{c,h}}(x) \not\sim \mathbf{M}_{c,h}(x)$. Without loss of generality, for the first approach, the server is assumed to construct $\tilde{\mathbf{M}}_{\mathbf{c,h}}$ from a single M_s^i that is a subset of the networks selected by the Selector.

Proposition 1: Reconstructing input from single neural network $\mathbf{M}_s^i$ results in $\tilde{\mathbf{M}}_{c,h}(x) \not\sim \mathbf{M}_{c,h}(x)$.

Let's consider the most extreme case, where the Selector selects $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$, and the adversarial server is constructing its shadow network with $\mathbf{M}_s^1$. Since $\mathbf{M}_{c,h}$ is trained with both networks, the gradient of weights of $\mathbf{M}_{c,h}$ is calculated as (roughly) $\eta(\nabla_x \mathbf{M}_s^1 + \nabla_x \mathbf{M}_s^2)$ during backpropogation. After minimization, as long as $\nabla_x \mathbf{M}_s^2$ is not zero, the weights of $\mathbf{M}_{c,h}$ will be affected by both $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$ (we will show how to let the gradient not be zero through loss regularization in the next section). Since $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$ have different weights, their gradient will point to different directions. A shadow network $\tilde{\mathbf{M}}_{c,h}$ trained on only $\mathbf{M}_s^1$ will represent different weights compared to $\mathbf{M}_{c,h}$, and thus the decoder based on $\tilde{\mathbf{M}}_{c,h}$ will also reconstruct incorrect client's secret input.

Proposition 2: Reconstructing input from the entire set $\{\mathbf{M}_s\}$ results in $\tilde{\mathbf{M}}_{c,h}(x) \not\sim \mathbf{M}_{c,h}(x)$.

A similar conclusion from the analysis in the first case may be derived. Let's still use the simple example of $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$. This time, the Selector selects only $\mathbf{M}_s^1$, but the server decides to train on both $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$. It is not hard to see that the weights of $\mathbf{M}_{c,h}$ and $\tilde{\mathbf{M}}_{c,h}$ are just swapped with each other, and now, the decoder is still based on incorrect weights and still leads to an incorrect reconstruction of $\mathbf{M}_{c,h}(x)$.

C. Training Stages

As mentioned above, the key idea behind *Ensembler* is to use N **different** networks with a private Selector to obfuscate MIA. To achieve the N different networks and make sure $M_{c,h}$ is properly trained with all of the selected subsets, the proposed architecture uses a three-staged training pipeline, as shown in the bottom half of Fig. III.

Stage 1: First, *Ensembler* obtains N different networks by injecting N different noises after each of the network's $M_{c,h}^i$ and training each of the networks. Since randomly initialized noises are quasi-orthogonal to each other and each network

needs to adjust its weights to reverse the effects of the added noise on the intermediate feature map, the resulting weights of $M^i_{c,h}$ will be different from each other. Formally, the first stage optimizes Eq. 2, where $N(0,\sigma)^i$ is a fixed Gaussian noise added to the intermediate output. We choose Gaussian noise because of simplicity in implementation, and we argue that any method leading to distinctive $M_{c,h}$ will be sufficient.

$$L^i_\theta = -\sum_j y_j * log\ M^i_{c,t}(M^i_s[M^i_{c,h}(x) + N(0,\sigma)^i])_j \quad (2)$$

Stage 2: After the first training stage, N different DNNs are obtained. The client secretly selects P out of the N networks as the activation of the Selector.

Stage 3: With the Selector defined, the client is now able to aggregate P networks together to form the "ensembled" inference pipeline. At the last stage of the training process, P server networks together form the body of the final network, and their weights are frozen. The client re-trains its private network $M_{c,h}$ and $M_{c,t}$, with a newly defined Gaussian noise added to the output features of $M_{c,h}$. This Gaussian noise follows the same definition as the Gaussian noise in Stage 1's training. Formally, this stage's training loss follows Eq. 3 by adding a high penalty to the model when the gradient descends only to the direction of some single server net M^i_s. It is worth noting that this loss has two terms. The first part is a standard cross-entropy loss used to train a classification network, and the second part is a regularization term to enforce M to learn a joint $M_{c,h}$ and $M_{c,t}$ representation from all of the P server networks. In the equation, CS is the cosine similarity, and λ is a hyperparameter to control the regularization strength. The idea behind such a regularization is that if $M_{c,h}$ only learns from one single network, then its weights will be very similar to one of the previous $M^i_{c,h}$. Thus, we regularize it to be as quasi-orthogonal to all of the previous $M^i_{c,h}$.

$$\begin{aligned} L_\theta &= \sum_i^{i\in P} CE + Regularization, \quad where \\ CE &= -\sum_j [y_j * log M_{c,t}(Sel[M^i_s[M_{c,h}(x) + N(0,\sigma)]])_j] \\ Regularization &= \lambda \max_i [CS(M_{c,h}(x), M^i_{c,h}(x))] \end{aligned} \quad (3)$$

D. Time Complexity of Ensembler

From the previous subsections, it is clear that the inference time complexity of the proposed framework on the client is the same as the standard non-private inference pipeline. On the server, it is N times the individual network on a single-core GPU, and there is negligible extra communication cost between the client and the server. However, it is noteworthy to emphasize that since each M^i_s is independent of each other, the proposed framework is friendly to parallel execution and even multiparty (multi-server) inference. Under those settings, the theoretical time complexity of $O(N)$ would be replaced with lower practical time costs.

Regarding the MIA's time complexity, following the analysis in Section III-B, it is not hard to see that the expected MIA time of a single server is $O(2^N)$. Since an arbitrary reconstruction on some subset of the network will lead to a reasonable shadow network $\tilde{\mathbf{M}}_{\mathbf{c,h}}$, the server has no way of telling whether its reconstruction is an actual representation of the client's network. Thus, to ensure optimal reconstruction, it must use brute force and try all possibilities. In addition, under multiparty inference scenarios, without access to a subset of the activated networks, the adversarial server will not be able to perform the desired reconstruction.

IV. Experiments and Evaluations

A. Implementation details

During the experiment, we consider the most strict setting, where h=1 and t=1 on a ResNet-18 [31] architecture for three image classification tasks, CIFAR-10 [32], CIFAR-100 [32], and a subset of CelebA-HQ [33]. That is, the client only holds the first convolutional layer as well as the last fully connected layer, which is also the minimum requirement for our framework. For CIFAR-10, the intermediate output's feature size is [64x16x16], for CIFAR-100, we remove the MaxPooling layer and the intermediate output's feature size is [64x32x32], and for CelebA, the intermediate output's feature size is [64x64x64]. We consider the ensembled network to contain 10 neural networks (N=10), each being a ResNet-18. The selector secretly selects $\{4,3,5\}$ out of the 10 nets (P=$\{4,3,5\}$), respectively. The adversarial server is aware of the architecture and the training dataset. It constructs a shadow network $\tilde{\mathbf{M}}_{c,h}$ consisting of three convolutional layers with 64 channels each, with the first one simulating the unknown $\mathbf{M}_{c,h}$, and the other two simulating the Gaussian noise added to the intermediate output. It also has $\tilde{\mathbf{M}}_{c,t}$ with the same shape as $\mathbf{M}_{c,t}$. An adaptive shadow network learns from all 10 server nets with an additional activation layer that is identical to the selector. For any noises added to the intermediate outputs during the training and inference stage, we consider a fixed Gaussian noise $\mathbf{g} \sim N(0, 0.1)$.

B. Experiment Setup

We employ two key metrics: Structural Similarity (SSIM) and Peak Signal Noise Ratio (PSNR). Higher SSIM and PSNR values indicate better reconstruction quality and thus worse defense. In addition, we report the changes in classification accuracy (ΔAcc) after adding the defense mechanism. As our proposed architecture operates in parallel with existing perturbation methods, we mainly compare our method with its single network counterpart (Single) [30], where only one $M^i : \{M_{c,h}, M^i_s, M_{c,t}\}$ is used. In the ablation study, we provide a deeper analysis using CIFAR-10, and we compare the following methods against *Ensembler*: no protection (None), training the noise injected to Single (Shredder [6]), adding dropout layer in the single network or ensembled network, but without the first stage training (DR-single and DR-10) [34]. For the proposed architecture, we evaluate the performance of both reconstruction attacks discussed in Section III-B. For reconstruction with a single server net, we report the best reconstruction, or worst defense, in terms of SSIM (Ours -

TABLE I

DEFENSE QUALITY OF *Ensembler* AND SINGLE [30] ACROSS DATASETS. FOR SSIM AND PSNR, LOWER VALUES INDICATE BETTER DEFENSE QUALITY. WE REPORT ON THE TWO RECONSTRUCTION ATTACKS DISCUSSED IN SECTION III-B. FOR RECONSTRUCTING WITH A SINGLE NETWORK, WE REPORT THE STRONGEST RECONSTRUCTION ATTACK (LEAST DEFENSE QUALITY) OUT OF N NETWORKS.

	CIFAR-10			CIFAR-100			CelebA-HQ		
Name	ΔAcc	SSIM ↓	PSNR ↓	ΔAcc	SSIM ↓	PSNR ↓	ΔAcc	SSIM ↓	PSNR ↓
Single	2.15%	0.39	7.53	-0.97%	0.46	8.52	-1.24%	0.27	14.31
Ours - Adaptive	-2.13%	0.06	5.98	0.31%	0.09	4.77	2.39%	0.09	13.37
Ours - SSIM	-2.13%	0.29	4.87	0.31%	0.26	5.07	2.39%	0.18	12.06
Ours - PSNR	-2.13%	0.22	5.53	0.31%	0.26	5.07	2.39%	0.18	12.06

TABLE II

DIFFERENT DEFENSE MECHANISMS WITH CIFAR-10. THE LAST THREE ARE THE PROPOSED FRAMEWORK. THE UNDERSCORED NUMBER REPRESENTS THE BEST DEFENSE FROM OTHER EXISTING METHODS.

Name	Δ Acc	SSIM ↓	PSNR ↓
None	0.00%	0.49	9.86
Shredder	-2.92%	<u>0.29</u>	6.70
Single	2.15%	0.39	7.53
DR-single	2.70%	0.35	<u>6.67</u>
DR-10 - SSIM	1.42%	0.37	7.35
DR-10 - PSNR	1.42%	0.32	7.96
Ours - Adaptive	-2.13%	0.06	5.98
Ours - SSIM	-2.13%	0.29	4.87
Ours - PSNR	-2.13%	0.22	5.53

SSIM) and PSNR (Ours - PSNR) among the N networks. For the reconstruction using all networks, we denote it as (Ours - Adaptive). We run the experiments on an A6000 GPU server using Python and PyTorch. For Section 4, we used a mixture of A6000 and T4 GPU. In addition, we report the real-world time delay of the proposed method compared with its single counterpart and a recent encryption-based method [35]. We evaluate the inference time on a Raspberry Pi communicating with the A6000 server over a wired network.

C. Comparison of Results

We provide the quantitative evaluations of *Ensembler* compared to its non-ensembled counterpart (Single [30]) in Table I. It could be seen that the proposed framework significantly decreases the reconstruction quality of the adversarial party. *Ensembler* incurs a 2.13% drop in classification accuracy compared to the model without any protection, which is acceptable compared to its advantage in protecting the privacy of the client's raw input. It should be noted that the adaptive MIA strategy performs worse than the best reconstruction with a single network, which is expected. While Equation 3 constraints $M_{c,h}$ to learn from all server networks, it might still have one that is more "favored". Thus, a reconstruction based on that favored network yields the best results. On the other hand, a reconstruction based on all networks might not select the same one as its "favored" network, which causes the shadow network's weights to significantly deviate from $M_{c,h}$.

Moreover, we delve into a more thorough analysis of *Ensembler*'s performance on the CIFAR-10 dataset compared to other existing methods. The results are summarized in Table II. It could be observed that *Ensembler* defends the client's input on par with Shredder [6] on structural similarity, and exceeds it on peak signal-noise ratio. In addition, it should be noted that Shredder [6] and dropout defense [34] can be combined with *Ensembler* together. The additive noise $N(0, \sigma)$ in the third stage could be replaced by Shredder's trained noise, and dropout can also be added to the network's FC layer to perform further protection of the client's sensitive data.

TABLE III

TIME TAKEN (SECONDS) TO RUN A BATCH OF 128 IMAGES THROUGH DIFFERENT DEFENSE STRATEGIES.

Name	Client	Server	Communication	Total
Standard CI	0.66	0.98	2.30	3.94
Ensembler	0.66	1.02	2.45	4.13
STAMP [35]	–	–	–	309.7

D. Latency

Time required by the standard CI setting and *Ensembler* is provided in Table III. In addition, STAMP [35] is a hardware-optimized version of encryption-based private inference, and we show their reported results (LAN-GPU). For all experiments, we use the setting in STAMP to run ResNet-18 on a batch size of 128 images. *Ensembler* incurs only **0.19 seconds (4.8%)** overhead compared with running the standard CI pipeline, but with much stronger protection on the client's sensitive data. It can also be observed that the increment of time comes from two parts, with more server computing taking the minor role and more communication between the client and the server taking the major role. In the future, more attention on improving client-server communication is pivotal for a more efficient and practical collaborative inference.

V. CONCLUSION

During collaborative inference, the client's sensitive information is prone to model inversion attacks from the adversarial server, especially when the client only holds a very small portion of the network. In this paper, we present a novel collaborative inference framework, *Ensembler*, designed to significantly increase the complexity of reconstruction for adversarial parties. *Ensembler* leverages a selective model ensemble on the adversarial server and a secret Selector to defend against the server's model inversion attacks. In addition, it seamlessly aligns with existing methods that introduce diverse forms of noise to intermediate outputs, potentially yielding robust and adaptable architectures if combined with them. Lastly, *Ensembler* is a latency-friendly framework that incurs only marginal overhead during inference. Our experiments highlight the substantial deterioration in reconstruction quality for images safeguarded by *Ensembler* when compared to those without its protection.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[2] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, "Model complexity of deep learning: A survey," *Knowledge and Information Systems*, vol. 63, pp. 2585–2619, 2021.

[3] R. Qin, D. Liu, C. Xu, Z. Yan, Z. Tan, Z. Jia, A. Nassereldine, J. Li, M. Jiang, A. Abbasi *et al.*, "Empirical guidelines for deploying llms onto resource-constrained edge devices," *arXiv preprint arXiv:2406.03777*, 2024.

[4] Z. He, T. Zhang, and R. B. Lee, "Model inversion attacks against collaborative inference," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 148–162.

[5] R. Lu, S. Shi, D. Wang, C. Hu, and B. Zhang, "Preva: Protecting inference privacy through policy-based video-frame transformation," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, 2022, pp. 175–188.

[6] F. Mireshghallah, M. Taram, P. Ramrakhyani, A. Jalali, D. Tullsen, and H. Esmaeilzadeh, "Shredder: Learning noise distributions to protect inference privacy," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 3–18.

[7] X. Yu, L. Cherkasova, H. Vardhan, Q. Zhao, E. Ekaireb, X. Zhang, A. Mazumdar, and T. Rosing, "Async-hfl: Efficient and robust asynchronous federated learning in hierarchical iot networks," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, 2023.

[8] D. Yaldiz, T. Zhang, and S. Avestimehr, "Secure federated learning against model poisoning attacks via client filtering," in *ICLR 2023 Workshop on Backdoor Attacks and Defenses in Machine Learning*, 03 2023.

[9] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *arXiv preprint arXiv:1912.12115*, 2019.

[10] S. Osia, A. Taheri, A. Shahin Shamsabadi, K. Katevas, H. Haddadi, and H. Rabiee, "Deep private-feature extraction," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[11] J. Li, A. S. Rakin, X. Chen, Z. He, D. Fan, and C. Chakrabarti, "Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10 194–10 202.

[12] W. Zhang, S. Tople, and O. Ohrimenko, "Leakage of dataset properties in {Multi-Party} machine learning," in *30th USENIX security symposium (USENIX Security 21)*, 2021.

[13] R. Xu, N. Baracaldo, and J. Joshi, "Privacy-preserving machine learning: Methods, challenges and directions," *arXiv preprint arXiv:2108.04417*, 2021.

[14] C. Xu, F. Yu, Z. Xu, N. Inkawhich, and X. Chen, "Out-of-distribution detection via deep multi-comprehension ensemble," in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 21–27 Jul 2024, pp. 55 465–55 489.

[15] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX secur. symp*, vol. 3, 2019.

[16] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[17] K.-Y. Lam, X. Lu, L. Zhang, X. Wang, H. Wang, and S. Q. Goh, "Efficient fhe-based privacy-enhanced neural network for trustworthy ai-as-a-service," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[18] J. Li and J. Xiong, "xmlp: Revolutionizing private inference with exclusive square activation," *arXiv preprint arXiv:2403.08024*, 2024.

[19] W. Sirichotedumrong and H. Kiya, "A gan-based image transformation scheme for privacy-preserving deep neural networks," in *2020 28th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 745–749.

[20] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4829–4837.

[21] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, "Membership inference attacks on machine learning: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, 2022.

[22] A. Salem, G. Cherubin, D. Evans, B. Köpf, A. Paverd, A. Suri, S. Tople, and S. Zanella-Béguelin, "Sok: Let the privacy games begin! a unified treatment of data inference privacy in machine learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 327–345.

[23] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, and A. V. Vasilakos, "Privacy and security issues in deep learning: A survey," *IEEE Access*, vol. 9, pp. 4566–4593, 2021.

[24] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The secret revealer: Generative model-inversion attacks against deep neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 253–261.

[25] Q. Wang and D. Kurz, "Reconstructing training data from diverse ml models by ensemble inversion," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2909–2917.

[26] N. Haim, G. Vardi, G. Yehudai, O. Shamir, and M. Irani, "Reconstructing training data from trained neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 911–22 924, 2022.

[27] Y. Xu, X. Liu, T. Hu, B. Xin, and R. Yang, "Sparse black-box inversion attack with limited information," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[28] M. Kahla, S. Chen, H. A. Just, and R. Jia, "Label-only model inversion attacks via boundary repulsion," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 15 045–15 053.

[29] S. Ding, L. Zhang, M. Pan, and X. Yuan, "Patrol: Privacy-oriented pruning for collaborative inference against model inversion attacks," 2023.

[30] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 2006, pp. 265–284.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[32] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[33] H. Zhu, W. Wu, W. Zhu, L. Jiang, S. Tang, L. Zhang, Z. Liu, and C. C. Loy, "CelebV-HQ: A large-scale video facial attributes dataset," in *ECCV*, 2022.

[34] Z. He, T. Zhang, and R. B. Lee, "Attacking and protecting data privacy in edge–cloud collaborative inference systems," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9706–9716, 2021.

[35] P. Huang, T. Hoang, Y. Li, E. Shi, and G. E. Suh, "Efficient privacy-preserving machine learning with lightweight trusted hardware," *arXiv preprint arXiv:2210.10133*, 2022.

Cross-Attention for AES Mode Variation in Side-Channel Analysis

Fanliang Hu*†, Jian Shen*, Haoyu Ma‡, and Qingming Jonathan Wu§
* School of Information Science and Engineering, Zhejiang Sci-Tech University
† School of Computer Science, Nanjing University of Information Science and Technology
‡ School of Cyberspace Science and Techonology, Beijing Jiaotong University
§ School of Electrical and Computer Engineering, University of Windsor
fanliang_1@163.com, s_shenjian@126.com, ma-haoyu@163.com, jwu@uwindsor.ca

***Abstract*—Portability poses a significant challenge for Deep Learning (DL)-based profiling Side-Channel Analysis (SCA) on AES encryption, as attackers cannot always ensure that training and target samples use the same encryption mode. To address this, we propose an Unsupervised Domain Adaptation (UDA) DL-SCA framework for achieving effective and robust cross-encryption-mode attacks. By incorporating cross-attention and UDA techniques, our framework aligns high-dimensional input samples, reducing interference from encryption mode mismatches. Evaluation across five distinct AES modes demonstrates that our method achieves robust SCA performance without requiring prior knowledge or multiple labeled datasets for analysis.**

***Index Terms*—side channel analysis, cross-attention mechanisms, unsupervised domain adaptation, deep learning.**

I. Introduction

Since Kocher's pioneering 1996 study on timing attacks [1], Side-Channel Analysis (SCA) has emerged as a critical technique for extracting sensitive information through the analysis of physical leakages [2]–[4]. One of the most powerful type of SCA, namely Profiling SCA, works by modeling leakages from similar devices to extract data from target devices [5], [6]. This approach was pioneered by Template Attacks (TA) [5], [7] and later enhanced by machine learning methods, such as Support Vector Machines (SVM) [8] and Random Forests (RF) [9]. The introduction of Deep Learning (DL) has further advanced profiling SCA. Specifically, deeper models such as Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP) are capable of handling complex inputs more efficiently, allowing SCAs powered by such techniques to provide better results with regard to both the overall generalization capability and the performance in managing masked data [10]–[12].

In a DL-based SCA (henceforth DLSCA) against the AES encryption algorithm, the attacker's goal is to infer the secret key by correlating side-channel measurements from the encryption process with a leakage model (trained in the off-line reconnaissance phase with data collected from attacker-controlled devices) [4], [13]. Intuitively, this attack strategy is more likely to succeed if training samples of the leakage model is collected from the exact same hardware/software configure with the targeted devices, and we have indeed witnessed extensive DLSCA studies which only considered the scenario where both the reconnoitered devices and the targeted ones run AES in the same encryption mode [14]–[16]. Unfortunately, practical attack scenarios could be more generalized, involving target devices that are different from the reconnoitered one at hardware-level, or alternatively, operate with AES modes different from the one observed during the off-line reconnaissance.

Related Work. Attentions on the portability of DLSCA, as well as the mode-specific AES vulnerabilities, has emerged only recently. While Wang *et al.* [17] analyzed the impact of mode diversity on DLSCA, Perusheska *et al.* [18] further highlighted the effectiveness of DLSCA using plaintext-ciphertext analysis across multiple AES modes. Other recent advancements focused on the scenario of across-device DLSCAs. Das *et al.* [19] demonstrated that training on diverse hardware in X-DeepSCA analysis enhances cross-device performance. Golde *et al.* [20] applied preprocessing methods, including dynamic time warping and Principal Component Analysis (PCA), with a MLP to mitigate cross-device discrepancies. Bhasin *et al.* [21] addressed portability challenges with a Multi-Device Model (MDM) that leverages three devices for training, validation, and attack. Cao *et al.* employed adversarial learning [22] and domain adaptation [23] to counteract device-based performance degradation, while Yu *et al.* [24] introduced meta-transfer techniques to address leakage inconsistencies in DLSCA across different device types. Nevertheless, researches on DLSCA that generalizes across multiple AES encryption modes remain limited. This gap underscores the need to study the possibility of building robust leakage models adaptable to diverse AES configurations, which could be the last piece of the puzzle on the completeness of DLSCA.

Contributions. Motivated by the current state-of-the-arts, we propose CA-SCA (Cross-Attention Side-Channel Analysis), a novel approach aiming to address the challenges in launching DLSCA against AES algorithms with the presence of mode variations. At the model design level, CA-SCA integrates cross-attention mechanisms with unsupervised domain adaptation (UDA) in order to have different sets of side-channel samples (which are potentially of different distributions) aligned at high-dimensional space to subsequently enable interactive training. To the best of our knowledge, while cross-attention has been widely used in multimodal [25] and

reinforcement learning [26] tasks [27], its application for AES mode adaptation in the SCA community remains unexplored. Thus our effort could be the first to make such an attempt.

In summary, the main contributions of this work are:

- We propose CA-SCA (Cross-Attention Side-Channel Analysis), a novel SCA approach that leverages cross-attention mechanisms and unsupervised domain adaptation to mitigate interference from mismatching of encryption modes, while requiring no specific preprocessing, fine-tuning, labeled attack traces or multiple analysis datasets.
- We evaluated CA-SCA on a software implementation of AES-128 across five encryption modes (ECB, CBC, CTR, CFB, and OFB) using the STM32F3 chip. The results demonstrate that CA-SCA reliably recovers keys across different encryption modes within 500 attack traces, outperforming current state-of-the-art methods[1].
- We present a comprehensive comparison of existing theoretical and experimental results on portable attacks using CA-SCA. Experimental results indicate that X-Deep and MDM may fail when encryption mode differences are substantial. In contrast, CA-SCA, using only a single mode as the analysis dataset, enables previously infeasible attacks.

The paper is organized as follows: Section II provides background knowledge and introduces the evaluation metrics. Section III outlines the proposed methodology. Section IV describes the experimental setup and presents the attack results, while Section V analyzes the impact of two key parameters on CA-SCA. Finally, Section VI concludes the paper and suggests future research directions.

II. Background

A. DLSCA

DLSCA differs from TA [5] for not requiring extensive prior knowledge. Instead, deep learning models automatically learn the relationship between physical leakages (traces) and labels, classifying traces through a neural network that maps them to labels based on the leakage patterns of sensitive operations.

During training, the neural network learns a classification function $\mathcal{F}_\theta$ that maps input traces $\mathbf{T}_i$ to score vectors $\vec{p} = \mathcal{F}_\theta(\mathbf{T}_i)$, where S is the number of possible classes. The network parameters θ are updated via backpropagation and an optimizer to minimize the batch loss:

$$\theta_{\text{epoch}} = \theta_{\text{epoch-1}} - \alpha \nabla_\theta L_{\text{batch}}, \tag{1}$$

where α is the learning rate. In the attack phase, the trained model classifies each attack trace and generates score vectors for key guessing. For AES in specific, in the first and last rounds of the en-/decryption, inference of the the neural network can be performed byte-by-byte (8 bits, yielding 256 possible values). This makes DLSCA particularly effective in attacks this cryptographic primitive.

[1]https://github.com/hfl110529/CA-SCA

B. Unsupervised Domain Adaptation

Unsupervised Domain Adaptation (UDA) [28] is a key technique in deep learning for improving model performance in unfamiliar domains, particularly when labeled data is scarce. The core idea is to leverage unlabeled data, enabling the model to learn general features across domains or modes, thereby enhancing robustness and adaptability to unseen data.

A common UDA method is Maximum Mean Discrepancy (MMD) [23], [29], which measures feature distribution differences between the source (X_s) and target (X_t) domains as:

$$\text{MMD}(X_s, X_t) = \left\| \mathbb{E}_{X_s}[\phi(X_s)] - \mathbb{E}_{X_t}[\phi(X_t)] \right\|^2, \tag{2}$$

where $\phi(\cdot)$ maps inputs to a high-dimensional space. By minimizing the combined loss function:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{classification}} + \lambda \cdot \mathcal{L}_{\text{MMD}}(X_s, X_t), \tag{3}$$

we jointly optimize the classification loss ($\mathcal{L}_{\text{classification}}$) and domain alignment loss, enhancing the model's generalization in the target domain. Here, λ is a weighting parameter that balances the MMD loss and classification loss.

C. Evaluation metric

Guessing Entropy (GE) [30] is a common security metric in SCA. Given N_{attack} traces in the attack phase, the attacker estimates a key guessing vector $\mathbf{k}_{\text{guess}} = [k_1, k_2, \ldots, k_{|\mathcal{K}|}]$, where $|\mathcal{K}|$ denotes the size of the keyspace, arranged in decreasing order of likelihood. When the rank of the true key k^* reaches zero in $\mathbf{k}_{\text{guess}}$, the number of traces used is referred to as $\bar{N}_{\text{GE}}$.

III. Methodology

Recall that CA-SCA is designed based on the combination of cross-attention mechanisms and UDA to enhance network portability. The core idea is to enhance the network's generalization across different data domains by leveraging attention mechanisms to emphasize relevant features, while domain adaptation techniques align analysis and target data in a high-dimensional space. In Subsection III-A outlines the threat model for the proposed approach, Subsection III-B discusses the rationale behind the attention mechanism, and Subsection III-C details the implementation of CA-SCA.

A. Threat Model

We consider a standard profiling side-channel setting focused on power and electromagnetic (EM) SCAs to recover secret keys from encryption algorithms. This threat model is widely recognized and realistic, as numerous certification laboratories evaluate security-critical products under similar conditions daily. Power and EM side-channels are commonly exploited to compromise modern communication devices [31] and have also been used for program flow tracking [32].

In the profiling phase, we assume the attacker has access to a cloned device running a cryptographic algorithm, though its encryption mode may differ from that of the target device. Typically deployed on an embedded platform, this cloned device can be queried with known or chosen parameters (e.g.,

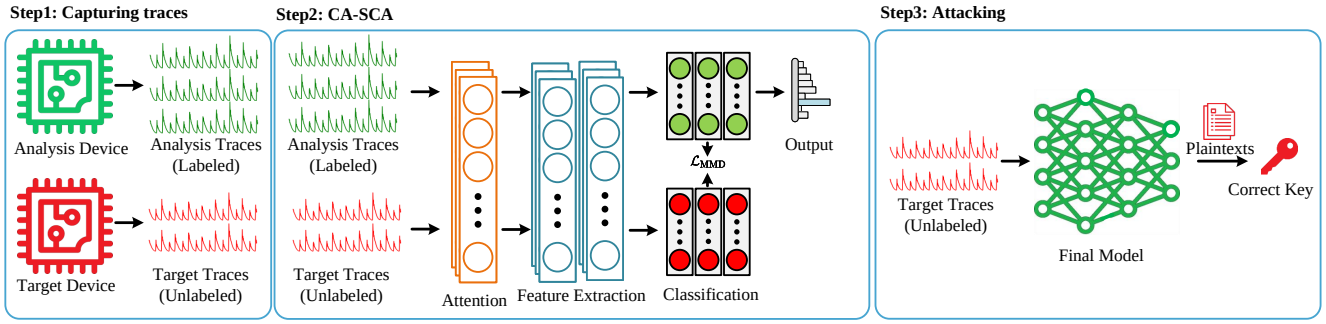


Fig. 1: Overall architecture of CA-SCA.

key, plaintext), allowing the attacker to record the resulting physical leakage (e.g., power or EM). Additionally, the attacker can acquire a limited amount of unlabeled leakage data from the target encryption mode, which is incorporated during model training to facilitate domain adaptation and address distributional differences between encryption modes.

In the attack phase, the attacker applies the trained leakage model and combines it with side-channel leakage traces from the target device to recover the encryption key. The domain adaptation technique allows the model to adapt to different encryption modes, enabling effective SCA even without labeled data from the target mode during training.

B. Attention Layer

Given an input sequence $X = [x_0, \ldots, x_{n-1}] \in \mathbb{R}^{n \times d}$, the attention layer computes an output sequence $\hat{V} = [\hat{v}_0, \ldots, \hat{v}_{n-1}] \in \mathbb{R}^{n \times d_v}$ as follows:

$$\hat{v}_i = \sum_{j=0}^{n-1} \text{softmax}\left(\frac{q_i^T k_j}{\sqrt{d}}\right) v_j = \sum_{j=0}^{n-1} \frac{\exp\left(\frac{q_i^T k_j}{\sqrt{d}}\right)}{\sum_{l=0}^{n-1} \exp\left(\frac{q_i^T k_l}{\sqrt{d}}\right)} v_j, \tag{4}$$

where

$$q_i = W_q x_i, \quad k_i = W_k x_i, \quad v_i = W_v x_i,$$

for $i = 0, \ldots, n-1$, with $W_q, W_k \in \mathbb{R}^{d_k \times d}$ and $W_v \in \mathbb{R}^{d_v \times d}$. The final output of the attention layer is given by:

$$\hat{Y} = \hat{V} W_o^T + X,$$

where $W_o \in \mathbb{R}^{d \times d_v}$ is the projection matrix.

Multi-Head Attention: A single attention operation using parameters W_q, W_k, W_v is referred to as an attention head. In practice, multiple attention heads are employed in parallel, forming an H-head attention layer as follows:

$$\begin{aligned} \hat{V}^{(i)} &= f_A(X; W_q^{(i)}, W_k^{(i)}, W_v^{(i)}), \quad i = 0, \ldots, H-1, \\ \hat{V} &= \text{concat}(\hat{V}^{(0)}, \ldots, \hat{V}^{(H-1)}), \\ \hat{Y} &= \hat{V} W_o^T + X, \end{aligned} \tag{5}$$

where concat denotes row-wise concatenation of the output matrices. This enables multi-head attention to capture information from different subspaces.

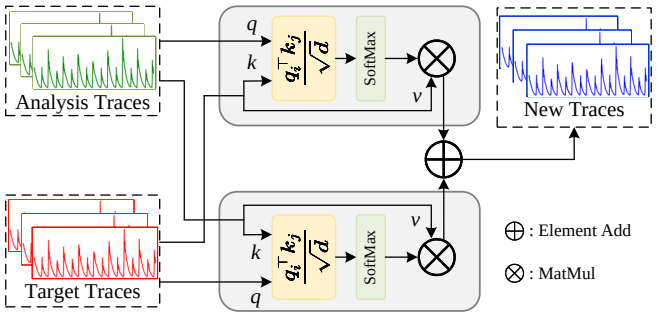


Fig. 2: Flowchart of the cross-attention mechanism.

Algorithm 1 CA-SCA Optimization Procedure

Input: Labeled analysis dataset $X_s = \{(x_i^s, y_i^s)\}_{i=1}^{n_s}$, Unlabeled target dataset $X_t = \{x_i^t\}_{i=1}^{n_t}$
Output: Optimized CNN model M_{best}
Split X_s into training set $X_{s,\text{train}}$ and validation set $X_{s,\text{val}}$
Initialize CNN model M with attention layer, feature extraction layer, and classifier parameters
Set initial best validation loss $val_loss_{\text{best}} \leftarrow \infty$
while not M.convergence **do**
    Forward pass on $X_{s,\text{train}}$ and X_t using M
    Compute total loss: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{classification}} + \lambda \cdot \mathcal{L}_{\text{MMD}}$
    Update M parameters via backpropagation to minimize $\mathcal{L}_{\text{total}}$
    Evaluate $val_loss = \text{Evaluate}(M, X_{s,\text{val}}, X_t)$
    if $val_loss < val_loss_{\text{best}}$ **then**
        $val_loss_{\text{best}} \leftarrow val_loss$
        $M_{\text{best}} \leftarrow M$
    end if
end while
return M_{best}

C. Tailoring CA-SCA for SCA

To implement UDA with an attention layer, we modified the model architecture to simultaneously optimize both classification loss and MMD loss (as shown in Fig. 1). This architecture enhances feature representation of both analysis and target data through an interactive attention mechanism, facilitating effective generalized learning.

The training process begins with a classic CNN stage, where

the parameters θ of the classifier f_θ are initialized, and the model parameters are optimized by minimizing the cross-entropy loss $\mathcal{L}_{\text{classification}}$ on the analysis dataset:

$$\mathcal{L}_{\text{classification}}(f_\theta) = \sum_{x_s^i, y_s^i \in X s} l(f_\theta(x), y). \tag{6}$$

Next, the target data undergoes feature extraction and, together with the analysis data, is fed into the attention layer. The attention layer computes a correlation matrix between the data, guiding the network to focus on key features, as shown in Fig. 2.

Throughout the training process (Algorithm 1), the total loss function $\mathcal{L}_{\text{total}}$ combines classification and MMD losses: the classification loss is calculated using labeled analysis data, while the MMD loss is based on features from both analysis and target data. Both losses jointly update all trainable model parameters, promoting effective generalization to the target data:

$$\theta' \leftarrow \theta - \alpha \frac{\partial \mathcal{L}_{\text{total}}}{\partial \theta}, \tag{7}$$

where α is the learning rate.

IV. Experiments

A. Dataset

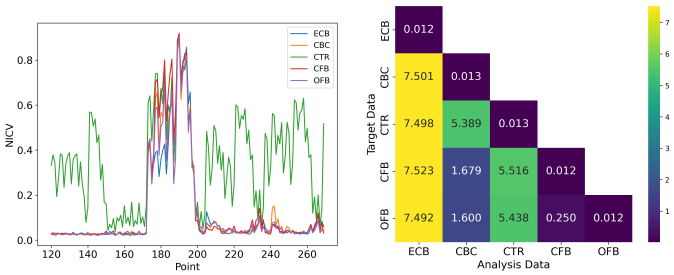
AES supports multiple encryption modes—ECB, CBC, CTR, CFB, and OFB—each influencing data processing and security differently. Although based on the same core algorithm, operational differences result in varying resistance to SCA, highlighting the need for cross-mode evaluation in comprehensive security analysis.

To evaluate generalization across modes, traces were collected from the CW308T-STM32F3 target board for each AES mode. Following the methodology in [33], Points of Interest (POIs) from the final encryption round were selected due to their favorable trace characteristics. Each dataset comprises 25,000 traces with 500 sampling points, and specific leakage models capturing key encryption operations are summarized in Table I.

To further analyze mode differences, we applied Normalized Inter-Class Variance (NICV [34]) to detect physical leakage variations. As shown in Fig. 3(a), leakage quality and timing vary across modes (e.g., CTR mode shows a time shift). Maximum Mean Discrepancy (MMD) was also used to quantify numerical differences, as shown in Fig. 3(b). Diagonal values indicate same-mode comparisons with minimal MMD values, while cross-mode comparisons yield higher values. Notably, CFB and OFB exhibit smaller MMD values due to structural similarities, while larger values highlight greater challenges in achieving generalization across different modes.

B. Setups

In the profiling phase, 20,000 traces are used, with 80% allocated for training and 20% for validation, while 500 unlabeled attack traces from the target dataset are reserved for model constraints. During the attack phase, 5,000 attack traces are used for evaluation. The metric $\bar{N}_{\text{GE}}$ is defined as the number of attack traces required for GE to reach rank 0. To estimate $\bar{N}_{\text{GE}}$, 1,000 attack traces are randomly selected, tested 200 times, and the average result is reported.



(a) Normalized Inter-Class Variance. (b) Maximum Mean Discrepancy.

Fig. 3: Analysis of AES mode variations using NICV and MMD metrics.

TABLE I: Leakage modeling for various encryption modes, where p_t denotes the plaintext, c_t denotes the ciphertext, SBox^{-1} represents the inverse SBox operation, InvSR denotes the inverse shift rows operation, and $k^*_{R^{10}}$ denotes the expanded key for round 10. All models utilize the ID leakage model.

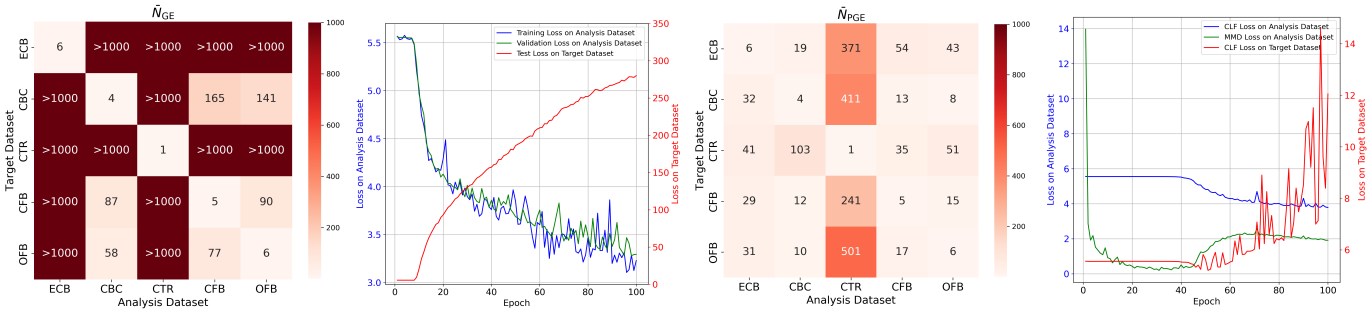
Leakage model	Encryption mode
$\text{InvSR}(c_t) \oplus (\text{SBox}^{-1}[c_t \oplus k^*_{R^{10}}])$	ECB
$\text{InvSR}(c_t) \oplus (\text{SBox}^{-1}[c_t \oplus k^*_{R^{10}}])$	CBC
$\text{InvSR}(c_t) \oplus ((\text{SBox}^{-1}[(c_t \oplus p_t) \oplus k^*_{R^{10}}]))$	CFB
$\text{InvSR}(c_t \oplus p_t) \oplus ((\text{SBox}^{-1}[(c_t \oplus p_t) \oplus k^*_{R^{10}}]))$	OFB
$\text{InvSR}(c_t \oplus p_t) \oplus ((\text{SBox}^{-1}[(c_t \oplus p_t) \oplus k^*_{R^{10}}]))$	CTR

Our focus is on assessing CA-SCA performance across different AES modes rather than fine-tuning network architecture. Accordingly, we use a simple CNN architecture with three convolutional layers and three fully connected layers. The attention layer has 8 heads with an embedding dimension of 64. The convolutional layers have filter sizes of 16, 32, and 64, and kernel sizes of 1, 50, and 3, respectively. Each fully connected layer consists of 20 neurons. *SELU* is used as the activation function for the input and hidden layers, and *softmax* for the output layer. We set the batch size to 100, epochs to 100, and the hyperparameter λ to 0.02, using the *Adam* optimizer with a learning rate of 0.001.

All experiments are conducted with PyTorch 1.8.0 and Python 3.8.0 on hardware with an Intel® Xeon® Platinum 8358P CPU @ 2.60 GHz and an NVIDIA GeForce GTX 4090 24GB GPU.

C. Results

To facilitate comparative analysis, we first evaluated the performance of DL-PA [10] across different AES modes. The attack results are illustrated in Fig. 4(a), where the x-axis represents the analysis dataset and the y-axis represents the target dataset. The CNN model is trained on the x-axis dataset and attempts to recover the key for the target dataset. As shown, the same mode (along the diagonal) can recover the



(a) $\bar{N}_{GE}$ (DL-PA) (b) Learning curves (DL-PA) (c) $\bar{N}_{GE}$ (CA-SCA) (d) Learning curves (CA-SCA)

Fig. 4: Analysis results across different AES modes.

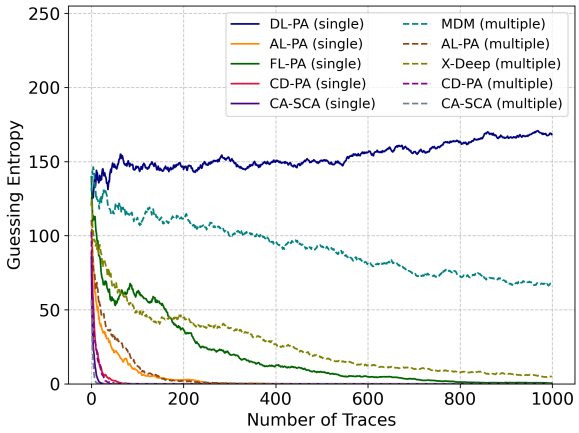


Fig. 5: Comparison with other methods. For the single-analysis dataset, training was performed on the ECB dataset and testing was performed on the OFB dataset. For multi-analysis datasets, training was performed on the ECB and CTR datasets and testing on the OFB dataset.

correct key within 10 attack traces. However, variations in AES modes may lead to reduced DL-PA performance or even failure. For instance, when the model is trained on analysis data in ECB mode, it fails to recover the key for the target mode, even after 1,000 attack traces.

To better characterize DL-PA's performance across AES modes, we present the learning curves for ECB and CBC modes in Fig. 4(b). Both training and validation losses decrease over time and stabilize around 3.2, while the testing loss on the target dataset continues to increase. This suggests that the discrepancy between target and analysis datasets is too large, leading to model failure through misclassification. In DL contexts, this is typically attributed to overfitting. However, in practical SCA, overfitting is difficult to detect without access to labeled attack traces, which prevents calculating test losses.

Fig. 4(c) summarizes CA-SCA's performance across AES modes, showcasing improved results in all cases, with some attacks nearly matching the performance achieved in identical mode scenarios. Notably, when CTR mode is used as the analysis dataset, its data-independent nature results in more pronounced leakage features. While the CNN model effectively learns these features, it struggles to generalize to other AES modes. The learning curve of CA-SCA for the ECB-CBC task, shown in Fig. 4(d), reveals that during the initial 40 epochs, the MMD loss decreases while the classification loss remains stable. Subsequently, the training loss decreases and stabilizes, but the testing loss initially decreases before rising again. This behavior suggests that the model first aligns the analysis and target datasets in high-dimensional space through attention mechanisms and MMD loss before proceeding with classification. Despite substantial differences between the datasets leading to overfitting on the analysis dataset, the intermediate model continues to perform effectively on the target data. Furthermore, as illustrated in Fig. 4(d), the results surpass those in Fig. 4(b), highlighting CA-SCA's exceptional capability to handle data variations and improve generalization across different modes.

We then compare CA-SCA with other state-of-the-art techniques. MDM [21] and X-Deep [19] are widely used for addressing portability issues, with model robustness generally increasing with the number of analytical datasets. However, MDM and X-Deep have limitations when there are substantial differences between target and analysis datasets. For example, as shown in Fig. 3(a) and Fig. 3(b), CBC, CFB, and OFB datasets are relatively similar, but differ more significantly from other datasets. To test this, we trained the model on ECB and CTR datasets and evaluated it on the OFB dataset. In Fig. 5, both MDM and X-Deep fail to recover the correct key for the target dataset. Unlike methods like AP-PA [22], FL-PA [35], and CD-PA [23], CA-SCA requires no fine-tuning or extensive prior knowledge for SCA parameter tuning. Table II provides a detailed performance comparison of the tested methods. In summary, CA-SCA requires no specific tuning knowledge, does not rely on multiple analysis datasets, and outperforms current state-of-the-art methods in certain cases.

V. Discussions

A. Impact of Parameter λ

The parameter λ in Equation 3 is critical in Unsupervised Domain Adaptation (UDA). If λ is set too small, the network focuses primarily on optimizing $\mathcal{L}_{\text{classification}}$, reducing the

TABLE II: Detailed performance compared to related works.

Scenario	Method	Fine-tune	$\bar{N}_{\text{GE}}$
Single Profiling Dataset	DL-PA [10]	✗	-
	AL-PA [22]	✓	493
	FL-PA [35]	✗	-
	CD-PA [23]	✓	110
	This work	✗	**31**
Multiple Profiling Dataset	MDM [21]	✗	-
	AL-PA [22]	✓	361
	X-Deep [19]	✗	-
	CD-PA [23]	✓	74
	This work	✗	**21**

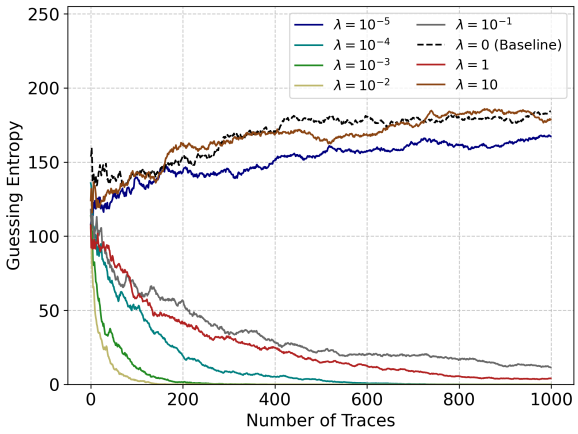


Fig. 6: Attack results of different λ on the ECB-CBC task, with $\lambda \in \{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$.

influence of $\mathcal{L}_{\text{MMD}}$ on the learning process. Conversely, setting λ too large may excessively impact the learning process, potentially leading to classification failure. To assess the sensitivity of λ, we illustrate its effect on $\bar{N}_{\text{GE}}$ in Fig. 6.

Experimental results indicate that setting λ too small (e.g., $\lambda = 10^{-5}$) or too large (e.g., $\lambda = 10$) fails to enhance attack performance, consistent with our analysis that $\lambda = 10^{-3}$ is an effective choice. While various values of λ improve performance, a practical strategy is to start with smaller values, particularly in scenarios with low signal-to-noise ratios, as this encourages the network to focus on challenging classification tasks. If the reduction in $\mathcal{L}_{\text{MMD}}$ is insufficient or progresses too slowly, λ can be incrementally increased to accelerate the UDA process.

B. Impact of the Number of Attention Heads

In prior experiments, we observed that setting the number of attention heads, denoted by H, to the default value of 8 provided strong performance. To evaluate CA-SCA's sensitivity to different H values, we tested the ECB-CBC task with varying H while keeping other hyperparameters constant. For each H value, three independent networks were trained, and the $\bar{N}_{\text{GE}}$ results are summarized in Table III.

As shown in Table III, CA-SCA demonstrates notable instability for smaller H values (e.g., 2 and 4). Specifically,

TABLE III: CA-SCA attack results for different values of H on the ECB-CBC task, with T_1, T_2, and T_3 representing the $\bar{N}_{\text{GE}}$ results from three independent tests.

H	ECB-CBC		
	T_1	T_2	T_3
2	984	$>1K$	580
4	282	133	184
8	37	30	29
16	41	33	47
32	47	52	44

CA-SCA required more than 580 and 133 attack traces, respectively, to reach a $\bar{N}_{\text{GE}}$ of 0, whereas $H = 8$ achieved this with fewer than 40 attack traces across all trials. This instability for smaller H values is likely due to limited information flow to distant sample points. In contrast, CA-SCA performs well with larger H values, such as 8 and 16, though performance slightly declines at $H = 32$, possibly due to overfitting caused by the increased parameter count in the attention layer. These findings suggest an optimal range for H in CA-SCA, with values between 8 and 16 consistently delivering robust and reliable performance across the tests.

VI. Conclusions and Future Work

In this paper, we introduce CA-SCA, a novel approach to address portability challenges across different AES modes. CA-SCA leverages cross-attention mechanisms and UDA to align input samples within a high-dimensional feature space, effectively reducing interference from encryption mode mismatches. Notably, CA-SCA does not require specific fine-tuning or multiple analysis datasets. Our evaluation on five distinct AES mode datasets shows that, while traditional SCA methods often struggle with substantial mode differences, CA-SCA enables previously infeasible attacks and outperforms state-of-the-art techniques.

In future work, we aim to further explore the impact of cross-attention mechanisms and UDA on SCA. Additionally, we plan to assess CA-SCA's performance across various hardware platforms to advance portability research within the SCA community.

VII. Acknowledgements

The work is supported by the National Natural Science Foundation of China (No. U21A20465), the National Key R&D Program of China (No. 2023YFB2703700), and the "Leading Goose" R&D Program of Zhejiang (No. 2025C02033). Jian Shen is the corresponding author, and Haoyu Ma is the co-corresponding author. The authors thank the anonymous reviewers and the shepherd for their valuable suggestions on how to improve this paper.

References

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 1996, pp. 104–113.

[2] B. Gras, C. Giuffrida, M. Kurth, H. Bos, and K. Razavi, "Absynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures." in *NDSS*, 2020.

[3] Y. Gao, H. Zhang, W. Cheng, Y. Zhou, and Y. Cao, "Electro-magnetic analysis of gpu-based AES implementation," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[4] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina, "Sok: Deep learning-based physical side-channel analysis," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–35, 2023.

[5] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic hardware and embedded systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*. Springer, 2003, pp. 13–28.

[6] L. Wu, G. Perin, and S. Picek, "The best of two worlds: Deep learning-assisted template attack," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 3, pp. 413–437, 2022.

[7] S.-C. You, M. G. Kuhn, S. Sarkar, and F. Hao, "Low trace-count template attacks on 32-bit implementations of ascon aead," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 4, pp. 344–366, 2023.

[8] A. Heuser and M. Zohner, "Intelligent machine homicide: Breaking cryptographic devices using support vector machines," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2012, pp. 249–264.

[9] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert, "Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis)," in *Constructive Side-Channel Analysis and Secure Design: 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers 6*. Springer, 2015, pp. 20–33.

[10] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*. Springer, 2016, pp. 3–26.

[11] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Methodology for efficient cnn architectures in profiling attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–36, 2020.

[12] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 148–179, 2019.

[13] F. Hu, J. Shen, and P. Vijayakumar, "Side-channel attacks based on multi-loss regularized denoising autoencoder," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2051–2065, 2023.

[14] L. Wu, L. Weissbart, M. Krček, H. Li, G. Perin, L. Batina, and S. Picek, "Label correlation in deep learning-based side-channel analysis," *Ieee transactions on information forensics and security*, vol. 18, pp. 3849–3861, 2023.

[15] S. Hajra, S. Chowdhury, and D. Mukhopadhyay, "Estranet: An efficient shift-invariant transformer network for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, pp. 336–374, 2024.

[16] L. Wu, G. Perin, and S. Picek, "I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[17] H. Wang, M. Brisfors, S. Forsmark, and E. Dubrova, "How diversity affects deep-learning side-channel attacks," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. IEEE, 2019, pp. 1–7.

[18] M. G. Perusheska, H. M. Trpceska, and V. Dimitrova, "Deep learning-based cryptanalysis of different aes modes of operation," in *Future of Information and Communication Conference*. Springer, 2022, pp. 675–693.

[19] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-deepsca: Cross-device deep learning side channel attack," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[20] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury, "Practical approaches toward deep-learning-based cross-device power side-channel attack," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2720–2733, 2019.

[21] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek, and R. Ranjan, "Mind the portability: A warriors guide through realistic profiled side-channel analysis," in *NDSS 2020-Network and Distributed System Security Symposium*, 2020, pp. 1–14.

[22] P. Cao, H. Zhang, D. Gu, Y. Lu, and Y. Yuan, "Al-pa: cross-device profiled side-channel attack using adversarial learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 691–696.

[23] P. Cao, C. Zhang, X. Lu, and D. Gu, "Cross-device profiled side-channel attack with unsupervised domain adaptation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 27–56, 2021.

[24] H. Yu, H. Shan, M. Panoff, and Y. Jin, "Cross-device profiled side-channel attacks using meta-transfer learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 703–708.

[25] R. G. Praveen and J. Alam, "Recursive joint cross-modal attention for multimodal fusion in dimensional emotion recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4803–4813.

[26] Y. Cai and Y. Yuan, "Car-transformer: Cross-attention reinforcement transformer for cross-lingual summarization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 718–17 726.

[27] J. Song, C. Mou, S. Wang, S. Ma, and J. Zhang, "Optimization-inspired cross-attention transformer for compressive sensing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 6174–6184.

[28] Y. Liu, Y. Chen, M. Gou, C.-T. Huang, Y. Wang, W. Dai, and H. Xiong, "Towards unsupervised domain generalization for face anti-spoofing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 20 654–20 664.

[29] H. Zhang, S. Li, P. Wang, D. Zeng, and S. Ge, "M3d: Dataset condensation by minimizing maximum mean discrepancy," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 8, 2024, pp. 9314–9322.

[30] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*. Springer, 2009, pp. 443–461.

[31] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, "Screaming channels: When electromagnetic side channels meet radio transceivers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 163–177.

[32] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1095–1108.

[33] D. Jayasinghe, R. Ragel, J. A. Ambrose, A. Ignjatovic, and S. Parameswaran, "Advanced modes in aes: Are they safe from power analysis based side channel attacks?" in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE, 2014, pp. 173–180.

[34] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, "Nicv: normalized inter-class variance for detection of side-channel leakage," in *2014 International Symposium on Electromagnetic Compatibility, Tokyo*. IEEE, 2014, pp. 310–313.

[35] F. Zhang, B. Shao, G. Xu, B. Yang, Z. Yang, Z. Qin, and K. Ren, "From homogeneous to heterogeneous: Leveraging deep learning based power analysis across devices," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

AttenPIM: Accelerating LLM Attention with Dual-mode GEMV in Processing-in-Memory

Liyan Chen†, Dongxu Lyu†, Zhenyu Li, Jianfei Jiang, Qin Wang, Zhigang Mao, Naifeng Jing*
Department of Micro/Nano Electronics, Shanghai Jiao Tong University, Shanghai, China
{liyan.chen, sjtuj}@sjtu.edu.cn

***Abstract*—Large Language Models (LLMs) have demonstrated unprecedented generative performance across a wide range of applications. While recent heterogeneous architectures attempt to address the memory-bound bottleneck from attention computations by processing-in-memory (PIM) offloading, they overlook two critical characteristics of attention GEMVs that distinguish them from traditional PIM scenarios: (1) dynamic matrix dimensions that scale with token length, and (2) distinct GEMV patterns between score computation ($Q \times K_t$) and context computation ($S \times V$). Existing PIM designs, employing either uniform or transposed computing modes, suffer from inefficiencies in newly generated element preparation or distinct GEMV execution. To address these limitations, we propose AttenPIM, a software-hardware co-design for efficient PIM-based attention acceleration. For bank-level execution, we propose dual-mode computing modes tailored for score and context computations with PIM-oriented data layouts and execution flows for KV storage, supported by a low-cost configurable per-bank PIM unit (PU). For system-level execution, we leverage token-level and head-level concurrency to ensure workload balance and maximize bank PU parallelism. Furthermore, dynamic allocation and kernel fusion methods are proposed to further minimize memory overhead. Experimental results demonstrate that AttenPIM achieves $1.13\times$–$5.26\times$ speedup and reduces energy consumption by 17%–49% compared to two state-of-the-art PIM baselines.**

I. Introduction

Large Language Models (LLMs) have been widely deployed across diverse applications [1], such as language understanding, content generation, and decision support. LLM inference demands immense amounts of compute and memory resources. While machine learning accelerators (e.g., GPU and TPU, referred to as xPU) demonstrate high performance on compute-bound layers, they struggle with memory-bound layers due to limited memory bandwidth and resulting underutilization [2]–[4]. For instance, with a batch size of 16 and a token length of $2K$, the memory-bound multi-head attention (hereafter attention) layers of GPT-3 13B consume more than 50% of the total time on A100 GPUs [2]. This proportion increases dramatically with larger batch sizes or longer sequence lengths, severely limiting the inference performance [4].

The attention layer comprises three operations: score, softmax, and context, with score and context computations being the most time-consuming matrix-vector multiplication (GEMV) computations. Recent studies [5]–[8] offload attention operations to processing-in-memory (PIM) designs, which have demonstrated significant performance improvements in GEMV computation [9], [10]. By integrating PIM units (PUs) within memory devices, PIM can achieve higher aggregated bandwidth and computational parallelism [11]–[14].

However, attention GEMVs in LLMs exhibit distinct characteristics that differentiate them from conventional PIM applications with static weight data and uniform execution flow [9], [10]. First, the dimensions of attention GEMV matrices dynamically expand with increasing token length, requiring additional matrix data preparation for each new token. Second, attention score and context computations, with transposed matrix dimensions, perform multiply-accumulate (MAC) operations along different dimensions during execution.

Existing PIM works fail to consider these characteristics with the PIM computing paradigm simultaneously, thus leading to performance degradation. NeuPIMs [6], built based on Newton [9], partitions both matrices column-wise while broadcasting vector data for parallel execution. However, the newly generated matrix elements cannot fully utilize DRAM burst transfer (16 bits out of 256/512 bits), incurring $50\times$ preparation time with masked accesses [7]. While AttAcc [5] proposes a multi-level partition method for two matrices in a transposed manner, it requires PUs at multiple memory levels for partial result accumulation. Additionally, the inter-PU communication would congest memory buses (particularly TSVs), leading to 15%–58% performance degradation.

We identify that it is crucial to simultaneously address both attention GEMV characteristics and PIM computing paradigm design. To achieve this goal, we propose AttenPIM, a software-hardware co-design for efficient PIM-based attention acceleration. Our contributions are as follows:

- For bank-level execution, we propose dual computing modes for score and context computations with PIM-oriented data layouts and execution flow, supported by a low-cost configurable per-bank PIM unit (PU).
- For system-level execution, we leverage token-level and head-level concurrency to ensure workload balance and maximize bank PU parallelism. Furthermore, we introduce dynamic allocation and kernel fusion methods to reduce memory space waste and memory accesses.
- We implement our design on TSMC 28nm technology and extensively evaluate it on real-world LLM datasets. AttenPIM achieves a $1.13\times$–$5.26\times$ speedup and reduces energy consumption by 17%–49% compared to two state-of-the-art PIM baselines.

This work was supported by National Natural Science Foundation of China (Grant U24B20165 and No.92264108).

† Co-first Author. *Corresponding Author.

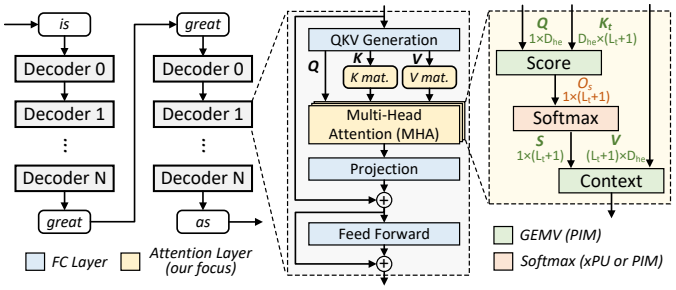


Fig. 1. Model architecture and inference (Gen stages) in LLMs.

II. Background and Motivation

A. LLM Model

LLM models are built from the Transformer decoder [15], as illustrated in Fig. 1. Each decoder consists of fully-connected layers (QKV Generation, FFN), attention layers (MHA). The inference process consists of an initial Sum stage followed by recurring Gen stages. The Sum stage computes the initial KV cache (matrices) from the input prompt and generates the first token while each Gen stage updates KV matrices by concatenating and generates a new token auto-regressively.

The MHA layer consists of multiple independent heads, each performing two types of GEMVs: score computation ($Q{\times}K_t$) and context computation ($S{\times}V$), and a softmax operation. Given head embedding dimension D_{he} and current sequence length L_t, the score computation involves a $1{\times}D_{he}$ vector Q and a $D_{he}{\times}(L_t{+}1)$ matrix K_t, while the context computation involves a $1{\times}(L_t{+}1)$ vector S and a $(L_t{+}1){\times}D_{he}$ matrix V. The MHA layer with low compute-memory ratio is underutilized in xPU. For instance, with GPT-3 13B under a batch size of 16 and token length of $2K$, it can consume more than 50% of the total time on A100 GPUs [4].

B. HBM-based PIM for Attention

An HBM device [16] integrates multiple DRAM dies with a buffer die through TSVs, as shown in Fig. 2(a). Each DRAM die comprises independent (pseudo) channels, with each channel subdivided into multiple bank groups (BGs) consisting of four banks. HBM-based PIM, which incorporates PUs at various memory hierarchies (die, BG, bank), offers promising solutions for GEMV through high aggregated bandwidth.

Recent studies have explored offloading attention GEMVs to HBM-based PIM for acceleration [5], [6]. NeuPIMs [6] and works [7], [8], depicted in Fig. 2(b), partitions both matrices K_t and V column-wise. They leverage a global shared buffer to broadcast vector data while confining GEMV execution within bank PUs. AttAcc [5], as shown in Fig. 2(c), partitions matrix K_t token-wise across BGs, with each matrix tile further subdivided embedding-wise within each BG, and leverages a transposed layout for matrix V. Partial results of $Q{\times}K_t$ in bank PUs are accumulated in BG PUs, while partial results of $S{\times}V$ are accumulated in die PU.

Considering that softmax operation execution time is typically masked through head-level pipelining [5], [6], this paper primarily concentrates on attention GEMVs. Notably,

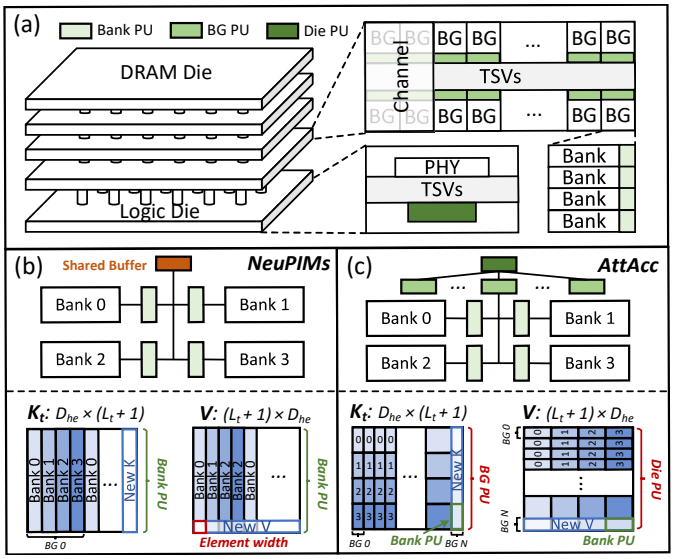


Fig. 2. (a) HBM-PIM. (b) NeuPIMs architecture. (c) AttAcc architecture.

AttenPIM design can also extend to PIM designs based on other memory devices, such as GDDR [10], DIMM [17].

C. Motivation

Attention GEMVs in LLMs exhibit unique characteristics that distinguish them from conventional PIM applications [9], [10]. First, attention matrices expand with token length, introducing an additional matrix preparation phase. Second, attention GEMVs encompass score and context computations with transposed matrix layouts, which perform MAC operations along different dimensions during execution. The efficiency of both phases is fundamentally determined by matrix data layout. While existing PIM designs adopt either uniform or transposed partitioning strategies to distribute matrices K and V, these one-size-fits-all approaches demonstrate inefficiencies in either the preparation or computation phase.

NeuPIMs distributes newly generated V head elements across different banks, where the element width (e.g., 16 bits) underutilizes burst transfer capabilities (e.g., 256/512 bits). The masked accesses for V preparation consume about $50{\times}$ more time compared with preparation for matrix K_t, accounting for 12%-77% of total execution time across various token lengths. AttAcc introduces contention on shared memory buses between PUs. Specifically, TSV bandwidth emerges as a bottleneck for context computation ($S{\times}V$), resulting in 15%-58% performance degradation. Additionally, the requirement for diverse vector data across bank PUs necessitates distributed input buffers in each bank PU. These performance penalties accumulate during token generation, substantially impacting end-to-end inference performance.

Motivated by these observations, we identify that there is an urgent need for specialized data layouts and compatible execution flows tailored separately for PIM-based attention score computation ($Q{\times}K_t$) and context computation ($S{\times}V$), which can maintain burst transfer efficiency and eliminate inter-PU communication overhead.

III. AttenPIM Architecture

AttenPIM implements a multi-level acceleration strategy for LLM attention, spanning from bank-level optimizations to system-level enhancements. At the bank level, we introduce dual computing modes for score and context computations with PIM-oriented data layouts and execution flows. We confine GEMV execution within bank PUs while leveraging broadcast mechanisms to eliminate distributed buffers. At the system level, we exploit token-level parallelism and head-level parallelism to ensure workload balance, complemented by kernel fusion and dynamic allocation methods to minimize memory accesses and memory waste. Given the independent nature of attention head computations, we use a single head as a representative example, unless stated otherwise.

A. Computing Mode: Score

Building upon the demonstrated efficiency of NeuPIMs in score computation ($Q \times K_t$) with bank PUs, we follow their proposed data layout and execution flow. We present a brief overview using a two-bank configuration as an illustrative example, while not claiming this as our innovation.

1) Data Layout: As illustrated in Fig. 3(a), when a new K_t column is generated, all head elements are mapped to a designated bank and stored continuously within a bank row. In this case, score computation ($Q \times K_t$) can be performed within bank PU with high DRAM operation efficiency during both the storage and execution stage.

2) Execution Flow: Inner-product is leveraged for score computation ($Q \times K_t$). As depicted in Fig. 3(c), head elements from two tokens are positioned in identical row and column locations across different banks to enable parallel read operations. In each phase with burst-width transfer, bank PUs retrieve head elements from DRAM arrays (❶), while the shared buffer broadcasts corresponding vector data of Q to all bank PUs (❷). These elements undergo multiplication and accumulation to generate a partial output element (❸). The output elements from different phases are accumulated to generate the final result (❹).

B. Computing Mode: Context

AttenPIM introduces a specialized data layout for V matrix storage and implements a compatible execution flow for context computation ($S \times V$).

1) Data Layout: We identify burst length as the minimum storage granularity that maintains DRAM operation efficiency, which we leverage for partitioning each newly generated vector V. Consider the burst transfer length of N_{bl} and model element precision of N_{pre}. As shown in Fig. 3(b), a token head is divided into N_{BB} burst blocks, where $N_{BB} = D_{he}/N_{bl}$. Each burst block consists of N_{bl}/N_{pre} elements. The mapping of these burst blocks follows two key principles: (1) burst blocks with identical indices across different tokens reside in the same bank (e.g., BB_1 of all tokens are stored continuously in bank 1), and (2) data alignment in identical rows and columns is maintained to enable parallel read operations between banks.

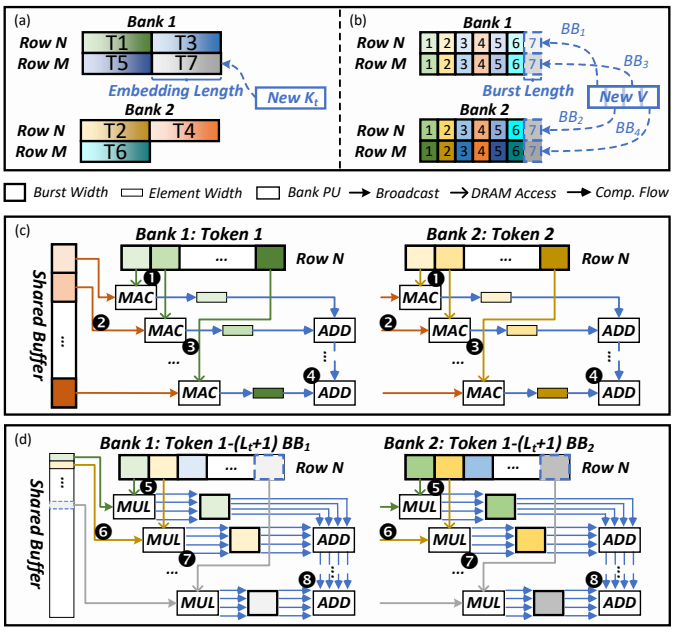


Fig. 3. (a) Data layout for K matrix. (b) Data layout for V matrix. (c) Execution flow for score. (d) Execution flow for context.

2) Execution Flow: The data layout ensures that elements requiring multiplication and addition operations are localized within the same bank. Based on it, we propose to perform context computation ($S \times V$) using an outer product approach. As depicted in Fig. 3(d), burst blocks BB_1 and BB_2 of N_t token are allocated to bank 1 and bank 2, respectively. During each phase, bank PUs read a burst block of a token from DRAM arrays (❺), while the shared buffer only needs to broadcast a single element of vector S to bank PUs (❻). This element is shared and multiplied with each head element of the burst block instead of MAC, generating N_{bl}/N_{pre} partial outputs (❼). These partial results undergo element-wise accumulation across phases to generate the final results (❽).

C. Hardware Support: Configurable PU

To efficiently support dual-mode GEMV operations, we propose a cost-effective configurable PIM unit at the bank level, complemented by a global shared buffer for data broadcasting. To match the burst transfer bandwidth, each PU incorporates N_{bl}/N_{pre} multipliers and adders, along with multiple buffers for pipelined execution.

1) Inner product: The inner product computation follows an adder tree architecture, as illustrated in Fig. 4(a). Using a configuration with four multipliers, four adders, and a result buffer as an example: during each phase, the multipliers receive and execute burst-length inputs of N_{bl}/N_{pre} elements from both the shared buffer and DRAM row buffer. The adder tree performs accumulation of outputs from multipliers and accumulation of partial outputs between phases. Consequently, $D_{he} \times N_{pre}/N_{bl}$ phases are needed to complete one head GEMV. The host can either read the result buffer for softmax operation (kernel fusion detailed in Section III-E2) or write it back to banks for subsequent computations.

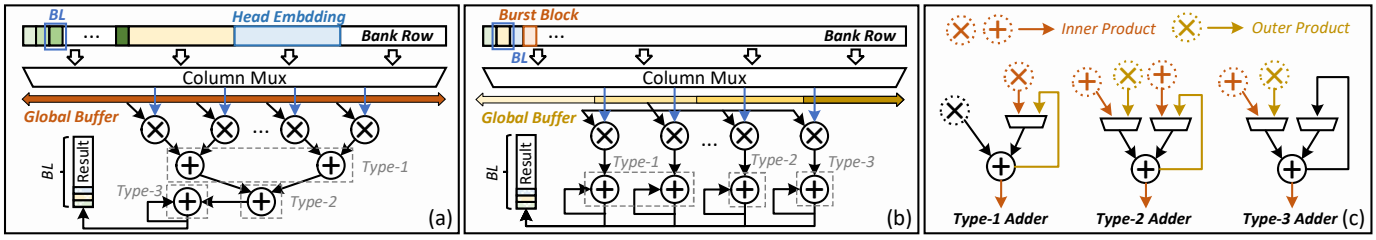


Fig. 4. (a) Inner product mode for score. (b) Outer product mode for context. (c) Three types of adders. Buffers for pipeline are not shown.

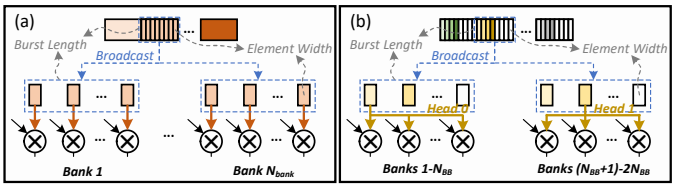


Fig. 5. (a) Broadcast in score execution. (b) Broadcast in context execution.

2) Outer product: The outer product implementation employs a self-accumulation approach, where outputs accumulate within individual multiplier-adder sets without inter-set data transfer. As shown in Fig. 4(b), in each phase, a single input element from the shared buffer is shared by all multipliers to be multiplied with data from DRAM row buffer. Each set's adder combines the new multiplier output with the previously accumulated sum. In this case, N_{bl}/D_{he} outputs are generated concurrently through self-accumulating. L_t+1 phases are needed to generate final results.

3) Configurable Path: The Bank PU implements three types of adders to support both inner and outer product operations, as depicted in Fig. 4(c). First, the type-1 adder combines input from a multiplier with either another multiplier's output (inner product) or its own previous output (outer product). Second, the type-2 adder processes inputs from two other adders (inner product) or combines a multiplier output with its own previous output (outer product). Third, the type-3 adder adds its previous output with another adder's output (inner product) or a multiplier's output (outer product). The mode configuration is performed only once for each head of the entire batch to amortize overhead.

D. Workload Balance

1) Score $Q\times K_t$: AttenPIM adopts the mapping strategy in NeuPIMs, assigning each head of K matrix to one channel and each token to one bank. In contrast, AttAcc maps a head across the entire HBM device and allocates each token to a BG for cross-level accumulation. Since the number of BGs in an HBM device significantly exceeds the number of banks in a channel, NeuPIMs and AttenPIM achieve better workload balance for requests with relatively short token lengths.

2) Context $S\times V$: The context execution of one head cannot exploit all-bank parallelism, thus leading to potential workload imbalance. Specifically, the number of burst block N_{BB} is typically smaller than the available channel banks N_{bank} (e.g., N_{BB}=8 in GPT-3 while N_{bank}>16 in a HBM device). To address this limitation, we implement concurrent context computation for multiple heads instead of single-head execution across all banks.

For head selection, we prioritize different heads from a single request over heads from different requests for two key reasons: (1) Requests with varying token lengths require different numbers of phases for context completion, potentially causing new workload imbalance. (2) Data arrangement in banks may differ across requests, preventing effective parallel read operations. Conversely, heads within the same request maintain consistent phase length and data layout. Consequently, we perform context execution for N_{bank}/N_{BB} heads in a channel at the same time.

Unlike score computation ($Q\times K_t$) where all bank PUs share the same input vector Q, context matrices V of different heads multiply with different input vectors S, challenging the broadcast mechanism. However, in our computing mode for context, only one element is required in the execution phase of N_{BB} bank PUs (one head), allowing the remaining bits of the shared buffer to be leveraged for other heads. As illustrated in Fig. 5(a), for score computation, the elements broadcast by the shared buffer are leveraged in all banks, multiplied with corresponding matrix data. In contrast, as shown in Fig. 5(b), different elements from the shared buffer are leveraged in context computations of different heads. Consequently, the shared buses can provide vector S elements to N_{bl}/N_{pre} heads' context computations concurrently, with each head utilizing N_{BB} banks, thereby ensuring optimal all-bank parallelism and workload balance.

E. Memory Optimization

1) Dynamic Allocation: Recent research has introduced dynamic KV allocation strategies (e.g., allocating memory space for several tokens when needed) to decrease memory waste from static allocation (e.g., allocating space for maximum token lengths at a time). While these strategies effectively increase batch size, they cannot be directly implemented in PIM designs due to PIM's requirement for precise data layout across columns, rows, and banks.

In AttenPIM, for score computation ($Q\times K_t$), to ensure all-bank parallelism, the same row and same columns of all banks in the channel should be allocated when existing memory space is exhausted. For context computation ($S\times V$), N_{BB} banks are employed per token head. Similarly, we allocate the same entire row of these banks as needed.

TABLE I
DYNAMIC ALLOCATION METHOD COMPARISON

	K Cache	V Cache
NeuPIMs	$N_{bank} \times N_{row}/D_{he}$	N_{row}/N_{pre}
AttAcc	$N_{ch} \times N_{bank} \times N_{row}/D_{he}$	$N_{ch} \times N_{bank} \times N_{row}/D_{he}$
AttenPIM	$N_{bank} \times N_{row}/D_{he}$	N_{row}/N_{bl}

Table I compares the number of tokens per dynamic allocation (parameters in Section IV). For K Cache allocation, NeuPIMs and AttenPIM allocate the same entire row (size of N_{row}) of all banks when needed, with an allocation size of $N_{bank} \times N_{row}/D_{he}$ tokens. Since AttAcc maps heads across the entire HBM device, it requires $N_{ch}\times$ memory space. For V Cache allocation, NeuPIMs allocates V cache of N_{row}/N_{pre} tokens at a time, which significantly wastes memory space. AttAcc leverages a transposed data layout of K Cache with the same allocation size. AttenPIM allocates entire row of D_{he}/N_{bl} banks, whith total allocation size is N_{row}/N_{bl}.

2) Kernel Fusion: NeuPIMs and AttAcc overlap softmax execution time between heads by executing GEMV operations in the PUs while performing softmax operations in softmax unit (SU, either in xPU or PIM) for different heads. To achieve this, NeuPIMs introduces double row buffers in each bank to enable concurrent access from PU and SU, while AttAcc implements direct data transfer from PU buffer to SU without interrupting PU execution. AttenPIM adopts AttAcc's approach for kernel fusion, achieving enhanced benefits due to its context operations within bank PUs, which reserves all TSV bandwidth for softmax operation transfers.

IV. EVALUATION

A. Experiment Setup

1) Baselines: We compare AttenPIM with two state-of-the-art PIM architectures: NeuPIMs [6] and AttAcc [5]. While these baselines provide comprehensive xPU-PIM heterogeneous acceleration for LLMs, AttenPIM specifically targets attention GEMV operations, which constitute the primary performance bottleneck. We leverage HBM as the memory device, with configurations detailed in Table II. To ensure a fair comparison, we maintain consistent hardware parameters across all implementations, including channel count, bank group count, bank count, and so on.

2) LLM Workloads: Our evaluation employs four variants of GPT-3 [18], the Large Language Model developed by OpenAI, as detailed in Table III. Since various LLM variants (including other architectures such as Llama [19]) differ primarily in their number of heads while maintaining consistent head dimensions, the attention performance metrics (e.g., execution time, energy consumption) scale approximately linearly with the number of heads. Consequently, we normalize the results relative to the baseline and present the averaged measurements.

For comprehensive performance assessment, we utilize four diverse real-world LLM inference datasets: Alpaca [20], ShareGPT [21], MultiNews [22], and LCC [22]. Alpaca dataset is an instruction dataset generated using OpenAI's text-davinci-003 engine, characterized by compact token lengths of less than 100 tokens on average. ShareGPT dataset is a set of conversations scraped from the real-world user logs of ChatGPT, with a maximum token length of $2K$ and average of about $0.5K$. Both MultiNews and LCC are from LongBench [22] with average token length of more than $2K$ and $4K$, respectively, which are specialized in long-context processing.

TABLE II
SIMULATOR DETAILS

HBM Parameters			
Banks/BG	4	Banks/Channel (N_{bank})	32
Burst Length (N_{bl})	32B	Row Buffer Size (N_{row})	2KB
Channels (N_{ch})	8	Capacity/Channel	1GB
Address Mapping	Row-Channel-Rank-Bank-BG-Column		
DRAM Timing Parameters	CCD_S=1:CCD_L=2:RRD=6:RCD=14:RAS=34:RP=14 CL=14:WL=4:WR=16:RTP_L=6:RTP_S=4:RFC=260		
Bank PU Parameters			
Multipliers	16	Adders	16
Frequency	666MHz	Buffer Size	32B

TABLE III
THE EVALUATED LLM CONFIGURATIONS

Model	Heads (N_h)	Embedding(N_e)	Model	Heads (N_h)	Embedding(N_e)
GPT3-7B	32	4096	GPT3-13B	40	5120
GPT3-30B	56	7168	GPT3-175B	96	12288

* Precision: $N_{pre} = 2$ (16-bit floating point); Head Embedding: $D_{he} = N_e/N_h = 128$.

3) Simulation: We develop an in-house simulator by modifying DRAMSim3 [23] to evaluate the performance of AttenPIM. We also implement the data layout and execution flow of NeuPIMs and AttAcc in this simulator for comparative analysis. Following NeuPIMs' methodology, we randomly pick sequence lengths from each dataset for simulation. This way, we can warm up the inference batch so that the batch is filled with requests having various sequence lengths.

For energy consumption, we distinguish memory access at die level (with TSVs) and bank-level (without TSVs), referring to energy consumption reported in previous work [24]. For hardware validation, we implement a prototype of AttenPIM PU with RTL and evaluate it with Synopsys Design Compiler under 28nm CMOS technology.

B. Performance of AttenPIM

We evaluate AttenPIM's execution time against two baseline architectures, with results normalized to NeuPIMs for each dataset, as illustrated in Fig. 6. The total execution time comprises four distinct components: K preparation, V preparation, score computation ($Q\times K_t$), and context computation ($S\times V$).

First, AttenPIM demonstrates superior performance across all LLM datasets, achieving speedups of $1.20\times$–$5.26\times$ over NeuPIMs and $1.13\times$–$3.91\times$ over AttAcc. Compared to NeuPIMs, AttenPIM takes DRAM operation efficiency into consideration, significantly decreasing the time for V preparation. Compared to AttAcc, AttenPIM constrain all computations within bank PU, eliminating TSV bandwidth constraints.

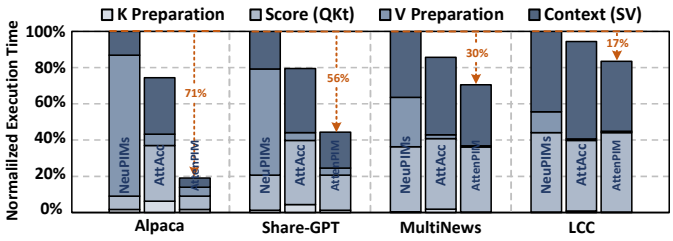


Fig. 6. Normalized performance comparison.

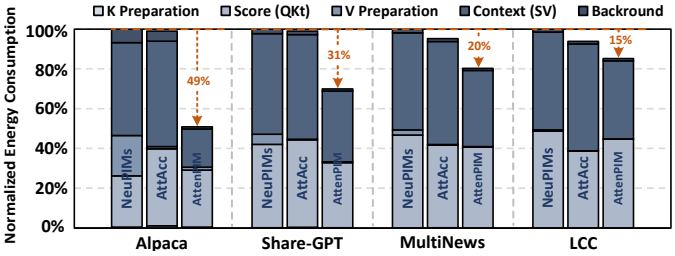


Fig. 7. Normalized energy consumption comparison.

Second, AttenPIM exhibits robust performance across varying token lengths. AttAcc shows worse performance with short token lengths than with long ones due to workload imbalance. NeuPIMs and AttenPIM partition heads to channels instead of HBM devices, whose bank number is usually smaller than token lengths, which results in less workload imbalance with short token lengths.

Third, the performance advantages of AttenPIM show a correlation with dataset token lengths. For individual token attention operations, the relative performance gain decreases as the number of previously generated tokens increases, primarily due to GEMV execution's growing dominance in total execution time. However, since long token sequences are generated incrementally from shorter sequences, AttenPIM's time reductions accumulate at each step, contributing to overall end-to-end inference efficiency.

C. Energy of AttenPIM

We compare the memory access energy consumption of AttenPIM and two baseline architectures. As illustrated in Fig. 6, results are normalized to the highest-consuming baseline for each dataset. The energy analysis is decomposed into five components: K preparation, V preparation, Score $Q \times K_t$, Context $S \times V$, and background (of entire execution).

First, AttenPIM demonstrates significant energy efficiency improvements, achieving reductions of 14.8%–48.1% and 14.4%–49.2% over NeuPIMs and AttAcc, respectively. The primary energy savings stem from optimized DRAM operations during V cache preparation (over NeuPIMs) and reduced memory accesses on TSVs (over AttAcc). Second, the energy efficiency advantages of AttenPIM also show a correlation with dataset token lengths. The energy waste of masked reads in NeuPIMs and TSV transfers in AttAcc are significant with short token lengths. For long token lengths, the energy for score and context would dominate the overall consumption.

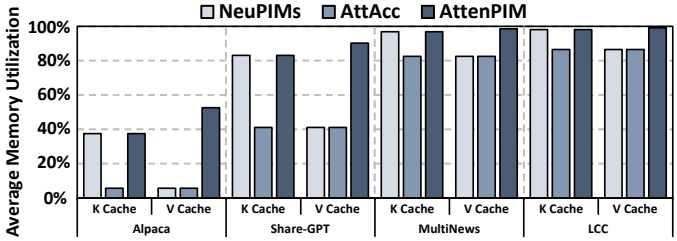


Fig. 8. Memory utilization of allocation method comparison.

TABLE IV
AREA AND POWER OVERHEAD

	NeuPIMs	AttAcc	AttenPIM	HBM
Area (mm^2)	0.79	1.58	0.86	>100 (DRAM process)
Power (W)	0.24	0.26	0.29	116 (Device Budget)

D. Memory Allocation Efficiency

We compare the memory utilization of allocation methods of AttenPIM and two baselines, as shown in Fig. 8. For K cache allocation, both NeuPIMs and AttenPIM demonstrate superior memory utilization compared to AttAcc across all evaluated datasets. This improved efficiency can be attributed to their optimized head distribution strategies. For V cache allocation, AttenPIM achieves the highest memory utilization through the burst block distributed data layout, which achieves memory utilization exceeding 90% as token length increases.

E. Overhead of AttenPIM

We implement bank PU of AttenPIM and two baseline designs in RTL and evaluate them using Synopsys Design Compiler. Based on NeuPIMs' configuration of banks per DRAM die, the result of 64 PUs is shown in Table IV. The evaluation shows that AttenPIM introduces negligible area and power overhead compared to NeuPIM, while achieving a 47% area reduction compared to AttAcc through the elimination of a large input buffer. When considering the scaling of PU area to the third-generation 10nm class (1z-nm) DRAM process, although the area density would be $10\times$ lower than the equivalent logic process, the overhead remains acceptable given the HBM specifications of more than $100mm^2$ DRAM dies and $116W$ power budget.

V. CONCLUSION

In this paper, we propose AttenPIM, a software-hardware co-design for efficient PIM-based attention acceleration. We propose PIM-oriented data layouts for KV storage and dual-mode computing paradigms tailored for score and context computations, which are supported by a configurable PIM unit design. In addition, we exploit token-level and head-level parallelism to optimize workload balance, while kernel fusion and dynamic allocation methods are proposed to further minimize memory overhead. Experimental results demonstrate that AttenPIM achieves $1.13\times$–$5.26\times$ speedup and reduces energy consumption by 17%–49% compared to two state-of-the-art PIM baselines.

REFERENCES

[1] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.

[2] H. Lee, G. Kim, D. Yun, I. Kim, Y. Kwon, and E. Lim, "Cost-effective llm accelerator using processing in memory technology," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[3] C. Li, Z. Zhou, S. Zheng, J. Zhang, Y. Liang, and G. Sun, "Specpim: Accelerating speculative inference on pim-enabled system via architecture-dataflow co-exploration," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 950–965.

[4] J. Choi, J. Park, K. Kyung, N. S. Kim, and J. H. Ahn, "Unleashing the potential of pim: Accelerating large batched inference of transformer-based generative models," *IEEE Computer Architecture Letters*, 2023.

[5] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "Attacc! unleashing the power of pim for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 103–119.

[6] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 722–737.

[7] G. Kim, J. Kim, N. Kim, W. Shin, J. Won, H. Joo, H. Choi, B. An, G. Shin, D. Yun *et al.*, "Sk hynix ai-specific computing memory solution: From aim device to heterogeneous aimx-xpu system for comprehensive llm inference," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–26.

[8] Y. Kwon, G. Kim, N. Kim, W. Shin, J. Won, H. Joo, H. Choi, B. An, G. Shin, D. Yun *et al.*, "Memory-centric computing with sk hynix's domain-specific memory," in *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 2023, pp. 1–26.

[9] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 372–385.

[10] Y. Kwon, K. Vladimir, N. Kim, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, G. Kim, B. An *et al.*, "System architecture and software stack for gddr6-aim," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE, 2022, pp. 1–25.

[11] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[12] F. Devaux, "The true processing in memory accelerator," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 2019, pp. 1–24.

[13] H. Kal, C. Yoo, and W. W. Ro, "Aespa: Asynchronous execution scheme to exploit bank-level parallelism of processing-in-memory," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 815–827.

[14] J. H. Kim, S.-h. Kang, S. Lee, H. Kim, W. Song, Y. Ro, S. Lee, D. Wang, H. Shin, B. Phuah *et al.*, "Aquabolt-xl: Samsung hbm2-pim with in-memory processing for ml accelerators and beyond," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–26.

[15] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[16] "High Bandwidth Memory DRAM (HBM3)." JEDEC,Standard, 2022.

[17] B. Hyun, T. Kim, D. Lee, and M. Rhu, "Pathfinding future pim architectures by demystifying a commercial pim technology," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 263–279.

[18] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[19] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[20] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," 2023.

[21] "ShareGPT," [Online], ShareGPT Team, 2023, avalable: https://sharegpt.com.

[22] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou *et al.*, "Longbench: A bilingual, multitask benchmark for long context understanding," *arXiv preprint arXiv:2308.14508*, 2023.

[23] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[24] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 41–54.

Leopard: Hardware Pass-Through Remote Storage Access with Queue Concurrency for Edge Intelligent Workstations

Wenjie Wang, Bo Peng, Jianguo Yao, Haibing Guan
Shanghai Jiao Tong University, Shanghai, China
{wang_wen_jie, pengbo_michael, jianguo.yao, hbguan}@sjtu.edu.cn

***Abstract*—Edge intelligent workstations load (store) massive empirical data from (to) remote cloud storage due to limited local storage. However, current remote storage access frameworks are complex. They use expensive computing resources to manipulate multiple concurrent request queues in modern high-speed storage devices for saturating performance. Complex software stacks and limited CPUs on edge intelligent workstations hinder saturating concurrent request queues, thus resulting in up to 75% performance degradation for remote storage. We propose Leopard, a hardware pass-through remote storage access framework with queue concurrency, which provides lossless remote storage access for edge intelligence. Leopard proposes a custom NVMe controller using SmartNIC's FPGA core to emulate it as an NVMe device, which eliminates complex remote storage stacks for edge workstations. Operations for remote storage access are implemented as hardware circuits inside the controller to eliminate CPU cycles. Parallelized and pipelined workflows are proposed for hardware circuits to accelerate remote storage access operations. Our evaluation presents that Leopard exhibits 1.09×∼6.04× lower remote storage access latency than SOTA solutions for realistic workloads in edge intelligent workstations.**

***Index Terms*—edge intelligent workstations, remote storage, hardware pass-through, queue concurrency, parallelized and pipelined workflows.**

I. Introduction

In recent years, more and more edge intelligent workstations [11], [43] have been deployed on the data generation side to perform edge intelligence. They reserve limited computing (CPUs or GPUs) and storage resources to implement edge intelligent applications using edge data and massive empirical data. The large volume of empirical data is persistently stored in remote cloud storage servers (remote storage) [42] and needs to be loaded/stored via remote storage access on demand. Remote storage access is implemented using a software stack known as the remote storage access framework.

There are several remote storage access frameworks have been proposed, such as famous and widely used iSCSI [1] and NVMe Over Fabric (*NVMe-oF*) [28]. iSCSI is a single-queue framework for traditional hard disk drives (HDDs) [7]. It introduces unacceptable performance degradation and CPU overheads for modern multi-queue non-volatile memory express [30] solid state drives (NVMe SSDs) [13]–[16] due to its inefficient single-queue design. Specifically, iSCSI reduces the application-level throughput of NVMe SSDs by >20% [10]. Furthermore, it needs to consume 14 CPU cores for saturating a single modern multi-queue NVMe SSD [12]. *NVMe-oF* is a remote storage framework with a multi-queue design. It mitigates the performance degradation for remote access by leveraging multiple physical request queues in modern NVMe SSDs. *NVMe-oF* can be categorized into NVMe-over-TCP [31] (*nvof-tcp*) and NVMe-over-RDMA [29] (*nvof-rdma*) according to the transport layer. *nvof-tcp* is constructed based on TCP [8] network protocol. *nvof-rdma* is constructed based on remote memory direct access (RDMA) [34] protocol and RDMA network card (RNIC) [23]. However, they are implemented as complex software stacks deployed on both Initiator and Target Hosts[1]. Multiple CPU cores are used to manipulate multiple physical request queues in SSDs to saturate performance. Experiments in Section III show that *nvof-tcp* reduces remote access performance of NVMe SSDs to about 1/4 of local access when using a 100Gbps network card (NIC). And *nvof-rdma* has a performance degradation of about 10%∼20% compared to local access [10].

Previous works [12], [26] have investigated the aforementioned disadvantages for traditional remote storage access frameworks. For example, an offloading method [26] is proposed to mitigate these disadvantages. Specifically, RNIC on the Target Host directly accesses NVMe SSDs in a peer-to-peer (P2P) manner, thereby slimming down the software stack on the Target Host. However, these works are proposed for cloud scenarios with sufficient CPU resources. Besides, they overlook multiple physical request queues in modern network devices, which can be used to improve concurrency for remote storage access. Figure 2 presents that these solutions still cannot avoid performance degradation for remote storage access in edge intelligent workstations when using modern high-performance NVMe SSDs. Therefore, we argue that existing remote storage access frameworks cannot fully leverage the queue concurrency in modern network devices and NVMe SSDs to implement high-performance remote storage access for edge intelligent workstations. The reasons are as follows: (i) The remote storage initiator (edge intelligent workstation) needs a complex software stack that integrates storage and network. (ii) The limited CPU resources in edge intelligent workstations cannot saturate multiple concurrent request queues in modern network devices and NVMe SSDs. (iii) The CPU still participates in remote storage access operations.

To this end, we propose Leopard, a hardware pass-through remote storage access framework using Field Programmable Gate Array (FPGA) [39]-based smart network cards (SmartNICs) [35]. It fully utilizes the queue concurrency in modern high-speed network devices and NVMe SSDs to implement lossless remote storage access for edge workstations. SmartNICs can independently perform tasks and optimize performance with low cost and low power consumption, such as [5], [20]–[22], [41]. The network functions in modern emerging SmartNICs also allow us to focus on investigating remote storage access workflows comprehensively without overly focusing on the network. First, Leopard customizes a native NVMe driver-compatible NVMe controller using SmartNIC's FPGA core to eliminate complex software stacks for Initiator Host. The Initiator Host only needs to submit remote storage requests without implementing remote-related operations. Second, Leopard constructs several FPGA-based accelerators that utilize multiple request queues in network devices and NVMe SSDs to accelerate remote storage access operations. Fi-

This work was funded by the National Key Research & Development Program of China (No. 2022YFB4502002), NSFC (No. 62402317 & 62032008), STCSM (No.24ZR1435500), and supported by the project from Wuxi Institute of Advanced Technology, a grant from State Key Laboratory of Avionics Integration and Aviation System-of-Systems Synthesis, and the Fund of Kunpeng&Ascend Center of Excellence, Shanghai Jiao Tong University. The corresponding authors are Bo Peng and Jianguo Yao.

[1]In this paper, we refer to edge workstations that perform edge computing and initiate remote storage requests as the Initiator Host, and remote cloud storage servers that manage persistent storage devices as the Target Host.

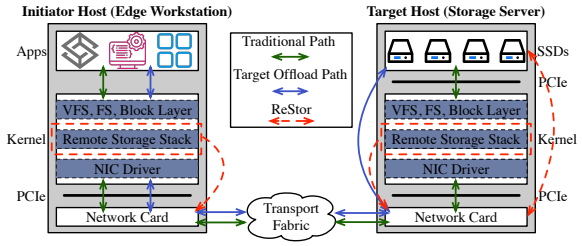


Fig. 1: Request/data path for remote storage access frameworks.

nally, Leopard accesses NVMe SSDs on remote cloud storage servers via SmartNIC's onboard peripheral component interconnect express (PCIe) slots. Leopard eliminates software stacks for Initiator Host and Target Host and provides queue-granularity remote storage access acceleration, thus optimizing remote storage access performance for edge workstations. The following are the main contributions:

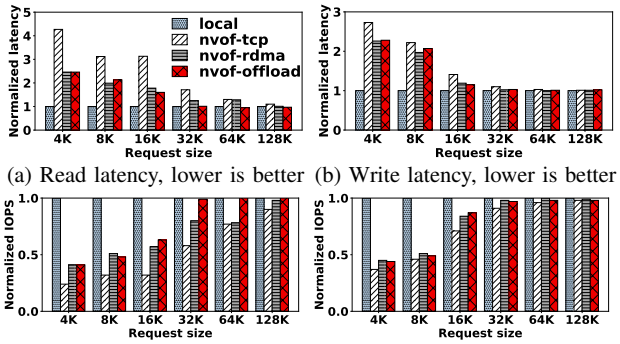
1) We propose Leopard, a hardware pass-through remote storage access framework based on SmartNICs. It enables remote storage performance close to local access for edge intelligent workstations with limited CPU resources.
2) We propose several hardware accelerators for remote storage access, which construct queue-granularity parallelized and pipelined workflows to fully exploit multiple request queues in network devices and NVMe SSDs. They accelerate request and data processing for remote storage access.
3) We implement a Leopard prototype and evaluate remote access latency and throughput using micro-benchmarks and realistic workloads. Experiments for realistic workloads exhibit that the remote read/write latency of Leopard is consistent with local access, which is 1.09×∼6.04× better than SOTA solutions.

II. Preliminaries and Related Work

A. Preliminaries

Edge Workstation. Edge workstations [11], [17], [43] are deployed in several fields to implement edge intelligence. They are implemented in remote sensing [33], healthcare [9], [24], manufacturing [4], and agriculture [36], [38] scenarios to collect geospatial, image, and sensor data. They also perform edge intelligent applications using empirical data and collected data. However, edge workstations typically reserve limited computing (CPUs/GPUs) and storage resources required by applications. The large amount of empirical data is persistently stored in remote cloud storage servers. Edge workstations read/write empirical data from/to remote cloud storage servers via remote storage access (frameworks). Therefore, remote storage access in this scenario affects the performance of applications.

Remote Storage Access. Figure 1 depicts the breakdown of remote storage access paths for traditional and target offloading [26] frameworks. Remote storage requests are initiated by applications on the Initiator Host and transmitted to NVMe SSDs on the Target Host. Remote access paths of traditional frameworks (*nvof-tcp* and *nvof-rdma*) are the longest. Specifically, storage requests initiated by applications (Apps) need to go through software stacks (file system, block layer, remote storage stack, and driver) and NIC on the Initiator Host, as well as NIC and software stacks on the Target Host. The target offloading framework utilizes RNIC to manipulate (P2P) NVMe SSDs, eliminating software stacks and PCIe overheads on the Target Host. However, remote storage access operations still consume edge workstation's CPUs due to software stacks present on the Initiator Host. Besides, existing frameworks utilize multiple CPU cores to operate multiple physical request queues in NVMe SSDs, thereby maximizing SSDs' performance. Unfortunately, CPU resources in edge intelligent workstations are limited. Therefore, complex software stack hierarchy and CPU dependency prevent traditional and target offloading frameworks from implementing efficient



(a) Read latency, lower is better (b) Write latency, lower is better

(c) Read IOPS, higher is better (d) Write IOPS, higher is better

Fig. 2: Remote read/write latency and throughput for *nvof-tcp*, *nvof-rdma*, and *nvof-offload*. request size=4KB(K)∼128KB(K), thread count=1, request depth=4, I/O engine=libaio.

remote storage access for edge intelligent workstations. Next, we provide an overview of existing remote storage access frameworks.

B. Related Work

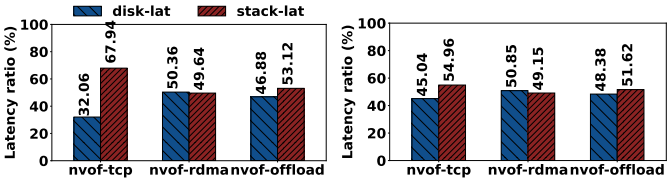
Traditional Remote Storage Frameworks. iSCSI [1] is a traditional remote storage access framework designed for low-speed HDDs. It is unable to provide low-latency and high-throughput remote storage access as the development of modern network and storage devices. *NVMe-oF* [28] was proposed in 2014 and scaled to Ethernet, Fiber, and InfiniBand [32] in 2016. *NVMe-oF*'s multi-queue design achieves better latency and throughput performance for remote storage access compared to iSCSI. *nvof-tcp* and *nvof-rdma* are two classic *NVMe-oF* protocols. *nvof-tcp* introduces significant performance overhead compared to local access since the network layer adopts the TCP protocol, which is implemented by CPU software. *nvof-rdma* is proposed to mitigate the performance degradation for remote storage access using RDMA protocol and RNIC. However, the remote access performance of *nvof-rdma* in edge intelligent workstations is still 10%∼20% worse than local access.

Remote Storage Framework Optimization. Several solutions [12], [19], [26] for optimizing existing remote storage access frameworks have been proposed. For example, *i10* [12] is proposed to optimize the performance of *nvof-tcp*. Its performance is similar to *nvof-rdma*. Highly optimized pipelines between CPU cores and remote storage access stacks are proposed to optimize bottlenecks at the boundary of the network stack and NVMe storage stack. Besides, NVME-OF TARGET OFFLOAD [26] is proposed to use RNIC's P2P ability for directly accessing NVMe SSDs, thus eliminating software stacks and PCIe overheads on the Target Host. ReFlex [19] designs and implements user-space storage and network frameworks to access remote Flash storage efficiently. However, the deployment and migration of user-space frameworks are complicated. They not only do not support POSIX interfaces but also are user-aware. Besides, these frameworks are designed for cloud scenarios with sufficient CPU resources. Their remote access performance is still worse than local access for edge workstations with limited CPUs.

III. Performance Characteristics

We conduct experiments to investigate performance bottlenecks and optimization opportunities for existing remote storage access frameworks in edge intelligent workstations. The software/hardware configuration of experiments is introduced in Section V. Intel SSD DC P4600 (P4600) [16] is used as the target disk.

Latency. Figures 2a and 2b present the remote read/write latency of existing remote storage access frameworks. Flexible I/O (FIO) [3] workloads with different request sizes (4KB∼128KB) are used to collect evaluation data. The thread count and request depth of workloads are 1 and 4, and the I/O engine is libaio. *local* indicates



(a) Read latency breakdown (b) Write latency breakdown

Fig. 3: Remote read/write latency breakdown for *nvof-tcp*, *nvof-rdma*, and *nvof-offload*.

TABLE I: The size of data in a single storage request for applications in edge intelligent workstations.

Data Type	Data Size	Data Type	Data Size
Sensor data [40]	<10KB	Log file	<100KB
Device status	<10KB	Image [18]	<100KB
Calculation result	<10KB	Configuration file	<100KB

accessing local SSDs, and *nvof-offload* indicates the target offloading framework proposed in [26]. Latency values are normalized to *local*. We make an observation from Figure 2a that the remote access constructed by existing remote storage access frameworks has higher read/write latency than *local*. Specifically, the remote read latency of *nvof-tcp* is 1.1×∼4.2× higher than the local read when the request size is <128KB. The remote read latency is still 1.63× and 1.52× higher than the local read on average, even using *nvof-rdma* and *nvof-offload* frameworks. Similarly, the remote write latency (Figure 2b) of *nvof-tcp*, *nvof-rdma*, and *nvof-offload* is higher than the local write when the request size is <32KB. **Therefore, existing remote storage access frameworks cannot provide low-latency remote storage access for edge intelligent workstations.**

Throughput. The number of I/O requests processed per second (IOPS) is the metric for storage throughput. FIO workloads in latency experiments are used to collect the remote read/write throughput for *nvof-tcp*, *nvof-rdma*, and *nvof-offload* frameworks. The remote read/write throughput exhibits a similar trend to the remote read/write latency, as shown in Figures 2c and 2d. The remote read/write IOPS of *nvof-tcp*, *nvof-rdma*, and *nvof-offload* frameworks is reduced to 1/4 and 1/3 of the local access when the request size is <128KB/32KB. **The throughput observation reveals that existing remote storage access frameworks in edge intelligent workstations cannot exploit the high throughput of modern high-speed NVMe SSDs.**

It is worth noting that the latency and throughput penalties introduced by existing frameworks decrease with increased read/write size. The insight is that NVMe SSDs handle small-size read/write faster than software remote storage access stacks, resulting in remote access stacks being the bottleneck for small-size remote read/write. On the contrary, SSDs require more time to handle large-size read/write, which leads to the overhead of remote access stacks being negligible as the read/write size increases. Therefore, existing remote storage access frameworks are more suitable for large-size read/write scenarios in edge intelligent workstations. However, **the small-size read/write is a common I/O characteristic for practical applications in edge intelligent workstations**, as shown in Table I.

Latency Breakdown. We breakdown the read/write latency of 8KB FIO workload for *nvof-tcp*, *nvof-rdma*, and *nvof-offload* frameworks to investigate the performance bottleneck. As shown in Figure 3, *disk-lat* represents the read/write latency of the SSD, and *stack-lat* represents the latency of remote storage access stacks. It is evident that *stack-lat* accounts for >49% of remote read/write latency in *nvof-tcp*, *nvof-rdma*, and *nvof-offload* frameworks, which results in SSD having higher remote read/write latency than local access. Next, optimization methods and insights in Leopard for mitigating these overheads are discussed.

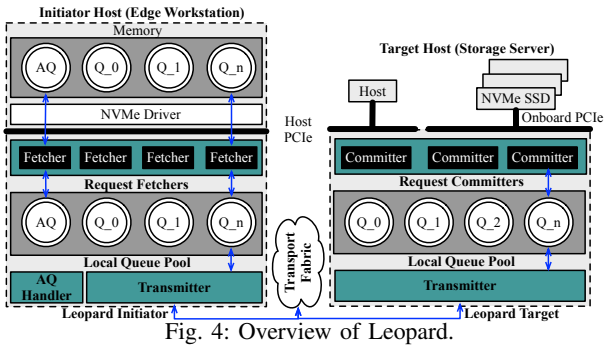


Fig. 4: Overview of Leopard.

IV. Leopard Design

A. Leopard Overview

We propose a hardware pass-through framework called Leopard. It consists of two SmartNICs, as shown in Figure 4. We customize them as Leopard Initiator and Leopard Target and connect them to Initiator Host and Target Host via PCIe. **Request Fetchers**, **Local Queue Pool**, **Admin Queue Handler (AQ Handler)**, **Transmitter**, and **Request Committers** are five novel accelerators. The processing of NVMe storage requests, including remote storage, is characterized by high density and high concurrency, as stated in the NVMe (Over Fabric) specification [28], [30]. FPGA is more suitable for concurrent tasks than traditional processors (such as x86 and ARM [2] CPUs). It also reserves the flexibility to customize hardware interfaces, including PCIe. Therefore, Leopard adopts FPGA-based SmartNICs instead of ARM-based SmartNICs [27].

Request Fetchers & Request Committers. Request Fetchers are implemented in Leopard Initiator. They parallelly fetch storage requests from Initiator Host memory using the direct memory access (DMA) engine of PCIe. It aims to minimize the fetching latency for remote storage requests as much as possible. Similarly, Request Committers are implemented in Leopard Target. They parallelly submit storage requests to NVMe SSDs using SmartNIC's onboard PCIe DMA engine. It aims to minimize the latency of remote storage requests reaching NVMe SSDs. The philosophy behind these accelerators is the potential parallelism between multiple NVMe request queues and multiple PCIe DMA channels. Specifically, multiple PCIe DMA channels in SmartNICs are used to parallelly fetch remote storage requests from multiple NVMe request queues in Initiator Host memory. They are also used to parallelly submit requests to multiple NVMe request queues in SSDs.

Local Queue Pool. Local Queue Pool is allocated in Leopard Initiator and Leopard Target to cache remote storage requests, aiming to construct multiple independent (queue-granularity) request processing pipelines. The Local Queue Pool in Leopard Initiator caches remote storage requests fetched from Initiator Host memory. And the Local Queue Pool in Leopard Target caches remote storage requests received from the Transmitter (Transport Fabric, Network). It consists of multiple isolated first-in-first-out (FIFO) buffers, corresponding to multiple NVMe request queues (Q_0∼Q_n) that Leopard Initiator exposes to Initiator Host. Leopard Initiator follows NVMe specification [30], thus each exposed NVMe request queue consists of a submission queue (SQ) and a completion queue (CQ), called SQ/CQ pair. Every two FIFOs are firmly bound to an SQ/CQ pair. One FIFO is used to cache pending requests submitted by the Initiator Host to SQ. Another FIFO is used to cache completion requests submitted by NVMe SSDs to CQ. The FIFO width is set to 512 bits for achieving compatibility with NVMe storage requests. The depth of FIFO is set to 1024 (1K) to construct request processing pipelines at queue granularity. Note that FIFOs are implemented using FPGA's on-chip memory that supports independent read/write. It aims to leverage parallelism in NVMe request queues and PCIe DMA channels.

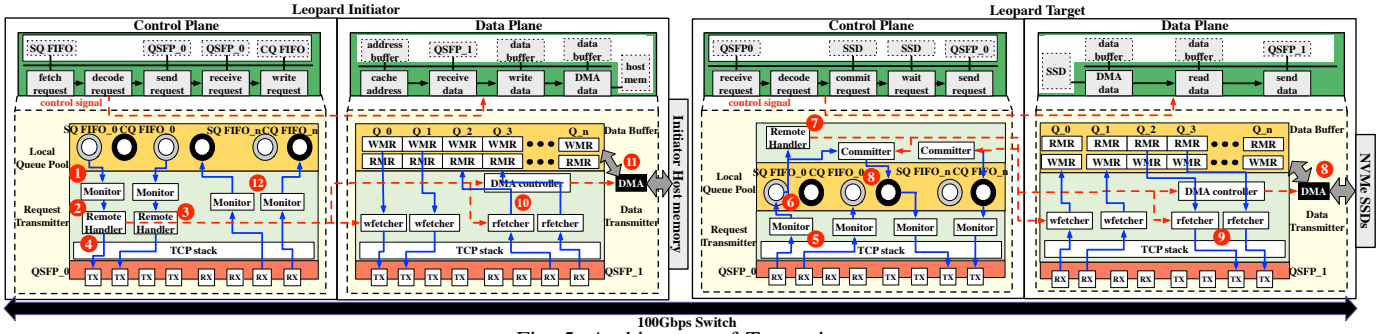


Fig. 5: Architecture of Transmitter.

AQ Handler. Initiator Host needs a dedicated software stack to implement remote storage access using existing frameworks. The software stack in edge workstations introduces performance overheads and CPU costs, as discussed in Sections III and II-A. AQ Handler disguises Leopard Initiator as a standard NVMe device compatible with the native NVMe driver, which eliminates remote storage access stacks for Initiator Host. In other words, AQ Handler is a customized NVMe controller. It allows Initiator Host's CPUs to simply submit remote storage requests to Leopard Initiator (via the native NVMe driver) for remote access without additional operations. The remote-access-related workflow is implemented by Leopard Initiator and Target. AQ Handler uses a workflow that follows the NVMe specification [30] to process NVMe admin requests submitted by the Initiator Host. NVMe admin requests are mainly used to manage and identify Leopard Initiator, such as creating and deleting NVMe request queues (Q_0∼Q_n). Therefore, admin requests submitted to the admin queue (AQ) are only cached to the corresponding FIFO in Leopard Initiator's Local Queue Pool, then processed by AQ Handler without being transmitted to Leopard Target.

Transmitter. Transmitter is the core accelerator for Leopard, which implements efficient hardware pass-through remote storage access. It not only accelerates storage and network stacks but also better utilizes multiple request queues in network devices and NVMe SSDs compared to existing solutions. Transmitter constructs a hardware pass-through queue-granularity parallelized and pipelined remote storage request processing flow between Leopard Initiator and Leopard Target, which eliminates CPU involvement. It mitigates software stacks and CPU overheads introduced by current remote storage access frameworks for Initiator and Target Hosts. Remote Handler and Monitor are its two hardware components. Remote Handler is used to implement the control plane for remote storage access (including remote storage requests and data). Monitor is used to poll FIFOs in Local Queue Pool or network receivers to fetch remote storage requests as fast as possible, thereby reducing latency. Next, we will discuss the detailed design for Request Fetchers, Request Committers, and Transmitter, which are closely related to remote storage access performance.

B. Design Details

Request Fetchers & Committers. The same number of Fetchers and Committers as NVMe request queues are implemented in Request Fetchers and Committers (Figure 4) to achieve queue-granularity fetching and submitting for remote storage requests. They exploit the parallelism of PCIe DMA channels in SmartNICs and NVMe request queues in SSDs by proposing independent hardware units (instead of multiple CPU cores). Fetcher and Committer consist of one Monitor and one DMA Controller, which are used to implement control and data planes for fetching and submitting remote storage requests.

Fetcher's workflow is as follows. **S1:** Monitor monitors the SQ of NVMe request queues in Initiator Host memory. It also monitors corresponding CQ FIFO in the Local Queue Pool of Leopard Initiator. **S2:** As soon as the Monitor detects a new remote storage (completion) request in SQ (CQ FIFO), it sends address information and control signal to the DMA Controller. **S3:** DMA Controller then manipulates PCIe DMA to copy remote storage requests from Initiator Host memory to corresponding SQ FIFO located in the Local Queue Pool of Leopard Initiator. Completion requests are copied from CQ FIFO to CQ of the corresponding NVMe request queue located in the Initiator Host memory.

The workflow of Committer is similar to Fetcher. The difference is that it implements queue-granularity NVMe request interaction with NVMe SSDs via onboard PCIe DMA in Leopard Target. It eliminates software stacks, as well as PCIe and CPU overheads for remote storage access in Target Host. We equip each SQ/CQ pair with a Fetcher and a Committer to fully utilize the parallelism between NVMe request queues. The independent Fetcher and Committer are also adopted within each SQ/CQ pair to achieve parallelized transmission for the SQ/CQ pair's internal requests.

Transmitter. Software stacks and limited CPU resources are the bottleneck for small-size remote read/write in edge workstations, as discussed in Sections III and II-A. Transmitter is proposed to further mitigate software stacks and CPU overheads for Initiator Host and Target Host. It implements hardware pass-through processing for remote storage requests and data to replace software remote storage access stacks. It also leverages parallelism among multiple NVMe request queues, multiple network channels, and multiple PCIe DMA channels to construct parallelized and pipelined workflows. Figure 5 depicts the architecture of Transmitter, which proposes a highly parallelized and pipelined processing flow. It is divided into a control plane and a data plane. Four full-duplex network (QSFP_0) channels are used in the control plane to transmit remote storage requests parallelly. Another four full-duplex network (QSFP_1) channels are used in the data plane to transmit data triggered by requests.

We use remote read to demonstrate the Transmitter's workflow. ①**:** In Leopard Initiator, independent Monitors poll corresponding SQ FIFO in the Local Queue Pool and notify the Remote Handler in the control plane if the SQ FIFO is not empty. ②**:** The Remote Handler fetches remote storage requests from the non-empty SQ FIFO. Note that multiple Remote Handlers are deployed to implement queue-granularity acceleration. ③**:** Remote Handler decodes the request's opcode field (OPC, byte 0). If OPC is the read operation, the Remote Handler decodes the target address and size of the data from the request. They are buffered into the corresponding DMA address buffer (in the DMA controller) in the data plane. The corresponding rfetcher in the data plane is also notified to poll the data receiving channel in QSFP_1. ④**:** Then, requests are sent to Leopard Target via the hardware TCP stack and dedicated channel in QSFP_0. Leopard adopts TCP for compatibility with most edge scenarios.

Leopard Target further processes remote storage requests. Specifically, ⑤**:** The multiple Monitors in Leopard Target's control plane poll four receiving channels in QSFP_0. ⑥**:** Monitors dispatch received requests to corresponding SQ FIFO in the Local Queue Pool. ⑦**:** The

TABLE II: Configuration of Initiator and Target Hosts.

Initiator Host (Edge Workstation)	AMD Ryzen 5 5600X 6-Core processor Mellanox ConnectX-5 [23] Western Digital (WD) HDD, 1TB DDR4 2666MHz, 16GB FPGA-based SmartNIC [35]
Target Host (Remote Cloud Storage Server)	Intel(R) Xeon(R) CPU E5-2680 processor Mellanox ConnectX-5 Western Digital (WD) HDD, 4TB DDR4 3200MHz, 64GB Intel SSD DC P4600 (P4600) [16], 2TB FPGA-based SmartNIC

TABLE III: Hardware cost & power of Leopard Initiator & Target.

	LUT	LUTRAM	FF	BRAM	Power (W)
Initiator	173719	10703	248981	536	16.67
	33.23%	6.64%	23.82%	54.47%	
Target	110721	10111	190547	386	14.19
	21.18%	6.27%	18.23%	39.23%	

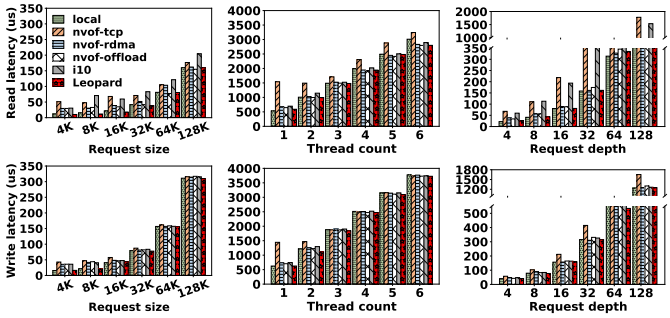
Remote Handler in Leopard Target's control plane fetches requests from SQ FIFO and parses OPC and data address fields. If OPC is the read operation, the Remote Handler utilizes the source address and size of the data to inform the DMA controller in Leopard Target's data plane. Besides, the Committer is also notified to submit requests to the corresponding request queue in SSD via onboard PCIe and wait for SSD's completion request. ⑧**:** The data will be read by DMA (manipulated by the DMA controller) from SSD to the memory region (RMR) of the corresponding SQ FIFO in the data buffer. The SSD's completion request is written to CQ FIFO by Committer and transmitted to Leopard Initiator by Monitor via QSFP_0. ⑨**:** Hardware TCP stack in Leopard Target's data plane encapsulates data in RMR into TCP packets and sends them to Leopard Initiator via the dedicated channel in QSFP_1.

Leopard Initiator then processes data. ⑩**:** The refetcher in Leopard Initiator receives data from the receiving channel in QSFP_1 and writes it to the corresponding RMR in the data buffer. ⑪**:** The data in RMR will be transferred by DMA to the target address (cached in Step ③) in Initiator Host memory. ⑫**:** Finally, the Monitor in the Leopard Initiator's control plane receives the completion request from the Leopard Target and writes it to the corresponding CQ FIFO in the Leopard Initiator's Local Queue Pool. Note that the data buffer is constructed using FIFO for correctness, which allows data to be processed in order of request submission.

The workflow for remote write is similar to remote read, but the data transmission direction is from Initiator Host memory to SSDs. We propose optimizing steps ①∼⑫ using queue-granularity parallelized and pipelined workflows, thereby accelerating the remote-access-related processing for storage requests and data.

V. Evaluation

Hardware Setup. Two FPGA-based SmartNICs [35], one edge workstation (Initiator Host), and one cloud storage server (Target Host) are used to construct the testbed, as listed in Table II. The FPGA-based SmartNIC contains one Xilinx xczu19eg-ffvc1760-2-e FPGA core and several communication interfaces, such as PCIe and QSFP [25] (100Gbps). We implement the Leopard prototype on FPGA (Frequency 250 MHz) using Verilog. Two FPGA-based SmartNICs are implemented as Leopard Initiator and Leopard Target, respectively. They connect to the Initiator Host and Target Host via PCIe. PCIe is configured as Gen3×8 in our implementation. Leopard is also compatible with high version PCIe (Gen4 or Gen5). Besides, Target Host is equipped with a 2TB Intel SSD DC P4600 (P4600) to test remote access performance for Leopard and baselines. The Initiator Host and Target Host are connected to a 100Gbps switch via Mellanox ConnectX-5 (RDMA NIC), and P4600 is connected to the Target Host via rack PCIe for testing baselines. The Initiator Host and Target Host are connected to the 100Gbps switch via Leopard, and P4600 is directly connected to Leopard Target via onboard PCIe for testing Leopard.



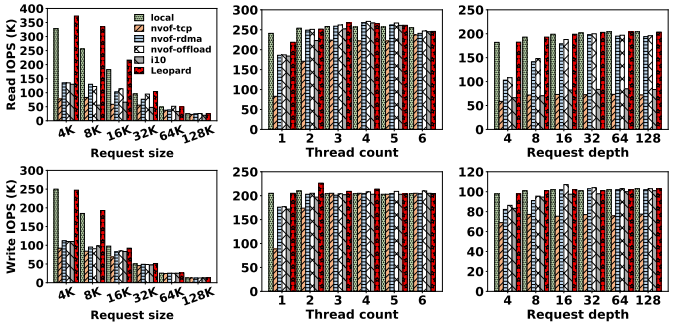
(a) Request size (b) Thread count (c) Request depth

Fig. 6: Remote read and write latency comparison using FIO workloads, lower is better.

Software Setup & Baselines & Metrics. Ubuntu 20.04.4 LTS operating system with Linux kernel 5.10.73 is installed on the Initiator Host and Target Host. FIO (version 3.31) [3], RocksDB [37], and YCSB [6] are installed to generate evaluation workloads. All workloads are running on Initiator Host. We adopt the following baselines: 1) ***local***: This is local access performance obtained by directly accessing local P4600 on the Initiator Host. 2) ***nvof-tcp*** [31]: Software remote storage access framework using TCP. 3) ***nvof-rdma*** [29]: Software remote storage access framework using RDMA. 4) ***i10*** [12]: Software optimized *nvof-tcp*. 5) ***nvof-offload*** [26]: *nvof-rdma* with target offload. We adopt remote read/write latency, throughput, and CPU cost to evaluate performance.

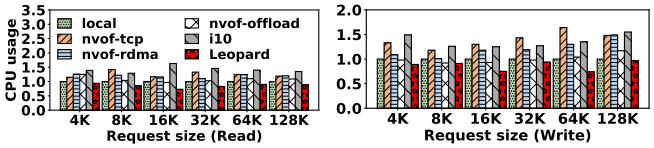
Hardware Cost & Power. The hardware cost and power consumption of Leopard are directly proportional to the number of parallel request queues (workflows). However, edge intelligent workstations typically adopt devices with low cost and low energy to minimize the total cost of ownership (TCO). Fortunately, we find that excessive queues (workflows) in Leopard result in unacceptable channel arbitration overhead due to limited channels in SmartNIC's PCIe DMA and network interfaces. The arbitration overhead will compromise remote storage access performance. Leopard adopts 32 queues (workflows), which is less than the request queues within modern NVMe SSDs, but evaluation shows that it is sufficient to saturate modern SSDs. If faster SSDs are used in the future, Leopard's design can scale at queue granularity using sufficient FPGA resources to maximize SSDs' remote access performance. Leopard Initiator and Target only consume 29.54%/21.23% of FPGA resources on average (Table III), which is approximately $174.65/$125.66 in hardware cost. The power consumption of Leopard Initiator and Leopard Target is 16.67 watts (W) and 14.19W, which is lower than implementing remote storage access using CPUs (>100W).

Latency and Throughput. We use remote read/write latency and throughput metrics to compare Leopard with baselines. FIO is installed on the Initiator Host. It generates sequential read/write workloads with different request sizes (4KB∼128KB), different thread counts (1∼6), and different request depths (4∼128) to access SSD on the Target Host via Leopard and baseline frameworks. The I/O engine (ioengine parameter in FIO) is set to libaio. Existing remote storage access frameworks are unfriendly to small-size requests in edge workstations, as discussed in Section III. Therefore, the small-size workloads are adopted in experiments with different request sizes to present Leopard's improvements for edge workstations. Figure 6 presents the remote read/write latency. It is obvious that only Leopard enables remote read/write latency to be consistent with



(a) Request size (b) Thread count (c) Request depth

Fig. 7: Remote read and write throughput comparison using FIO workloads, higher is better.



(a) Remote read, lower is better (b) Remote write, lower is better

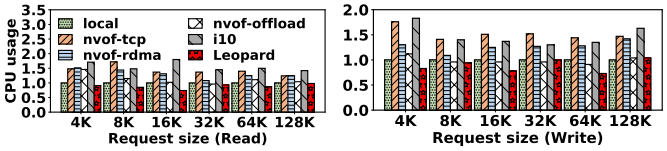
Fig. 8: CPU cost for remote read/write, normalized to ***local***.

local read/write (***local***). Compared to baselines (***nvof-tcp***,***nvof-rdma***, ***nvof-offload***, and ***i10***), Leopard's remote read (write) latency for workloads with different parameters (request size, thread count, and request depth) is reduced by 2.59×, 1.75×, 1.66×, and 2.61× (1.49×, 1.34×, 1.36×, and 1.34×) on average. Leopard's remote read/write throughput is also better than baselines, achieving the same throughput as local read/write. The remote read (write) throughput of Leopard is 2.55×, 1.73×, 1.63×, and 2.60× (1.51×, 1.35×, 1.38×, and 1.36×) higher than baselines on average, as shown in Figure 7.

Leopard is more suitable for scenarios with small request sizes, according to Figures 6a and 7a. The response of SSD is fast in this scenario, causing existing complex remote storage access frameworks to become a bottleneck, as discussed in Section III. However, the small-size request is a typical remote storage access type for real applications in edge workstations (Table I). Leopard optimizes this bottleneck by (1) Customizing a highly parallelized and pipelined hardware pass-through workflow for fast remote storage access. (2) Integrating remote access tasks into FPGA-based SmartNIC to mitigate software layers and PCIe and CPU overheads for remote storage access. Therefore, Leopard is more suitable for edge workstations.

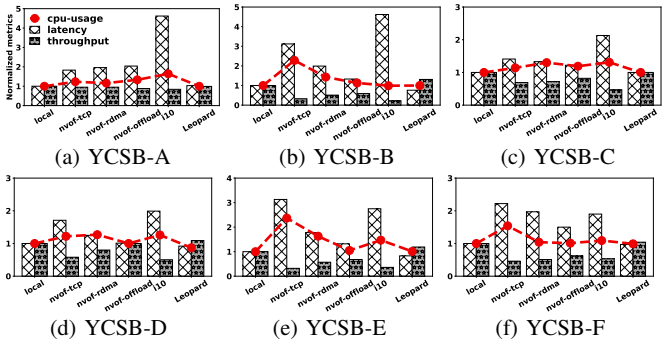
CPU Cost. Existing frameworks use multiple CPU cores to maximize the remote access performance for SSDs, as discussed in Section II-A. This not only introduces latency overhead but also incurs additional CPU cycles for edge intelligent workstations with limited CPU resources. We measure CPU usage in Initiator and Target Hosts for Leopard and baselines using FIO workloads with different request sizes (4KB∼128KB). The queue depth and thread count are 16 and 8. The I/O engine is set to libaio. Figure 8 presents the experimental results, which are normalized to ***local*** values for intuitive comparison. The CPU usage of Leopard is close to local access. Compared to baselines (***nvof-tcp***,***nvof-rdma***, ***nvof-offload***, **and** ***i10***), Leopard's CPU usage for remote read (write) is reduced by 2.90×, 1.58×, 1.33×, and 2.22× (2.22×, 1.76×, 1.41×, and 1.83×).

Figure 9 also presents the CPU usage in kernel space for Leopard and baselines. Leopard's read (write) CPU usage in kernel space is 1.64×, 1.50×, 1.28×, and 1.80× (1.74×, 1.45×, 1.13×, and 1.69×) lower than baselines. The advantage of CPU cycles for Leopard comes from two factors. (1) Leopard Initiator is compatible with the native NVMe driver. It allows Initiator Host's CPUs to simply submit remote storage requests to Leopard Initiator for remote access without additional operations. Leopard Initiator enables edge workstations to access SSDs on remote cloud storage servers as if they were accessing



(a) Remote read (b) Remote write

Fig. 9: CPU cost in kernel space for remote frameworks, normalized to ***local***, lower is better.



(a) YCSB-A (b) YCSB-B (c) YCSB-C

(d) YCSB-D (e) YCSB-E (f) YCSB-F

Fig. 10: Remote read/write comparison using YCSB workloads, all experimental values are normalized to ***local***.

local SSDs. (2) The processing flow (including storage and network) for remote read/write is handled by Leopard Initiator and Leopard Target without CPU intervention. Therefore, Leopard is more suitable for edge intelligent workstations with limited CPUs.

Realistic Workloads. The latency, throughput, and CPU usage (cpu-usage) of remote storage access for Leopard and baselines are collected using realistic workloads to evaluate the benefits of Leopard in real-world scenarios. The YCSB [6] workloads based on RocksDB [37] are used to collect evaluation metrics. YCSB is a widely used evaluation tool for storage systems. It can generate standard and customized realistic workloads. Figure 10 collects the remote access latency, throughput, and CPU usage for Leopard and baselines using standard YCSB-A∼YCSB-F workloads. YCSB-A∼YCSB-F are six different realistic workloads. They have different request types (read-only, write-only, mixed, read-modify-write, etc), different request sizes, different read/write ratios, different thread counts, and different request depths. All experimental values are normalized to ***local*** for intuitive comparison. As shown in Figures 10a∼10f, Leopard provides a remarkable optimization in remote storage access latency and CPU usage for YCSB-A∼YCSB-F, achieving an optimization of 1.09×∼6.04× and 1.03×∼2.34× compared to existing remote storage access frameworks. It also exhibits a 1.04×∼6.02× improvement for throughput. More importantly, Leopard provides the same latency, throughput, and CPU usage for remote storage access and local storage access. This experiment presents that Leopard effectively maintains remote storage access performance consistent with local access for realistic workloads in edge intelligent workstations.

VI. Conclusion

We present Leopard, an efficient hardware pass-through remote storage access framework based on SmartNIC for edge intelligent workstations. The end-to-end remote storage access workflow is formulated as multiple accelerators in SmartNIC's FPGA core. The AQ Handler helps Leopard adopt the native NVMe driver, thus eliminating the dedicated software stack. Request Fetchers, Request Committers, Local Queue Pool, and Transmitter construct a hardware pass-through remote storage access workflow using pipeline architecture. In the workflow, the concurrency of request queues, DMA channels, and network interfaces is fully utilized for fast remote access. The evaluation presents that Leopard provides lossless remote storage access for edge intelligence in edge intelligent workstations.

REFERENCES

[1] S. Aiken, D. Grunwald, A. R. Pleszkun, and J. Willeke, "A performance analysis of the iscsi protocol," in 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. IEEE, 2003, pp. 123–134.

[2] M. N. Asghar, "A review of arm processor architecture history, progress and applications," Journal of Applied and Emerging Sciences, vol. 10, no. 2, pp. pp–171, 2020.

[3] J. Axboe, "Fio: Flexible i/o tester," https://github.com/axboe/fio, (Accessed 8 March 2024).

[4] A. Bonello, E. Francalanza, and P. Refalo, "Smart and sustainable human-centred workstations for operators with disability in the age of industry 5.0: A systematic review," Sustainability, vol. 16, no. 1, p. 281, 2023.

[5] M. Bonola, G. Belocchi, A. Tulumello, M. S. Brunella, G. Siracusano, G. Bianchi, and R. Bifulco, "Faster software packet processing on fpga nics with ebpf program warping," in 2022 USENIX Annual Technical Conference (USENIX ATC 22), 2022, pp. 987–1004.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in Proceedings of the 1st ACM symposium on Cloud computing, 2010, pp. 143–154.

[7] W. Digital, "WD Gold Enterprise Class SATA HDD," https://www.westerndigital.com/products/internal-drives/wd-gold-sata-hdd?sku=WD1005FBYZ, (Accessed 10 November 2023).

[8] B. A. Forouzan, TCP/IP protocol suite. McGraw-Hill Higher Education, 2002.

[9] W. Gong, Z. Xu, L. Yuan, H. Zhou, S. Chen, Y. Ding, A. Nayak, and J. Liu, "Embracing self-powered wearables for intelligent healthcare data management," IEEE Internet of Things Journal, 2024.

[10] Z. Guz, H. Li, A. Shayesteh, and V. Balakrishnan, "Performance characterization of nvme-over-fabrics storage disaggregation," ACM Transactions on Storage (TOS), vol. 14, no. 4, pp. 1–18, 2018.

[11] K. Hsu and H. Tseng, "Accelerating applications using edge tensor processing units," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–14.

[12] J. Hwang et al., "Tcp≈rdma: Cpu-efficient remote storage access with i10," in 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 2020, pp. 127–140.

[13] Intel, "Intel Optane SSD 900P Series," https://www.intel.com/content/www/us/en/products/sku/123626/intel-optane-ssd-900p-series-480gb-12-height-pcie-x4-20nm-3d-xpoint/specifications.html, (Accessed 10 November 2023).

[14] Intel, "Intel Optane SSD DC P4800X/P4801X series," https://www.intel.com/content/www/us/en/products/docs/memory-storage/solid-state-drives/data-center-ssds/optane-ssd-dc-p4800x-p4801x-brief.html, (Accessed 10 November 2023).

[15] Intel, "Intel® optane™ dc ssd series," https://www.intel.com/content/www/us/en/products/details/memory-storage/data-center-ssds/optane-dc-ssd-series.html, (Accessed 10 November 2023).

[16] Intel, "Intel® ssd dc p4600 series," https://ark.intel.com/content/www/us/en/ark/products/series/96947/intel-ssd-dc-p4600-series.html, (Accessed 10 November 2023).

[17] A. Isosalo, J. Islam, H. Mustonen, E. Räinä, S. I. Inkinen, M. Brix, T. Kumar, J. Reponen, M. T. Nieminen, and E. Harjula, "Local edge computing for radiological image reconstruction and computer-assisted detection: A feasibility study," Finnish Journal of eHealth and eWelfare, vol. 15, no. 1, pp. 52–66, 2023.

[18] V. Jeyakumar, K. R. Abirami, S. Saraswathi, R. S. Kumaran, and G. Marthi, "Secure medical image storage and retrieval for internet of medical imaging things using blockchain-enabled edge computing," in Intelligent Edge Computing for Cyber Physical Applications. Elsevier, 2023, pp. 85–110.

[19] A. Klimovic, H. Litz, and C. Kozyrakis, "Reflex: Remote flash≈local flash," ACM SIGARCH Computer Architecture News, vol. 45, no. 1, pp. 345–359, 2017.

[20] D. Kwon, J. Boo, D. Kim, and J. Kim, "Fvm: Fpga-assisted virtual device emulation for fast, scalable, and flexible storage virtualization," in 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, pp. 955–971.

[21] J. Lin, T. Ji, X. Hao, H. Cha, Y. Le, X. Yu, and A. Akella, "Towards accelerating data intensive application's shuffle process using smartnics," Proceedings of the ACM on Measurement and Analysis of Computing Systems, vol. 7, no. 2, pp. 1–23, 2023.

[22] J. Liu, A. Dragojević, S. Flemming, A. Katsarakis, D. Korolija, I. Zablotchi, H.-c. Ng, A. Kalia, and M. Castro, "Honeycomb: ordered key-value store acceleration on an fpga-based smartnic," IEEE Transactions on Computers, 2023.

[23] N. Mellanox, "NVIDIA Mellanox ConnectX-5 Ethernet Adapter Cards User Manual," https://img-en.fs.com/file/user_manual/connectx-5-ethernet-adapter-cards-user-manual.pdf, (Accessed 10 November 2023).

[24] K. Mishra and S. Yadav, "Edge intelligence–a smart healthcare scenario in ambient-assisted living," in Reconnoitering the Landscape of Edge Intelligence in Healthcare. Apple Academic Press, 2024, pp. 157–171.

[25] Q.-D. MSA, "Qsfp-dd hardware specification for qsfp double density 8x pluggable transceiver," Revision, vol. 1, pp. 1–49, 2019.

[26] NVIDIA, "Nvme-of target offload," https://enterprise-support.nvidia.com/s/article/howto-configure-nvme-over-fabrics--nvme-of--target-offload, (Accessed 1 November 2023).

[27] M. NVIDIA, "Nvidia mellanox bluefield smartnic for ethernet," https://network.nvidia.com/files/doc-2020/pb-bluefield-smart-nic.pdf, (Accessed 10 November 2023).

[28] NVMEXPRESS, "Nvm express over fabrics," https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1a-2021.07.12-Ratified.pdf, (Accessed 1 November 2023).

[29] NVMEXPRESS, "Nvm express rdma transport specification," https://nvmexpress.org/wp-content/uploads/NVM-Express-RDMA-Transport-Specification-1.0b-2022.10.04-Ratified.pdf, (Accessed 1 November 2023).

[30] NVMEXPRESS, "Nvm express specification," https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-2_0-2021.06.02-Ratified-5.pdf, (Accessed 1 November 2023).

[31] NVMEXPRESS, "Nvm express tcp transport specification," https://nvmexpress.org/wp-content/uploads/NVM-Express-TCP-Transport-Specification-1.0c-2022.10.03-Ratified.pdf, (Accessed 1 November 2023).

[32] G. F. Pfister, "An introduction to the infiniband architecture," High performance mass storage and parallel I/O, vol. 42, no. 617-632, p. 102, 2001.

[33] D. A. Quattrochi, N. S.-N. Lam, H.-L. Qiu, and W. Zhao, "Image characterization and modeling system (icams): a geographic information system for the characterization and modeling of multiscale remote sensing data," in Scale in remote sensing and GIS. Routledge, 2023, pp. 295–307.

[34] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia, "A remote direct memory access protocol specification," Tech. Rep., 2007.

[35] Resnics, "Stargate-f1000-sn," http://www.resnics.com/product/netcard, (Accessed 1 November 2023).

[36] A. Sanaeifar, M. L. Guindo, A. Bakhshipour, H. Fazayeli, X. Li, and C. Yang, "Advancing precision agriculture: The potential of deep learning for cereal plant head detection," Computers and Electronics in Agriculture, vol. 209, p. 107875, 2023.

[37] F. D. E. Team, "Rocksdb: A persistent key-value store for flash and ram storage," https://github.com/facebook/rocksdb, (Accessed 20 March 2024).

[38] H. Thai, K. Le, and N. L. Nguyen, "Towards sustainable agriculture: A lightweight hybrid model and cloud-based collection of datasets for efficient leaf disease detection," Future Generation Computer Systems, vol. 148, pp. 488–500, 2023.

[39] S. M. Trimberger, Field-programmable gate array technology. Springer Science & Business Media, 2012.

[40] T. Wang, Y. Liang, X. Shen, X. Zheng, A. Mahmood, and Q. Z. Sheng, "Edge computing and sensor-cloud: Overview, solutions, and directions," ACM Computing Surveys, vol. 55, no. 13s, pp. 1–37, 2023.

[41] Z. Wang, H. Huang, J. Zhang, F. Wu, and G. Alonso, "Fpganic: An fpga-based versatile 100gb smartnic for gpus," in 2022 USENIX Annual Technical Conference (USENIX ATC 22), 2022, pp. 967–986.

[42] W. Zhang, Z. Shi, Z. Liao, Y. Li, Y. Du, Y. Wu, F. Wang, and D. Feng, "Graph3po: A temporal graph data processing method for latency qos guarantee in object cloud storage system," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2023, pp. 1–16.

[43] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–15.

DIAS: Distance-based Attention Sparsity for Ultra-Long-Sequence Transformer with Tree-like Processing-in-Memory Architecture

Zekai Chen*, Yiming Chen*, Teng Wan, Tianyi Yu, Yu Wang, Huazhong Yang and Xueqing Li
Department of Electronic Engineering, LEFT/BNRist/SKLSNC, Tsinghua University, Beijing, China
xueqingli@tsinghua.edu.cn

***Abstract*—Long-context inference has become a central focus in recent self-regressive Transformer research. However, challenges still remain in performing decode stage due to the memory bandwidth bottleneck of attention mechanisms and the substantial memory overhead associated with KV cache. Although attention sparsity has been proposed as a potential solution, conventional sparsity methods that rely on heuristic algorithms often suffer from accuracy degradation when applied to ultra-long sequences.**

To break through the dilemma between accuracy-performance and bandwidth-capacity, this work proposes DIAS, a distance-based irregular attention sparsity approach with processing-in-memory (PIM) architecture. DIAS employs approximate top-K attention (AKAttention) scores through graph-based search to improve inference efficiency while maintaining accuracy. Furthermore, a scalable tree-like PIM (TreePIM) architecture is introduced to achieve both memory capacity and bandwidth improvement by isolating enormous memory access for KV cache into the PIM units. Evaluations on various configurations of DIAS for Longbench with Llama3-405B models with 1M sequence length show up to 75 times speedup compared with the state-of-the-art LLM accelerator, with accuracy drop of less than 1%.

***Index Terms*—AI and Machine Learning, Architecture & System Design**

I. Introduction

Large Language Models (LLMs) built on the attention-based Transformer architecture [1] have demonstrated remarkable potential and achieved significant advancements across computer vision (CV) [2], natural language processing (NLP) [3], and automatic control [4]. Especially, emerging applications, such as long-term agency, retrieval-augmented generation (RAG) [6], and multimodal processing [7], underscores the growing demand for long-context LLM inference, with sequence length requirements increasing from approximately 1K to beyond 10M.

However, it is challenging to deploy Transformer-based LLM in long-context scenarios in terms of memory bandwidth and capacity. As shown in Figure 1 (a), GB-level KV cache has to be totally loaded to generate the next token, which results in serious memory bandwidth and memory capacity bottleneck. What's worse, the memory bandwidth and capacity is a dilemma in physic. Existing scale-out scheme for expansion is severely constrained due to inter-chip communication. which incurs significant performance degradation.

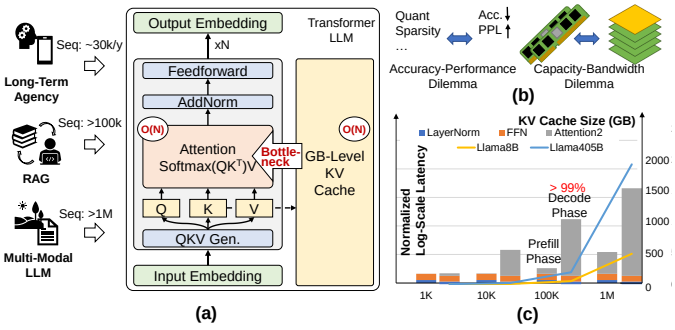


Fig. 1: The long-context LLM inference challenges illustration of (a) memory bandwidth bottleneck due to KV cache, (b) accuracy-performance and capacity-bandwidth dilemma, and (c) the latency breakdown and memory capacity trends.

Fortunately, various solutions for the long-context LLM inference have been proposed, including quantization [8] [9], speculative decoding [10] [11], alternative attention mechanism [12] [13], and attention sparsity [14] [15]. Quantization, while compatible with other techniques, inevitably leads to accuracy degradation, as shown in Figure 1 (b), and can result in significant fine-tuned knowledge loss in industrial LLMs. Furthermore, speculative decoding can accelerate inference by utilizing the multiple consecutive inference results of a small model. But this approach highly depends on the capacity of small models and could increase the decoding energy cost and delay. Alternative model architectures, such as Mamba [33] and TTT [34], have been proposed as potential replacements for Transformers. However, these emerging structures introduce significant information loss in long-text scenarios due to single hidden state compression. Thus far, sparsity exploitation remains the most promising and effective approach for addressing the memory-bond in long-context inference.

Numerous works have attempted to leverage the sparsity in attention computation shown in Figure 2. Recent work ToPick [14] could early eliminate the K cache retrieving by calculating the probability of attention score. However, the compression rates is still limited due to the pruning method. What's more, heuristic sparse methods like Window Attention [15], Stream LLM [16], H_2O [17] are proposed to discard part of KV cache based on different importance prediction strategy. For instance,

* : Both authors contributed equally to this research.

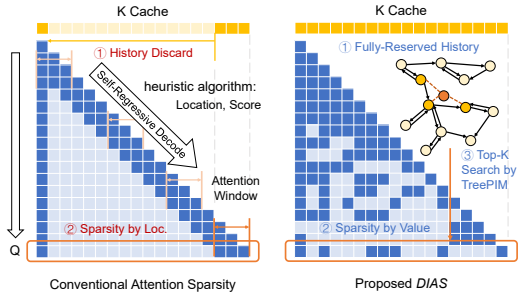


Fig. 2: Illustration of the bottlenecks of conventional attention sparsity [15] [16] and the innovations of proposed DIAS.

Window Attention only utilizes local context windows, which completely loss the history information. Other methods, such as infLLM [18], attempt to offload KV cache from GPU memory to CPU memory by selecting representative keys for different groups and retrive KV cache with look up table . Although these methods claim efficiency, there remains significant accuracy drop in ultra-long-context scenarios.

To tackle the aforementioned problems, we proposed DIAS, a distance-based irregular attention sparsity for long-sequence P/D separated transformer acceleration. As a software-hardware co-design solution, DIAS focuses on the bottleneck in decode phase of self-regressive LLM to break through the accuracy-performance and bandwidth-capacity dilemma. The innovations includes: (1) a spatially structured and dynamic updated representation of KV cache, (2) an effective attention sparsity computing method based on approximate top-K value to achieve performance improvement with accuracy maintaining, and (3) TreePIM architecture based on switches to achieve high bandwidth with large capacity. Notably, this work is compatible with recent researches on attention modifications, such as decoder-decoder model [20], razor attention [21], making DIAS applicable widely. Additionally, this method could be further researched and utilized in ultra-long-context scenarios to tackle attention noise problems and the bottleneck of model pretraining length, both of which have emerged as central challenges in the field of LLM algorithms

The major contributions of this paper are as follows:

- A distance-based irregular attention sparsity algorithm AKAttention in attention computing with almost no accuracy drop to address the memory bandwidth bottleneck due to KV cache retrieve.
- An effective hardware architecture TreePIM is presented for balancing the gap between memory bandwidth and capacity by processing-in-memory to address the dilemma between memory capacity and bandwidth.
- Rich experiment results shows that the proposed DIAS achieves about x25.3-171 speedup and x1.7-5.0 energy efficiency compared with conventional works.

II. Background

A. Attention Mechanism

The key contribution of Transformer [1] is the self-attention mechanism, which enables the model to capture long-range dependencies between elements in a sequence, thereby enhancing its ability to model complex patterns.

As is shown in Figure 1, in a typical Transformer layer, each input token undergoes a series of transformations through multiple sub-layers, the most important of which is the multi-head self-attention mechanism. For a given input sequence of tokens, represented as embedding $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, self-attention model computes as follows:

$$O = softmax(\frac{XW_Q \cdot (K_{Cache}, XW_K)}{\sqrt{d_k}}) \cdot (V_{Cache}, XW_V)$$

It could be found that attention computing is a typically memory-intensive application. The memory access and computation complexity grows linearly with sequence length, which results in serious performance bottlenecks.

B. ANNS Algorithms

Approximate nearest neighbor search (ANNS) algorithms are essential for accelerating large-scale search tasks by efficiently identifying approximate neighbors in high-dimensional spaces. These methods are particularly valuable in applications like similarity search and are widely used in large-scale machine learning systems, such as recommendation system[li2019approximate].

ANNS algorithms have been extensively researched, with each approach offering distinct advantages suited to different scenarios. Well-established methods such as KD-Tree [23], Product Quantization (PQ) [24], and Inverted File (IVF) [25] are widely used for efficient nearest neighbor search. Graph-based methods, however, are considered state-of-the-art for ANNS due to their high efficiency and accuracy. Hierarchical Navigable Small World (HNSW) graphs [26], for example, is one of the most effective and widely used graph-based ANNS methods. More recently, CAGRA [27] has emerged as an advancement over HNSW. CAGRA is designed to leverage the parallel processing capabilities of GPUs, achieving substantial improvements in speed and scalability compared to traditional CPU-based approaches.

C. Processing-in-Memory (PIM) Architectures

Processing-in-Memory (PIM) architectures have become a key solution for reducing data movement in data-intensive tasks. In Transformer models, PIM approaches like ReTransformer [28] use ReRAM-based memory to speed up attention calculations, while TransPIM [30] combines PIM with High Bandwidth Memory (HBM) for faster processing. More recently, AttAcc [36] and NeuPIMS [37] have delved into architecture design and in-memory parallelism. Furthermore, SpecPIM [38] leverages a PIM architecture to speed up speculative decoding in language models.

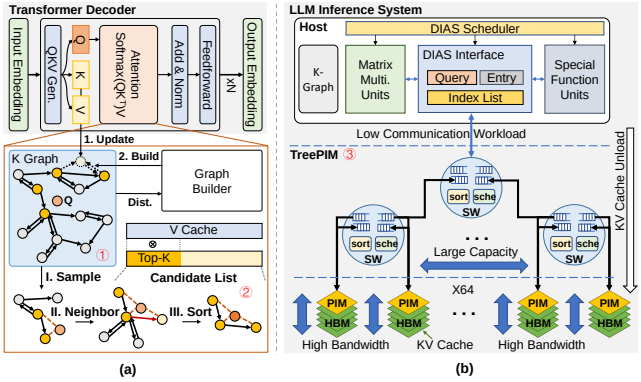


Fig. 3: Proposed DIAS: (a) irregular attention sparsity based on top-K score search algorithm, and (b) TreePIM architecture for graph-based sparsity attention acceleration.

For Approximate Nearest Neighbor Search (ANNS), PIM architectures like NDSearch [29] and Pyramid [31] leverage near-data processing to optimize graph traversal, introducing different designs in dataflow, hierarchy and memory management methods. These PIM designs are critical for scaling Transformer models and ANNS tasks in large applications, providing high throughput and reduced energy costs.

III. Proposed Work

A. Overall Architecture DIAS

The overall architecture is depicted as Figure 3. The P/D separation technique is adopted in this work to focus on the decode phase of self-regressive LLM. This work proposed (1) graph-structure dynamic-updated K Cache, (2) irregular attention sparsity based on top-K attention score search, and (3) tree-like PIM architecture for isolating the high data communication into the separated PIM units to achieve both large memory capacity and high bandwidth.

Firstly, a K-graph is built based on K Cache during the prefill stage on the host device. While the large KV cache is unloaded to the PIM device. At the costly decode stage, new query, key, and value vectors are computed on host by matrix multiply unit (MMU). Then, DIAS scheduler would iteratively generate and dispatch query batches based on the K-graph. Switch units (SW) are organized as a binary tree to support scalable capability and will route query batches to corresponding PIM units. The PIM units effectively compute the inner production benefited with high parallelism and DRAM bandwidth, while inner product results will be gathered to Host by SWs and sorted with parallel bitonic sort algorithm. After several iterations, Top-K attention score of $softmax(QK^T)$ will be calculated. and finally be scattered to the PIM unit to complete the sparse attention computation with V Cache.

Moreover, graph update is also performed at the same time. Graph update is performed on host and KV vectors are uniformly assigned to PIM units to keep TreePIM balance and highly utilized. Both operations are time-overlapped and hidden from the attention computation part.

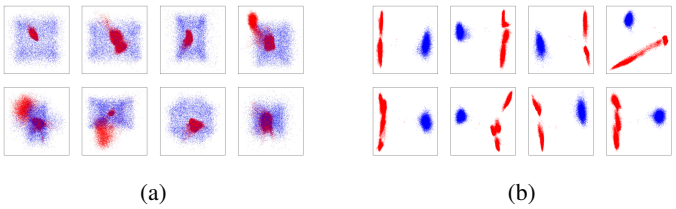


Fig. 4: Illustration of distribution on (a) separated dims, and (b) PCA with red refer to query and blue refer to key.

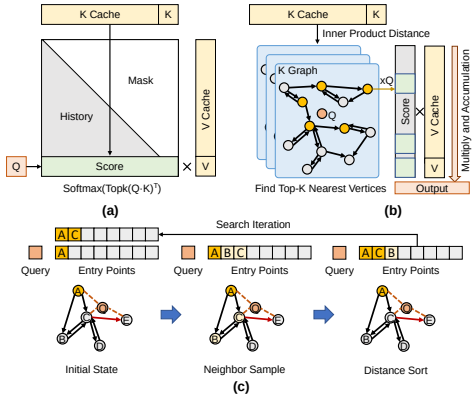


Fig. 5: Distance-based sparsity attention computing (a) principle, (b) illustration of the method, and (c) workflow.

The proposed DIAS achieves an $O(1)$ complexity and constant memory access when generating a token in Transformer-based LLMs. In contrast, conventional solutions generate one token in the conventional Attention layer with a complexity of $O(L)$ and increasing memory access, making the total complexity of generation $O(L^2)$, where L represents the sequence length.

B. AKAttention Algorithm

1) Theoretical Analysis: Figure 8 visualizes Llama3.1 as a sample of query, and key distributions. It shows disperse key vectors and centralized query vectors, which motivates the sparsity attention by ignoring the unrelated key cache. In contrast to conventional given keys pruning methods, this work proposes a graph-based technique to search the top-K related key cache for each query. Considering the attention score $S = softmax(QK^T)$, the relativity between query and key cache could be represented as the inner product distance.

As shown in Figure 5(b), with the transitivity of inner product, a graph built by key cache (K-Graph) could be constructed with inner product distance to represent the relativity between tokens and perform efficient attention computation with graph based ANNS algorithm integrated in work flow.

2) Initialization: The K-graphs of AKAttention is initialized in the prefill stage. Redundant edges in K-graph is pruned motivated by CAGRA [27] using detourable route counts, making each vertices with a fixed out-degree to fully utilize the parallelism. The entry points are set to recent visited or

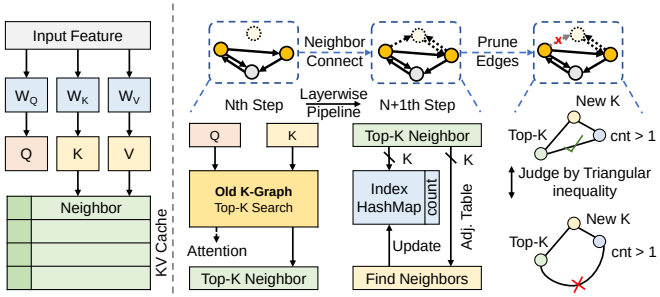


Fig. 6: Dynamic K-Graph update method in the proposed AKAttention.

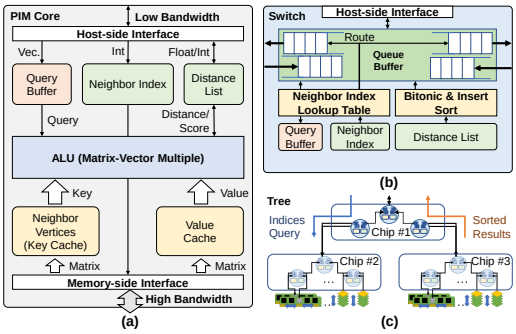


Fig. 7: Circuits design of (a) processing-in-memory unit for effectively sparsity attention computing, and (b) switch unit for (c) dynamic scalable capability.

updated vertices cached on host, while generated KV cache are uniformly offload to TreePIM as interleave or random mode.

3) AKAttention Workflow: Figure 5(c) shows how AKAttention finds the top-K attention scores after the new key, value, and query generation:

At the first step of iteration. AKAttention computes the inner product between the entry points of K cache and the new query. The results will be sorted and the top-K largest keys will be reserved. At the second step of iteration, the neighbors of the reserved keys in the K-graph is sampled from key graph as the new entry points.

After reaching the maximum number of iterations, AKAttention will compute the attention scores by softmax the inner production results and multiply with the corresponding value cache to generate the final attention computing results.

Notably, the smallest value in top-K queue is cached in the host to prune useless candidates and their inner product result.

4) K-Graph Dynamic Update: The dynamic update method for K-graph is illustrated as Figure 6.

During the decode phase, the new key in K-graph is firstly connected to top-K neighbors, which is searched together with the new query in the AKAttention. Then detourable routes are counted using the same criteria in proxy graph initialization and edges with the most detourable routes are pruned. With index hash map, this process is performed $O(1)$ complexity, which could be fully parallel among different layers.

The detourable criteria used in AKAttention is: $(e_{X\to Z}, e_{Z\to Y})$is a detourable route if $\max(w_{X\to Z}, w_{Z\to Y}) < w_{X\to Y}$

C. TreePIM Architecture

1) TreePIM Structure: The proposed AKAttention provides an approach to maximum utilizing the sparsity in attention computing. Furthermore, an processing-in-memory architecture TreePIM is proposed to cope with the dilemma between the memory bandwidth and memory capacity.

As depicted in Figure 7, TreePIM is a tree-like scalable PIM architecture utilizing the relatively small communication cost of sending indices and inner production results in AKAttention to achieve ultra high memory bandwidth with scalable large memory capacity in AKAttention computation. TreePIM is composed by PIM devices and switches while switches connect in a tree-like mode while the PIM devices are organized as leaves.

The low-cost indices and inner production results is passed through the switch tree, where the high-bandwidth key-query inner production are performed parallel in PIM devices, boosting the total memory bandwidth in the system level.

Also, TreePIM is internally flexible with tree-like topology, meeting the large variation of memory budget in different inference scenarios ranging from 8B Model 10k tokens to 405B Model with 10M tokens.

2) PIM Unit: The PIM unit used in this work shows as Figure 7(a). As only inner product and weighted sum are involved in PIM computation, the proposed PIM unit is relatively simple, resulting in small area and energy overhead.

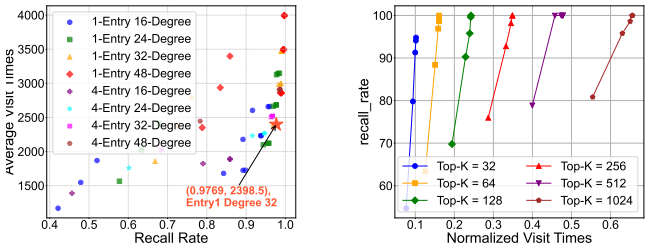
When the PIM unit receives query vectors and entry point indices, query vectors are cached in the query buffer and reused in the following top-K search iterations. PIM utilizes high bandwidth communication to read the KV cache vectors from memory and compute inner production, then send the lightweight result back through switches.

3) Switch: The switch unit shows as Figure 7(b), consisting of lookup tables and sort units. When a switch unit receives packages from its father node, it redirects the package to left or right according to the lookup tables. The package will be passed to both left and right if the lookup tables miss. The lookup table is dynamically updated using the return results from its left son and right son.

The sort part works when results are returned from left son and right son. In switch, high parallelism sort is performed utilizing the bitonic sort algorithm. Insert sort is performed if a switch is connected to PIM unit to make a bitonic sequence.

4) Workload Balance: To deal with different scenarios, '-S', '-L', '-LL' memory manage modes are proposed.

S mode is designed for relatively small KV cache. In the S mode, graph structure and location of each key are stored on host. In this scenario, each package is assigned with a certain target dram and switch could transfer based on it. And sampling is conducted on host which alleviate the communication cost between switch.



(a) (b)

Fig. 8: (a)Top-K search average retrieved vertices and corresponding recall rate on different numbers of search entry and graph degrees (b) Analysis of recall rate on different iteration times and top-K setup.

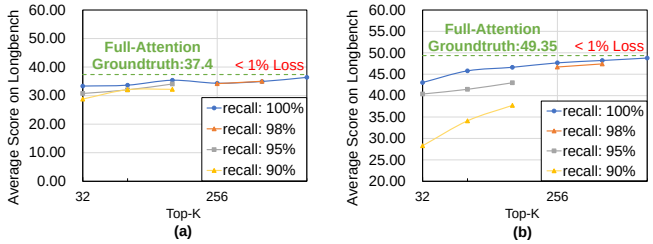


Fig. 9: Performance variation with top-K and recall rate of Longbench on (a) Mistral-7B, and (b) Llama3.1-8B.

L mode is designed for larger KV cache size. In L mode graph structure is stored in TreePIM, and retrieved each time with indices and inner product results. Only the neighbor lists of updated top-K vertices and top-K vertices for new generated key vector are stored on host, which will be used for sampling and update in next layer computation respectively.

In LL mode, location table of each key are also abandoned to alleviate memory cost. Lookup table is used to instruct transfer process in switch and candidate broadcast could lead to more transfer cost.

Notably, KV pairs are uniformly assigned to the DRAM devices in TreePIM with interleave or random method. This mechanism ensures load balancing and system efficiency.

IV. Experiment

A. Experiment Setup

1) Software setup: PyTorch is used as the Transformer-based LLM framework for the tasks inference of the proposed DIAS architecture on 8 NVIDIA A100. The proposed AKAttention is adopted as the attention sparsity mechanism. The LLM structure used in experiments is the widely used Llama3.1 and Mistral. All models are well pretrained large model. To evaluate the advantages of AKAttention in long-context scenario, the Longbench [32] is selected as benchmark, which includes 21 datasets in different scenarios. The datasets contains prompts with average length approximate to 10K, and maximum length up to 40k, which is fairly evaluated

TABLE I: System-Level Performance Comparison

	DIAS (This Work)	Vanilla	Window Attention	ToPick DAC'24
Config	4 * 16 32GB HBM	HBM-PIM		16GB HBM2
Model&Seq Len	Llama3.1-405B&1M			
Acc. Drop	<1%	Full	>30%	Negligible
Throughput (Token/min)	461.5	2.705	45.801	6.21
Energy Eff. (Token/kJ)	1.7	0.42	4.1	1.68
Speedup	**171.7**	1	16.9	2.3

the accuracy impact of different attention sparsity methods in varied long-sequence LLM application scenarios.

2) Hardware setup: The performance of PIM units and switch units in TreePIM architecture are based on DC simulations in a 28nm CMOS process. The memory and peripheral latency and energy consumption are obtained from commercial memory and bus standard, DDR, HBM, and PCIE. In the system-level experiment, two configurations of DIAS are used for Llama3.1-8B decoding from 100k tokens scenario and Llama3.1-405B decoding from 1M tokens scenario. In 100k-8B model scenario, DIAS decodes with top-1K tokens utilizing 8-PIM-units 3-level TreePIM managed in S mode. For the other scenario, DIAS decodes using top-3K tokens with 4 channel of 16-PIM-units 4-level TreePIM managed in L mode.

3) Baseline setup: In the system evaluation, vanilla is assigned with conventional attention mechanism and typical loop-reduced-based scale-out architecture. TreePIM-Only performs conventional attention mechanism in TreePIM architecture, and AKAttention-Only performs AKAttention with conventional architecture. DIAS denotes the proposed design combined with AKAttention and TreePIM. The software evaluation compares the accuracy on the proposed AKAttention of 128-top-K and 1k-top-K with previous attention sparsity methods: Streaming LLM [16], InfLLM [18], Infinite [35], H2O [17]. The system-level performance is compared with vanilla, slide window attention [35], and SOTA LLM accelerating architecture ToPick [14] utilizing retrive pruning.

B. Experiment results

1) Top-K retrieving analysis: Figure 8a shows the relationships between average number of retrieved vertices in searching and the corresponding recall rate in AKAttention of different configurations. It could be found that there is a significant increase in recall rate as out-degree of the K-graph grows, which is useful for improving the inference accuracy with attention sparsity. However, the average retrieved vertices, which represents the overhead of AKAttention also increased. On the other hand, 1 entry for 1 top-K out performs 4 entry in both recall rate and efficiency. Considering the trade-off between performance and accuracy, this work suggests the use of 1-Entry 32-Degree to achieve the best system-level throughput with negligible accuracy drop.

Further, analysis on the selection of top-K is discussed in Figure 8b. It shows that the proposed AKAttention in DIAS could achieve over 99% recall rate on most of the top-k based

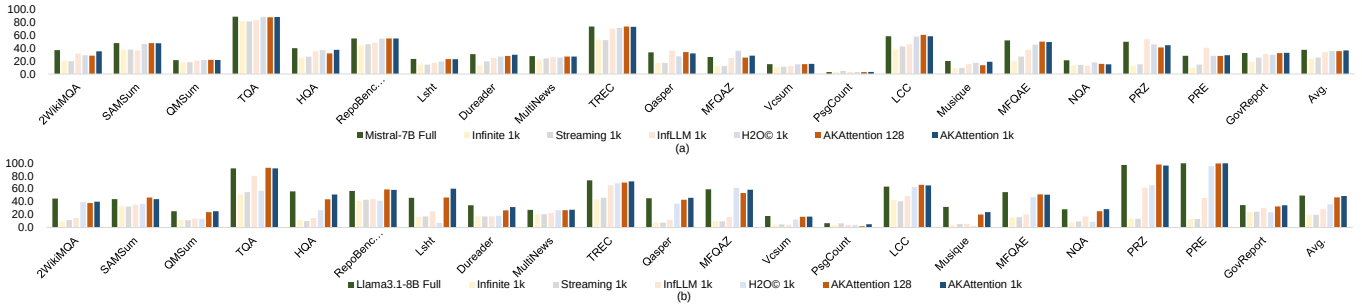


Fig. 10: Accuracy comparison between the proposed DIAS and conventional sparsity methods on Longbench with (a) Mistral-7B, and (b) Llama3.1-8B.

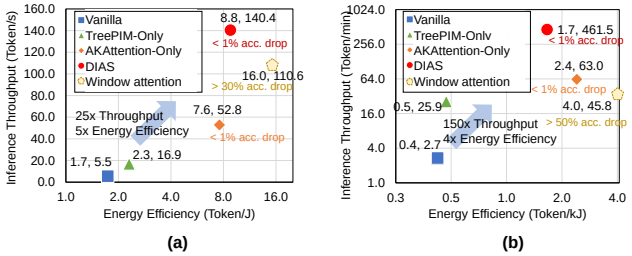


Fig. 11: System-level inference throughput and energy efficiency simulation results with (a) 100k seq len on Llama3.1-8B, and (b) 1M seq len on Llama3.1-405B.

on the previous configuration. In this work, the selection of top-K is a configurable parameter that depends mainly on the accuracy and performance needs of the tasks. As shown in Figure 9, an accuracy drop less than 1% could be achieved on both Mistral-7B and Llama3.1-7B, with a proper top-K.

2) End-to-end accuracy evaluation: Figure 10 shows the accuracy comparison between the proposed AKAttention in DIAS and the conventional sparsity methods on the long-context dataset Longbench. As shown in the figure, the proposed AKAttention achieves less than 1% accuracy drop or even better on most tasks in Longbench even with the top-128 setup configuration, which is much smaller than the other methods that could lose more than 30% of accuracy.

3) System-level performance comparison: Table. I shows the performance comparison with conventional architecture and the prior SOTA ToPick [14]. Despite extreme energy efficiency of Window Attention, its accuracy degradation in long-sequence scenarios is lethal. ToPick achieves about 2x efficiency improvement by employing prediction in K cache retrieving. However, the improvement is still limited due to the constraint of the possibility calculation.

Fig.11 visualize the performance comparison between various scheme. In 1M tokens Llama3.1-405B scenario, DIAS achieves about 170x speedup and 1.7x energy efficiency improvement. While in 100k tokens Llama3.1-8B scenario, DIAS achieves 25x speedup and 5x energy efficiency improvement. Compared with window attention algorithm of over 30% accuracy loss, the proposed DIAS effectively utilizes the sparsity in attention with less than 1% accuracy degradation. DIAS shows great potential on long-context LLM inference and could be compatible to many other techniques.

V. Overhead Discussion

Although this work achieves an extremely high throughput improvement using a Top-K based irregular sparse attention mechanism, a large number of inter-switch transfers become a new bottleneck. inter-switch transfers k cache indices with a relatively small amount of data. However, the bandwidth of switch units remains a bottleneck under the scalability constraint due to the low dimension of each header in the current large model architecture, which is only 128. There are several potential mitigation schemes for this problem, including expanding hidden size of each head, index list compression and packetized fetching of multiple headers. These will be further investigated in the future work.

VI. Conclusion

In this paper, a scalable, high-throughput PIM architecture DIAS for long-context LLM inference is proposed to deal with the memory bandwidth bottleneck in attention computing. DIAS utilizes approximate near neighbor search (ANNS) to find the top-K relative key cache with the given query, which greatly alleviates the memory bandwidth challenges. On the one hand, AKAttention only needs to fetch a small portion of the key cache, reducing the total communication volume. On the other hand, TreePIM isolates the high-bandwidth key retrieving into the PIM region with scalabilty. The communication inter-chip is reduced by hundreds times compared with conventional loop-reduced scale-out architecture. The system-level evaluation shows that the proposed DIAS achieves up to 75 times speedup compared with the SOTA LLM accelerator, with less than 1% accuracy degradation.

VII. Acknowledgment

This work was supported in part by National Key R&D Program of China (2024YFB3214500) and by NSFC under Grant U21B2030, Grant U24B6015, Grant 92264204 and Grant 62325405.

REFERENCES

[1] A. Vaswani, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[2] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[3] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[4] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, "Transformer networks for trajectory forecasting," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 10335–10342.

[5] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2022, pp. 27268–27286.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, D. Kiela, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[7] V. Gabeur, C. Sun, K. Alahari, and C. Schmid, "Multi-modal transformer for video retrieval," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV*, vol. 12349, 2020, pp. 214–229.

[8] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "Kvquant: Towards 10 million context length LLM inference with KV cache quantization," *arXiv preprint arXiv:2401.18079*, 2024.

[9] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, and X. Hu, "Kivi: A tuning-free asymmetric 2bit quantization for KV cache," *arXiv preprint arXiv:2402.02750*, 2024.

[10] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the International Conference on Machine Learning*, 2023, pp. 19274–19286.

[11] X. Liu, L. Hu, P. Bailis, A. Cheung, Z. Deng, I. Stoica, and H. Zhang, "Online speculative decoding," *arXiv preprint arXiv:2310.07177*, 2023.

[12] P. D. Letourneau, M. K. Singh, H. P. Cheng, S. Han, Y. Shi, D. Jones, et al., "PADRe: A unifying polynomial attention drop-in replacement for efficient vision transformer," *arXiv preprint arXiv:2407.11306*, 2024.

[13] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[14] J. Park, M. Kang, Y. Han, Y. G. Kim, J. Shin, and L. S. Kim, "Token-picker: Accelerating attention in text generation with minimized memory transfer via probability estimation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[15] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 10012–10022.

[16] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[17] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, et al., "H2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 34661–34710.

[18] C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, et al., "InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[19] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.

[20] Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, and F. Wei, "You only cache once: Decoder-decoder architectures for language models," *arXiv preprint arXiv:2405.05254*, 2024.

[21] H. Tang, Y. Lin, J. Lin, Q. Han, S. Hong, Y. Yao, and G. Wang, "RazorAttention: Efficient KV cache compression through retrieval heads," *arXiv preprint arXiv:2407.15891*, 2024.

[22] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475–1488, 2019.

[23] J. L. Bentley, "K-d trees for semidynamic point sets," in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, 1990, pp. 187–197.

[24] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2010.

[25] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys (CSUR)*, vol. 38, no. 2, pp. 6-es, 2006.

[26] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[27] H. Ootomo, A. Naruse, C. Nolet, R. Wang, T. Feher, and Y. Wang, "CAGRA: Highly parallel graph construction and approximate nearest neighbor search for GPUs," in *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 4236–4247.

[28] X. Yang, B. Yan, H. Li, and Y. Chen, "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–9.

[29] Y. Wang, S. Li, Q. Zheng, L. Song, Z. Li, A. Chang, and Y. Chen, "NDSEARCH: Accelerating graph-traversal-based approximate nearest neighbor search through near data processing," in *Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 368–381.

[30] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A memory-based acceleration via software-hardware co-design for transformer," in *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1071–1085.

[31] Z. Zhu, J. Liu, G. Dai, S. Zeng, B. Li, H. Yang, and Y. Wang, "Processing-in-hierarchical-memory architecture for billion-scale approximate nearest neighbor search," in *Proceedings of the 2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[32] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, and J. Li, "LongBench: A bilingual, multitask benchmark for long context understanding," *arXiv preprint arXiv:2308.14508*, 2023.

[33] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[34] Y. Sun, X. Li, K. Dalal, J. Xu, A. Vikram, G. Zhang, and C. Guestrin, "Learning to (learn at test time): RNNs with expressive hidden states," *arXiv preprint arXiv:2407.04620*, 2024.

[35] C. Han, Q. Wang, W. Xiong, Y. Chen, H. Ji, and S. Wang, "LM-Infinite: Simple on-the-fly length generalization for large language models," *CoRR*, vol. abs/2308.16137, 2023.

[36] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "AttAcc! Unleashing the power of PIM for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 2, 2024, pp. 103–119.

[37] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 3, 2024, pp. 722–737.

[38] C. Li, Z. Zhou, S. Zheng, J. Zhang, Y. Liang, and G. Sun, "SpecPIM: Accelerating speculative inference on PIM-enabled system via architecture-dataflow co-exploration," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 3, 2024, pp. 950–965.

A Novel Covert Timing Channel for Cloud FPGAs

Brian Udugama* †, Darshana Jayasinghe‡, Hassaan Saadat§, Aleksandar Ignjatovic† and Sri Parameswaran‡
*CSIRO Space and Astronomy, PO Box 76, Epping NSW 1710, Australia.
†School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.
‡School of Electrical and Computer Engineering, University of Sydney, Sydney, Australia.
§School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, Australia.

***Abstract*—This paper presents a novel covert timing channel (CTC) that enables a malicious entity to exfiltrate data from a benign cloud FPGA user without requiring dedicated outgoing messages from the cloud FPGA, minimizing the detection risk by both the victim and the cloud service provider. The proposed CTC exploits the handshake signals of the Advanced eXtensible Interface (AXI) protocol and inter-packet delay of the Internet to establish the CTC from a cloud Field-Programmable Gate Array (FPGA) to an off-cloud computer. This paper analyzes the bit-error rate (BER) of the AXI-based CTC under varying conditions and demonstrates its effectiveness in truly enabling remote power analysis attacks on cloud services, such as Amazon Web Services Elastic Compute Cloud (AWS EC2). The proposed CTC achieves a BER as low as 0.01988%.**

***Index Terms*—Covert Timing Channels, Cloud FPGA, AXI Protocol, Remote Power Analysis Attacks.**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are increasingly deployed in cloud environments, vitalizing the importance of investigating remotely deployable hardware-level security threats. Cloud FPGA services, such as Amazon Web Services Elastic Compute Cloud (AWS EC2) F1 instances, provide users with FPGA resources in a virtual environment, restricting direct user interaction with peripherals for consistency and security. In AWS EC2 F1 instances, the Advanced eXtensible Interface (AXI) protocol [1] manages communication between the AWS-provided shell [2], which controls peripherals, and the user-generated custom logic circuit.

This paper introduces, for the first time, a covert timing channel (CTC) [3] that enables data exfiltration from a cloud FPGA user, without requiring dedicated outgoing messages from the cloud FPGA. The proposed CTC is composed of two phases. In the initial phase, the CTC leverages the handshake signals of the AXI protocol to covertly transfer data from a cloud FPGA to an off-FPGA resource. In the second phase, it exploits inter-packet delays over the Internet to continue the covert transfer from the cloud resource to an off-cloud computer, creating a CTC from the cloud FPGA to the off-cloud computer. To demonstrate the threat posed by the proposed CTC, power consumption data gathered from a cryptographic circuit deployed on a cloud FPGA are exfiltrated to an off-cloud computer, where it is analyzed to extract the secret key of the cryptographic circuit. Moreover, this paper evaluates the efficacy of the CTC by analyzing the bit-error rate (BER) of the proposed CTC under varying design and operational conditions. Hence, the CTC presented in this paper poses a significant security risk to benign cloud FPGA users, exhorting users to take preventive measures.

Paper Contributions:

- Presents the first covert communication channel to exfiltrate data from a cloud FPGA without requiring explicit outgoing messages from the FPGA.
- Demonstrates the possibility of exploiting the handshake signals of the AXI protocol to establish a CTC on an FPGA.
- Signifies the importance of CTC to fully realize a remote power analysis (RPA) attack [4], emphasizing the implications for cloud FPGAs and their users.
- With substantial experimentation, analyzes the effectiveness of the CTC with respect to BER under design and operating condition changes.

Paper Organization: The rest of this paper is organized as follows. Section II provides key background knowledge for understanding this paper, while Section III discusses related works from the literature. The assumed threat model is detailed in Section IV. Section V presents a performance and effectiveness evaluation of the proposed CTC while the subsequent Section VI presents the design of signal processing techniques associated with the CTC. Section VII concludes this paper.

II. Background

Architecture of AWS EC2 F1 Instances: AWS EC2 F1 instances allow the deployment of FPGA-based hardware accelerators on cloud FPGAs. Each AWS EC2 F1 instance includes virtual central processing units (vCPUs) running a general-purpose operating system and FPGA cards implementing user-defined custom hardware accelerators. Configuring an FPGA on an AWS EC2 F1 instance requires loading an Amazon FPGA Image (AFI) onto the FPGA. First, the users are required to compile the custom logic circuit to an intermediate design checkpoint (DCP), and then submit the DCP to AWS to perform design rule checks (DRCs). A DCP that passes all the DRCs, conducted by AWS, is transformed into an AFI by wrapping the custom logic circuit of the DCP with an AWS-designed shell [2]. The shell implements the interfaces for FPGA external peripherals, PCIe, DRAM, and interrupts [2]. User interactions with PCIe and DRAM are to be conducted through the shell interfaces, which are implemented using the AXI protocol [2].

AXI Protocol: AXI protocol is an on-chip interconnect specification [1] that utilizes five unidirectional channels, namely Write Address (AW), Write Data (W), Write Response (B), Read Address (AR) and Read Data (R). Each AXI channel uses a pair of handshake signals to indicate the readiness of the endpoints for a data exchange. The source asserts the VALID signal when it has loaded valid data onto the AXI connection. The READY signal is asserted by the destination when the destination is ready to accept incoming data. In this manner, the handshake allows both endpoints to control the rate of data exchange on an AXI channel and thereby control the rate of data exchange on the AXI connection.

In the AXI protocol, a transfer refers to a sequence of signals that perform a data exchange with a single handshake. A transaction refers to a sequence of transfers that begins with an address transfer, followed by one or more data transfers. In a write transaction, a response transfer follows all the data transfers.

Correlation Power Analysis (CPA): CPA is a statistical method to analyze power measurements to extract the secret key of a cryptographic module [5]. In CPA, the attacker partitions the secret

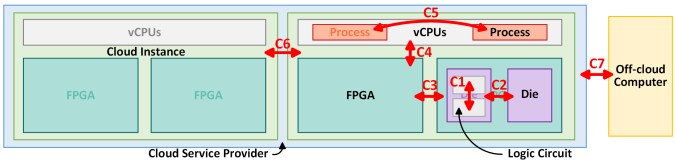


Fig. 1. Covert Communication Channels in Cloud Context

key into bytes, models the power consumption for each byte guess and calculates correlation between observed and modeled power consumption for each one of the byte guesses [6]. With sufficient observations, the correlation factor of the correct guess stands out, revealing the secret key.

Low-Density Parity-Check (LDPC) Codes: LDPC is a scheme of error-correcting codes designed to recover errors caused by transmission over noisy channels [7]. An LDPC code is defined by a parity-check matrix, typically generated randomly under certain constraints. When decoding using an LDPC code, the parity-check matrix is used to iteratively refine an initial guess of the error-correct output for a predefined number of times.

III. Related Work

The schematic in Fig. 1 depicts various covert communication channels implementable in a cloud service hosting FPGAs. The bidirectional arrow labeled C1, in Fig. 1, represents a covert communication channel between two disjoint logic circuits on an FPGA die within a cloud FPGA. A covert communication channel between two dies, also known as Super Logic Regions (SLRs), within an FPGA is represented by the bidirectional arrow labeled C2, in Fig. 1. Similarly, the bidirectional arrows labeled C3, C5 and C6, in Fig. 1, correspondingly illustrate covert communication channels between two FPGAs within a cloud instance, two vCPUs within a cloud instance and two cloud instances within a cloud service.

The covert communication channels represented by C4 and C7, in Fig. 1, are fundamentally different from the other covert communication channels due to the asymmetry of the endpoints. The upward C4 covert communication channel, as illustrated in Fig. 1, requires a transmitter on the FPGA and a receiver on vCPUs, while the downward covert communication channel requires the reversed endpoint configuration. Thus, a covert communication channel that is designed to infiltrate data to a cloud FPGA fails to exfiltrate data from a cloud FPGA. In C7, the leftward covert communication channel has the transmitter implemented on an off-cloud computer and the receiver implemented within the cloud service provider. Conversely, the rightward C7 covert communication channel features a transmitter within the cloud service provider and a receiver on an off-cloud computer.

Giechaskiel et al. [8] demonstrated a covert communication channel exploiting crosstalk between adjacent long wires in an FPGA. A malicious entity capable of routing a long wire next to a lengthy piece of wire belonging to an unsuspecting victim residing on an FPGA could exfiltrate data from the victim by using crosstalk-induced signal propagation delay variations [8]. This methodology [8] enables covert communication between two logically isolated circuits deployed on an FPGA. Gnad et al. [9] presented a voltage-based covert communication channel that exploits the power delivery network (PDN) of the FPGA. In this method [9], the malicious user deploys power wasters to modulate data into voltage fluctuations that occur on an FPGA and capture the voltage fluctuations using a logically isolated circuit. Both of the above methods [8], [9] illustrate implementations of the C1-labeled covert communication channel in Fig. 1.

Using voltage fluctuations that traverse the PDN of an FPGA, Giechaskiel et al. [10] established a covert communication channel between the SLRs of an FPGA. The authors [10] demonstrated the deployment of the covert communication channel in Amazon and Huawei cloud computing services, achieving a minimum bandwidth of 4.6 Mbps and an accuracy over 97.6%. Covert communication channels with similar capabilities are represented by C2 in Fig. 1.

Giechaskiel et al. [11] further demonstrated a methodology to exploit a shared power supply unit to construct a covert communication channel capable of infiltrating data into a cloud FPGA from a central processing unit (CPU), a graphics processing unit (GPU) or another FPGA. The authors [11] modulated the data as voltage fluctuations induced by power-wasting techniques, and demonstrated that induced voltage fluctuations are observable from an FPGA sharing the power supply unit with the component inducing voltage fluctuations. As per Fig. 1, the FPGA to FPGA covert communication channel [11] fits category C3, the transmitter being a power waster circuit constructed on an FPGA and the receiver being a voltage sensing circuit constructed on another FPGA. The CPU to FPGA covert communication channel [11] implements the C4 covert communication channel in the downward direction, as shown in Fig. 1, where CPUs adjust the load, thereby the instantaneous power, to transmit information while voltage sensing circuits on FPGAs measure the voltage fluctuations to receive covertly transmitted information.

Over several decades, using response time as a covert communication channel has been investigated by researchers. In 1987, Girling [12] demonstrated the use of inter-packet delay of a local area network (LAN) to establish a covert communication channel; a concept later applied in software-defined networks (SDNs) [13] and the Internet [14]. Lee et al. [14] demonstrated a covert communication channel spanning several thousand miles over the Internet, achieving 81 kbps bandwidth and less than 10% BER using a Software-defined Network Interface Card (SoNIC) [15]. In FPGA-equipped cloud instances, a SoNIC can be constructed within the FPGA, which can be exploited as proposed by Lee et al. [14] to implement C7 in Fig. 1. Ristenpart et al. [16] and Pessl et al. [17] have proposed techniques that implement the covert communication channel labeled C6, in Fig. 1. Ristenpart et al. [16] exploited the behavior of caches, while Pessl et al. [17] exploited the addressing mechanism used in Dynamic Random Access Memory (DRAM) to exfiltrate data from one cloud instance to another cloud instance that reside on the same physical computer. Alagappan et al. [18] developed a covert communication channel between CPU processes, represented by C5 in Fig. 1, utilizing power governors of a CPU to covertly modulate data into the CPU operating frequency, achieving up to 20 bps.

In the literature, covert communication channels that implement the upward, as illustrated in Fig. 1, covert communication channel C4 (FPGA to vCPU) are scarce and often tied to specific attack scenarios. Purnal et al. [19] exploited the behavior of CPU caches to construct an FPGA to the CPU covert communication channel. This method [19] uses a malicious process, running on the CPU, to prime the caches by loading specific symbols, which the FPGA then selectively overwrites based on the information to be exfiltrated. Finally, the malicious process measures the read times, revealing the symbols overwritten by the FPGA. The authors [19] achieved a 264 kbps bandwidth with a 1. 43% BER and also claimed that their proposition is less detectable to runtime analysis as the FPGA design belonging to the malicious user can rapidly overwrite symbols written by the malicious process, without affecting the runtime of a process.

Despite the covert communication channel by Purnal et al. [19]

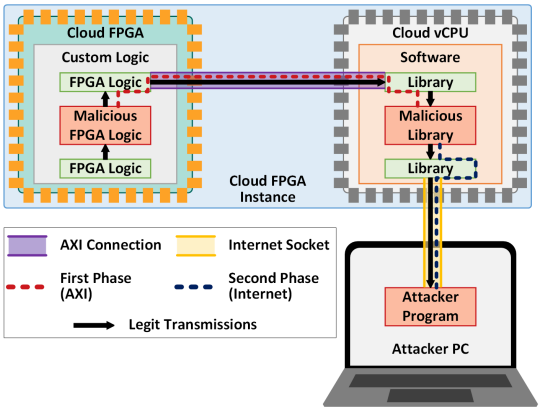


Fig. 2. Threat Model

is capable of staying undercover within the software domain, it lacks effective concealment on the hardware. The authors [19] in their threat model, assume a malicious user controls at least one CPU core and a secondary device such as an FPGA, dispatching operations to both software and hardware while sharing memory between them. In cloud FPGA instances, where AXI interfaces are limited, overt transmission over AXI poses the risk of exposing the covert communication channel to the cloud service provider or other users of the cloud FPGA instance. The covert communication channel from FPGA to CPU proposed by Weissman et al. [20] also relies on cache exploitation and data transfers between the FPGA and the memory, making the covert communication channel similarly vulnerable to detection.

Establishing effective FPGA to vCPU and cloud to off-cloud covert communication channels is pivotal for malicious entities residing on cloud FPGAs, to exfiltrate data from the cloud FPGA to an off-cloud resource. The existing literature lacks effective covert communication channels for establishing the FPGA to vCPU connection in the context of cloud FPGAs, with existing techniques [19], [20] exhibiting notably specific attack scenarios. Furthermore, the literature lacks effectiveness assessment of covert timing channels (CTCs) over the Internet, in the context of cloud. This paper, for the first time, proposes a CTC effective for exfiltrating data from a cloud FPGA to an off-cloud computer. In the first phase, the proposed CTC exploits the VALID and READY handshake signals of the AXI protocol to covertly transmit data from the FPGA to the vCPUs of a cloud FPGA instance. In the second phase, the proposed CTC exploits the inter-packet delay over the Internet to covertly transmit data from the cloud to an off-cloud computer.

IV. Threat Model

The threat model in this paper focuses on a cloud FPGA instance, such as an AWS EC2 F1 instance, targeted by an attacker residing external to the cloud service. As depicted in Fig. 2, the cloud FPGA instance hosts a system with a malicious hardware-software suite injected by the attacker. The malicious hardware-software suite consists of a logic circuit (labeled Malicious FPGA Logic) integrated into the custom logic of an unsuspecting victim, and a software library (labeled Malicious Library) invoked by the software process executing on cloud vCPUs. The malicious hardware-software suite aims to covertly transfer data from the cloud FPGA to the cloud vCPUs and thereafter to the off-cloud personal computer (PC) controlled by the attacker, by implementing the upward C4 covert communication channel of Fig. 1 and the rightward C7 covert communication channel of Fig. 1, respectively.

Complex FPGA logic circuits often use third-party intellectual property (IP) cores like encryption engines, audio/visual pipelines and arithmetic units [21]. With cloud FPGA instances offering powerful vCPUs, users can offload vCPU-efficient tasks onto vCPUs, paving the way for third-party hardware-software IPs to become prevalent in the near future. To infiltrate the infrastructure of an unsuspecting victim, the malicious hardware-software suite disguises itself as a useful hardware-software IP by providing services verified through test vectors. The functionality of the Malicious FPGA Logic and the Malicious Library is verified individually by supplying test vectors and comparing the outputs with the expected algorithmic output. The Malicious FPGA Logic does not necessarily have a direct connection to the Malicious Library, blocking the opportunity for output manipulation. However, the Malicious FPGA Logic can alter the time it takes to validate the data transferred between the FPGA and vCPU sides in the cloud FPGA instance.

In comparison to existing work, this paper assumes a holistic and realistic attack scenario with broader applicability to cloud FPGAs. An attacker with access to a small logic area on the cloud FPGA and a minimal portion of runtime on vCPUs can deploy both phases of the proposed CTC, without requiring the ability to share memory between software and hardware. Prior works [19], [20] typically assumed that the malicious user could dispatch operations to both software and hardware while sharing memory between them. *Furthermore, note that the attacker does not need to be an endpoint of Internet communication to intercept inter-packet delay.* It is sufficient for the attacker to merely eavesdrop on an ongoing Internet traffic.

V. Performance Evaluation

A. Experiment Setup

The experiment setup implements the threat model in an AWS EC2 F1 instance to evaluate the performance of the proposed CTC. The AWS EC2 F1 instance featured eight vCPUs derived from an Intel Xeon E5-2686 v4 multiprocessor and an AMD Xilinx UltraScale+ XCVU9P-FLGB2104-2-I FPGA. To mimic FPGA logic of the victim, a simple echo server was implemented on the FPGA and the AXI Connection, as shown in Fig. 2, was used to receive and echo back messages between the FPGA and the vCPU. In addition, the transmitter of the first phase of the CTC was placed at the FPGA side interface of the AXI Connection (to be specific, R channel of the AXI Connection). The FPGA logic circuit was synthesized and implemented using Xilinx Vivado v2021.2, available through the FPGA Developer AWS Machine Image (AMI) at the time of the experiment. *Please note that the FPGA logic was clocked at 125 MHz.*

The vCPU side of the AWS EC2 F1 instance implemented a software service to route User Datagram Protocol (UDP) network clients to the echo server, mimicking the software belonging to the victim. Moreover, the receiver for the first phase and the transmitter for the second phase of the CTC too were implemented on the vCPUs. The receiver of the first phase of the CTC utilized the RDTSCP instruction (Read Time Stamp Counter and Processor ID) [22] to capture precise time stamps from the Time Stamp Counter (TSC) of the processor [23]. Modern Intel 64 and IA-32 processors, including vCPUs in AWS EC2 F1 instances, feature an advanced Invariant TSC [23] which continues to increment regardless of the state of the processor (sleep or active) and remains invariant to the core executing the RDTSCP instruction [23]. Inspired by [12], the transmitter for the second phase of the CTC performed the covert transmission by selectively delaying the transmission of UDP packets echoed back from the server. The attacker PC analyzed the inter-packet delay to recover the covertly transmitted information.

TABLE I
COVERT TIMING CHANNEL PERFORMANCE – RPA ATTACK EFFICACY

Power Trace Set	Bit Errors (out of 204.8M bits)	Power Traces to Recover Key	Increase in Power Traces to Recover Key (vs. Reference)
Reference	*Not Applicable*	17,100	*Not Applicable*
After CTC first phase	46,592	18,600	9%
After CTC second phase	1,302,835	21,300	24%

To demonstrate the effectiveness of the proposed CTC in RPA attacks, the exact set of power consumption traces from the RPA attack specific Power to Pulse Width Modulation Sensor (PPWM), published as artifacts [5], was used. PPWM is an area-efficient on-chip sensor that produces a single bit output per measurement of power consumption [5]. This characteristic enables attackers to effectively utilize a lower-bandwidth channel, such as the proposed CTC, to discreetly transfer power consumption data from a cryptographic engine deployed on a cloud FPGA to an off-cloud PC, where key inference can be performed using Correlation Power Analysis (CPA). *Please note the power consumption traces correspond to an unprotected 128-bit Advanced Encryption Standard (AES) circuit.*

In this experiment, the set of power consumption traces was transmitted via the first phase (AXI) and the second phase (Internet) of the CTC. After each phase, the output was collected at the receiver end and subjected to CPA for key inference. A CPA was also conducted on the original set of power consumption traces as reference.

In the first phase of the CTC, a logic `1` was covertly transmitted by delaying the VALID signal of the AXI Connection by 2 clock cycles, while a logic `0` was transmitted without inducing a delay. In the second phase of the CTC, a logic `1` was transmitted by delaying a UDP packet by 2 ms, with no added delay for the transmission of a logic `0`. Synchronization between transmitter and receiver is essential for accurate demodulation [24]. However, as UDP is a best-effort service without packet delivery guarantees [25], packet loss can disrupt synchronization, hindering the demodulation of subsequent transmissions. To address the aforesaid issue, the transmitter introduced a 100 ms delay after every 2,000$^{\text{th}}$ unit of data, defined as a *block*. A *block* contains data bits along with a set of parity bits and reference bits to aid in demodulating and decoding the data bits. The receiver advanced the computation to the start of a *block* upon observing the increased delay. Missing observations were padded with zeros. Data in 2,000 *blocks* equated to 100 power consumption traces from PPWM and the frequency of inserting the 100 ms delay was experimentally determined. In this paper, a streaming protocol, feeding continuous data, is assumed above the *block* level of the protocol stack. However, the streaming protocol can be replaced by a packet-based protocol [26], [27], if required.

B. Results

Table I summarizes the key experiment result of this paper. The first column of Table I identifies the power consumption trace sets used for CPA. The next column presents the number of bit errors observed in the power consumption traces, which in total spanned 204.8 million bits. The subsequent column displays the number of power consumption traces needed to successfully recover the secret key of the targeted cryptographic engine. The last column quantifies, as a percentage, the increase in the number of power consumption traces required for complete key recovery relative to the reference trace set.

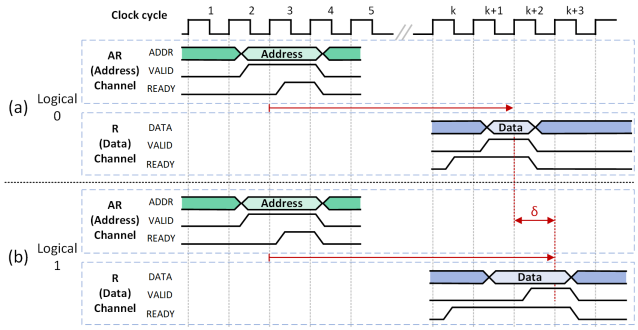


Fig. 3. AXI Read Transaction - Timing Diagrams

In this experiment, the reference trace set needed 17,100 power consumption traces to fully recover the cryptographic key. After transmission through the first phase of the CTC, this number increased by 9% to 18,600. Subsequently, the final output after the second phase of the CTC required 21,300 power consumption traces, marking a 24% increase compared to the reference. Overall, this experiment demonstrated the possibility of using the proposed CTC together with a PPWM sensor to perform an RPA attack that does not require physical access to measure or retrieve power consumption traces from a cloud FPGA to an off-cloud PC. *Please note that BERs of first phase and second phase of the CTC were* 0.02275% *and* 0.6134%*, respectively.*

VI. THE PROPOSED COVERT TIMING CHANNEL – DESIGN PARAMETERS

A. Overview of Design Parameters

This section evaluates the impact of key design parameters on the first phase of the CTC. The authors expect similar effects to be observed in the second phase of the CTC. The transmitter of the first phase of the CTC is implemented as a logic circuit on the FPGA that can manipulate the timing of the VALID signal of the AXI Connection and the receiver of the CTC is a software component integrated within the compute pipeline on vCPUs, which gauges the time consumed by a read transaction. The transmission medium of the CTC is the time consumption of transactions performed through the AXI Connection.

When read transactions occur through the AXI Connection, the transmitter of the CTC bit-by-bit injects the *block* into the transmission medium, the time consumption of read transactions, by delaying the VALID signal of the R channel by an arbitrary number of clock cycles. In the bit modulation scheme for the CTC, a logic `0` is transmitted by inducing no delay in the VALID signal. Conversely, a logic `1` is transmitted by delaying the VALID signal by δ clock cycles. Upon completing the read transaction, the receiver residing on the vCPU side measures the elapsed time and uses the elapsed time measurements of the modulated reference bits to demodulate the data bits and the parity bits. Afterwards, the covertly transmitted data bits are decoded using the intrinsic information from the received data bits and the extrinsic information from the received parity bits. *Note that the number of delay clock cycles,* δ*, is chosen arbitrarily by the attacker at the design time of the CTC.*

The timing diagrams (a) and (b) in Fig. 3 correspondingly show the behavior of the signals on the AR and R channels of the AXI Connection for covertly transmitting a logic `0` and a logic `1`. The timing diagrams in Fig. 3 assume $\delta = 1$ and a read transaction

Algorithm 1 Algorithm for Decoding a *Block*

1: **function** DECODE($[D_1, D_2, D_3, \ldots, D_n]$)
2: $m \leftarrow$ row count of H
3: $E \leftarrow [D_1, D_2, D_3, \ldots, D_n]$
4: $L \leftarrow \begin{bmatrix} 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}_{m \times n}$
5: **for** $i \leftarrow 1, \ldots, n$ **do**
6: **for** $j \leftarrow 1, \ldots, m$ **do**
7: **if** $H[j, i] = 1$ **then**
8: $L[j, i] \leftarrow D_i$
9: **for** $iterations \leftarrow 1, \ldots, 8$ **do**
10: **for** $j \leftarrow 1, \ldots, m$ **do**
11: $row \leftarrow [L[j, 1] \;\; L[j, 2] \;\; \ldots \;\; L[j, n]]$
12: $pos \leftarrow 0, \quad min_1 \leftarrow \infty$
13: **for** $k \leftarrow 1, \ldots, n$ **do**
14: **if** $(min_1 > |row[k]| \wedge H[j, k] = 1)$ **then**
15: $min_1 \leftarrow |row[k]|$
16: $pos \leftarrow k$
17: $min_2 \leftarrow \infty$
18: **for** $k \leftarrow 1, \ldots, n$ **do**
19: **if** $(min_2 > |row[k]| \wedge H[j, k] = 1 \wedge pos \neq k)$ **then**
20: $min_2 \leftarrow |row[k]|$
21: $sign \leftarrow 1$
22: **for** $k \leftarrow 1, \ldots, n$ **do**
23: **if** $(H[j, k] = 1)$ **then**
24: $sign \leftarrow sign \times sgn(row[k])$
25: $row_r \leftarrow [0 \;\; 0 \;\; \ldots \;\; 0]_{1 \times n}$
26: **for** $k \leftarrow 1, \ldots, n$ **do**
27: **if** $(H[j, k] = 1)$ **then**
28: **if** $(k = pos)$ **then**
29: $row_r[k] \leftarrow min_2 \times (sign \times sgn(row[k]))$
30: **else**
31: $row_r[k] \leftarrow min_1 \times (sign \times sgn(row[k]))$
32: $[L[j, 1] \;\; L[j, 2] \;\; \ldots \;\; L[j, n]] \leftarrow row_r$
33: $E \leftarrow E + [\sum_{i=1}^{m} L[1, i] \;\; \sum_{i=1}^{m} L[2, i] \;\; \ldots \;\; \sum_{i=1}^{m} L[n, i]]$
34: $L \leftarrow \begin{bmatrix} E[1] & E[2] & \ldots & E[n] \\ \vdots & \vdots & \ddots & \vdots \\ E[1] & E[2] & \ldots & E[n] \end{bmatrix}_{m \times n} - L$
35: **return** E

consumes k clock cycles. It should be emphasized that the value of k can vary across read transactions.

The receiver of the CTC serves two main functions: (1), demodulating the received signal and (2), decoding the received information. In the demodulation step, the receiver measures the time taken to perform each of the read transactions over the AXI Connection and remaps the expected value of a transmission of logic `1` and the expected value of a transmission of logic `0` to the real numbers -1 and 1, respectively. The remapping is chosen to simplify the subsequent decoding step and is influenced by binary phase-shift keying (BPSK) modulation [28]. In BPSK, a digital signal is modulated into an analog carrier signal by shifting the phase into one of the two (hence, binary) phase offsets.

1) Decoding at the Receiver: The algorithm shown in Algorithm 1 is used for decoding a *block* and it is derived from the min-sum algorithm used in the LDPC code schemes [29]. Min-sum algorithm produces a minimal set of simple operations for decoding a BPSK transmission over an additive white Gaussian noise (AWGN) channel and the inherent resource efficiency of the algorithm is essential to preserve the stealthiness of the CTC. The parity check matrix used in experiments is defined in Equation 1 and is referred to as H, in Algorithm 1.

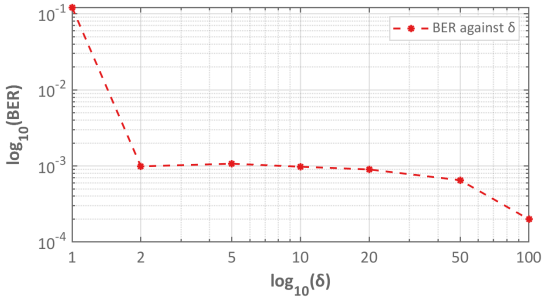


Fig. 4. Variation in Bit-Error Rate – δ

Algorithm 1 is structured as a function named DECODE, which takes an array, $[D_1, D_2, D_3, \ldots, D_n]$, as input. At the start of the DECODE function, m, E and L are initialized to the number of rows in the parity check matrix, the input array and an $m \times n$ matrix of zeros, respectively. Then, each element $L[j, i]$ of matrix L is set to D_i, the input array element at the index corresponding to the column index of $L[j, i]$, if $H[i, j]$ equals 1 (lines 5 - 8). The subsequent section of the DECODE function iteratively improves the beliefs stored in array E, aiming to recover the originally transmitted bits. Based on preliminary experiments, the iterative process (lines 10 - 34) required eight rounds for the optimal result.

$$H = \begin{bmatrix} 1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&1 \end{bmatrix} \quad (1)$$

The iterative process transforms the matrix L by replacing each of its rows with a replacement row (row_r) computed using the values of the row being replaced. Firstly, the algorithm finds the minimal value (min_1) among the elements of row that have a 1 stored in the identical position of H (lines 13 - 16). Simultaneously, pos is updated with the position of min_1 (line 16). Next, a second minimal value (min_2) is found among the elements of row, excluding the element in position pos, that have a 1 stored in the identical position of H (lines 18 - 20). Then, the product of the signs of all elements of row that have 1 stored in the identical position of H is computed (lines 22 - 24) and stored in $sign$. The last step in constructing row_r, populates an array of zeros of length n with $min_2 \times (sign \times sgn(row[k]))$ if $k = pos$ and $min_1 \times (sign \times sgn(row[k]))$ otherwise, provided that the identical position of H is 1. After constructing row_r, row of L is replaced by row_r and the computation moves to the next row until all rows are transformed. Then, the array E is updated by adding the column sum of L to E (line 33). As the final step of the iterative process, L is updated by subtracting L from a matrix that is generated by replicating E, m number of times to match the height of L. The final output of Algorithm 1 is the values stored in E after the eighth round.

After executing Algorithm 1, the error-corrected data bits received can be inferred by observing the sign of the corresponding elements of E. If the sign of the element is negative, the bit is inferred as a logic `1` and if the sign of the element is positive or zero, the bit is inferred as a logic `0`. In this manner, bits that are covertly transmitted by the transmitter are inferred at the receiver, establishing

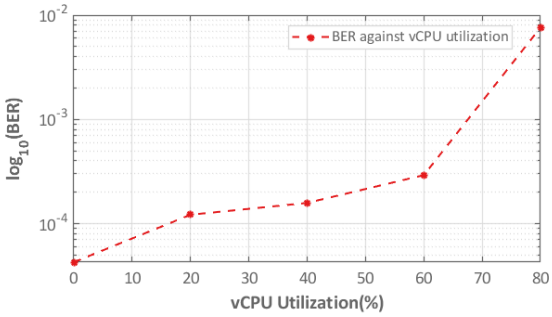


Fig. 5. Variation in Bit-Error Rate – vCPU Utilization

a CTC between the FPGA side and the vCPU side of a cloud FPGA instance. The experiments in this paper were based on $n = 16$ and *blocks* containing 2 data bits.

B. Impact of δ on Bit-Error Rate

To evaluate the dependence of the BER of the CTC on δ, the number of delay clock cycles induced on the VALID signal to covertly transmit a logic `1`, multiple experiments were carried out by varying δ between 1 and 100. The result of this experiment is shown in Fig. 4, with δ on the x-axis and BER on the y-axis, both presented on the base-10 logarithmic scale ($\log_{10}$). The graph in Fig. 4 shows that the CTC is effective for all δ values tested in the experiment. Notably, the BER decreases sharply when δ increases from 1 to 2 and an overall downward trend continues, reaching a minimum BER of 0.01988% at $\delta = 100$. The decrease in BER, as δ increases, can be attributed to improved distinguishability between covertly transmitted logic `1`s from covertly transmitted logic `0`s and the sharp decline in BER from $\delta = 1$ to $\delta = 2$ can be hypothesized as a consequence of exceeding the ambient noise level of the AWS EC2 F1 instance.

In this experiment, changing the value of δ impacted the time taken to complete a read transaction. However, the interval between consecutive read transactions depended on the frequency of echo requests from the client side, regulated by the scheduling of processes on the vCPUs and therefore unaffected by the value of δ. It is important to highlight that the bandwidth of the CTC is influenced by both the intra-read and inter-read transaction delays. When changing δ from 1 to 100, no discernible pattern was observed in the overall time required to execute each experiment, indicating the prominence of the uncertainty associated with the inter-read transaction delay.

C. Impact of vCPU Utilization on Bit-Error Rate

This experiment examined the impact of the vCPU utilization on BER by placing an artificial CPU-intensive workload on the vCPU side of the cloud FPGA instance. Using the `taskset` command, the workload was assigned to the same vCPU as the receiver of the CTC, while CPU utilization of the workload was managed with the `cpulimit` utility. δ was set to 10 for this experiment.

As depicted in Fig. 5, with vCPU utilization on the x-axis and BER (in the $\log_{10}$ scale) on the y-axis, BER shows an upward trend with increasing vCPU utilization. This observation aligns with expectation, as the receiver of the CTC is unable to precisely capture the timestamp when the artificial workload is scheduled on the vCPU. Consequently, inaccuracies in the recorded time intervals lead to decoding errors and demodulation errors, causing higher BER. vCPU utilization becomes relevant to the BER only when the artificial workload interrupts an ongoing read transaction. *Note that, scheduling the artificial workload between two read transactions does not lead to bit errors and as seen from Subsection VI-B, the inter-read transaction delay was prominent, reducing the likelihood of the artificial workload interrupting an ongoing read transaction.* Therefore, in contrast to approaches that rely on modulating covert information via inter-read transaction delay, the proposed technique, which encodes covert information by modulating the duration required to complete a read transaction, offers enhanced reliability for the attacker.

TABLE II
VARIATION IN BIT-ERROR RATE – MODULATION TECHNIQUE

δ	BER		Relative Improvement as Percentage
	Proposed Scheme (LDPC-based)	Repetition Code	
1	12.140%	22.510%	185%
2	0.099%	0.144%	146%
5	0.048%	0.066%	138%
10	0.098%	0.116%	119%
20	0.090%	0.191%	213%
50	0.065%	0.158%	243%
100	0.020%	0.147%	739%

D. Impact of Modulation Technique on Bit-Error Rate

In order to assess the effectiveness of the LDPC-based error correction scheme used in the decoding algorithm, an experiment was carried out by augmenting Algorithm 1 to implement a decoder based on repetition code. The structure of a *block* was also changed, to facilitate the repetition code and the error corrected two data bits of the *block*, d_1 and d_2, were calculated according to Equation 2 and Equation 3, respectively. The experiment was repeated for different values of δ in the range of 1 to 100.

$$d_1 = \begin{cases} 0, & \text{if } \frac{(d_{11}+d_{12}+d_{13}+d_{14}+d_{15}+d_{16}+d_{17}+d_{18})}{8} > 0 \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

$$d_2 = \begin{cases} 0, & \text{if } \frac{(d_{21}+d_{22}+d_{23}+d_{24}+d_{25}+d_{26}+d_{27}+d_{28})}{8} > 0 \\ 1, & \text{otherwise} \end{cases} \quad (3)$$

Table II presents the results of the experiments evaluating LDPC-based and repetition code schemes for the CTC. The first column of Table II represents the values of δ used in the experiment. The second and third columns show the BER obtained from the experiment using the proposed LDPC-based scheme and repetition code, respectively. The fourth column shows the improvement in BER as a percentage, calculated by dividing the repetition code BER by the LDPC-based BER. The results in this section show that the proposed LDPC-based scheme outperformed the repetition code, even though it is composed of a minimal set of simple operations. In the pair of experiments where $\delta = 100$, the BER improvement of the proposed LDPC-based scheme is over seven folds compared to the repetition code. Hence, these results portray the ability of the proposed LDPC-based scheme to effectively decode *blocks* while maintaining the stealthiness of the CTC.

VII. CONCLUSION

This paper introduced, for the first time, a CTC that provides an attacker with the capability to covertly transfer data from a cloud FPGA to an off-FPGA resource. Using experiments, this paper demonstrated the application of the CTC in exfiltrating power consumption traces from an on-chip sensor on a cloud FPGA to an off-cloud attacker PC for performing RPA attack. Further, an analysis on the BER of the proposed CTC under various operating conditions was also presented. The findings emphasize the need for increased vigilance and security measures from both cloud FPGA providers and users to protect logic circuit designs deployed on cloud FPGAs.

References

[1] Arm Limited, "Learn the architecture - an introduction to AMBA AXI," https://developer.arm.com/documentation/102202/0300/AXI-protocol-overview, 2023.

[2] D. Patel, "AWS shell interface specification," https://github.com/aws/aws-fpga/blob/master/hdk/docs/AWS_Shell_Interface_Specification.md, 2023.

[3] R. W. Shirey, "Internet Security Glossary, Version 2," https://www.rfc-editor.org/info/rfc4949, Aug. 2007.

[4] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "Remote inter-chip power analysis side-channel attacks at board-level," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–7.

[5] B. Udugama, D. Jayasinghe, H. Saadat, A. Ignjatovic, and S. Parameswaran, "A power to pulse width modulation sensor for remote power analysis attacks," *TCHES*, vol. 2022, no. 4, p. 589–613, Aug. 2022, artifact available at https://artifacts.iacr.org/tches/2022/a17.

[6] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *CHES 2004*, 2004, pp. 16–29.

[7] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[8] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between FPGA long wires," in *13th ASIACCS*, 2018, p. 15–27.

[9] D. R. E. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-based covert channels using FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 6, jun 2021.

[10] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Reading between the dies: Cross-SLR covert channels on multi-tenant cloud FPGAs," in *37th IEEE ICCD*, 2019, pp. 1–10.

[11] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *41st IEEE Symposium on Security and Privacy*, 2020, pp. 1728–1741.

[12] C. Girling, "Covert channels in LAN's," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 292–296, 1987.

[13] R. Krösche, K. Thimmaraju, L. Schiff, and S. Schmid, "I DPID it my way! a covert timing channel in software-defined networks," in *17th IFIP Networking*, 2018, pp. 217–225.

[14] K. S. Lee, H. Wang, and H. Weatherspoon, "PHY covert channels: Can you see the idles?" in *11th USENIX Symposium on Networked Systems Design and Implementation*, Seattle, WA, Apr. 2014, pp. 173–185.

[15] Department of Computer Science, Cornell University, "SoNIC: Software-defined network interface card," http://fireless.cs.cornell.edu/sonic/index.php, 2023.

[16] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *16th ACM CCS*, 2009, p. 199–212.

[17] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for Cross-CPU attacks," in *25th USENIX Security Symposium*, Austin, TX, Aug. 2016, pp. 565–581.

[18] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani, "DFS covert channels on multi-core platforms," in *2017 IFIP/IEEE International Conference on VLSI-SoC*, 2017, pp. 1–6.

[19] A. Purnal, F. Turan, and I. Verbauwhede, "Double trouble: Combined heterogeneous attacks on Non-Inclusive cache hierarchies," in *31st USENIX Security Symposium*, Boston, MA, Aug. 2022, pp. 3647–3664.

[20] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "Jackhammer: Efficient rowhammer on heterogeneous FPGA-CPU platforms," *TCHES*, vol. 2020, no. 3, p. 169–195, Jun. 2020.

[21] Advanced Micro Devices Inc, "Intellectual property," https://www.xilinx.com/products/intellectual-property.html, 2023.

[22] Intel Corporation, "How to benchmark code execution times on intel® IA-32 and IA-64 instruction set architectures," https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf, 2010.

[23] ——, "Intel® 64 and IA-32 architectures software developer's manual combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, and 4," https://cdrdv2.intel.com/v1/dl/getContent/671200, 2023.

[24] C. Shen, J. Zhang, and G. Qu, "Mes-attacks: Software-controlled covert channels based on mutual exclusion and synchronization," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[25] "User Datagram Protocol," https://www.rfc-editor.org/info/rfc768, Aug. 1980.

[26] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "On countermeasures against the thermal covert channel attacks targeting many-core systems," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[27] J. González-Gómez, M. B. Sikal, H. Khdr, L. Bauer, and J. Henkel, "Smart detection of obfuscated thermal covert channel attacks in many-core processors," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[28] K. Sourabh, R. Deepika, V. Lekshmi, S. Sudhakar, and J. Manikandan, "Design and evaluation of BPSK demodulator using model based programming," in *16th INDICON*, 2019, pp. 1–4.

[29] J. Zhang, M. Fossorier, D. Gu, and J. Zhang, "Two-dimensional correction for min-sum decoding of irregular LDPC codes," *IEEE Communications Letters*, vol. 10, no. 3, pp. 180–182, 2006.

StreamCSD: SSD-Autonomous Stream Management via In-Storage Content Learning

Wenjie Li[2*], Xiang Chen[1*],
Yelin Shan[2], Jiapin Wang[2], Yunxin Huang[2], Yafei Yang[2], Tao Lu[2✉], You Zhou[1], Fei Wu[1✉]
[1]Huazhong University of Science and Technology, [2]DapuStor Corporation
[*]Co-first authors, [✉]Corresponding authors: lvtao@dapustor.com, wufei@mail.hust.edu.cn

***Abstract*—Write amplification (WA) from migrating valid pages during garbage collection (GC) degrades SSD performance and lifespan. Although stream management based on high-level software semantics reduces WA, existing solutions require host modifications, hindering their adoption. We introduce StreamCSD, an SSD-autonomous stream management approach using in-storage content learning, eliminating host-side changes. Leveraging compression ratios from embedded compressors in computational storage drives (CSDs), StreamCSD employs a streaming K-means algorithm to cost-efficiently cluster data into streams. Evaluations show that StreamCSD reduces WA from 1.7 to 1.06 under multimodal generative AI workloads, matching state-of-the-art methods with minimal impact on bandwidth. StreamCSD operates without host modifications, promoting broader adoption of multi-stream SSDs.**

I. Introduction

SSDs in data centers typically serve multiple workloads, such as databases, virtual machines, and file storage. These workloads produce writes, updates, and deletions, each with distinct patterns and timing. Consequently, the lifetime of the data is inconsistent. A standard block-interface SSD cannot distinguish between pages from different write streams due to a lack of semantic information. As a result, pages within a flash erase block have varying lifetimes, leading to the migration of many valid pages during garbage collection (GC) [21], [22] and causing write amplification (WA), which degrades SSD performance and requires over-provisioned flash resources for mitigation, which are major sources of the *block interface tax* [9].

Effectively mitigating WA involves storing pages with similar lifetimes in the same physical stream or block group within SSDs. However, accurately predicting page lifetime is a challenge due to workload variability and dynamics [19], [23]. In practice, instead of relying directly on lifetime, existing approaches utilize programmer hints through new IO interfaces [9], [10] and stream IDs based on data types [17] and program contexts [19], which are semantically or empirically linked to data lifetimes [28], to manage streams.

Open-Channel [10] and Zoned Namespace (ZNS) [9] revamp the block interface, empowering the host with direct control over the placement of data in flash blocks. However, they are incompatible with systems and applications designed for block interfaces. In contrast, multi-stream SSDs [17], [19], [25] and Flexible Data Placement (FDP) [27] enhance the existing block interface by allowing the host to use software semantics to establish distinct write streams. This enables SSDs to judiciously place data with the same stream ID across separate flash blocks. However, these solutions require extensions to the host-SSD communication protocol. This necessitates modifications to NVMe drivers or applications to include stream or placement identifiers, presenting a barrier to adopting these stream management schemes.

The interface changes or host dependencies result in significant limitations to existing schemes. As a supplier of data center SSDs, more than 95% of our SSD shipments consist of traditional block-interface SSDs. Given the still prevailing use of block-interface SSDs, we raise a critical and fundamental question: Can we retain block interfaces while eliminating host dependencies to devise a fully SSD-autonomous, cost-efficient stream management mechanism for reducing GC-induced WA?

We introduce *Learned Streaming*, an SSD-autonomous stream management approach that leverages content as a novel semantic dimension to eliminate host dependency. However, learned streaming requires data feature extraction, which presents challenges in terms of *universality* and *efficiency*. For instance, traditional content clustering algorithms, such as TF-IDF [8], are limited to text formats (e.g., txt, JSON) and suffer from slow execution speeds, making them unsuitable for latency-sensitive SSD stream management.

We propose *StreamCSD* to address the above challenges. By using Shannon entropy (ENT) or compression ratio (CR) as features, *StreamCSD* can differentiate a wide range of streams with varying lifetimes. By leveraging the computational storage drive (CSD) [32] architecture and piggybacking page feature extraction on SSD hardware compressors (emerging SSDs tend to integrate hardware compressors to reduce storage cost and prolong lifetime) [11], [29], [31], *StreamCSD* introduces negligible overhead. Furthermore, for SSDs without built-in compressors, the overhead can be mitigated by implementing a Shannon entropy accelerator. Following feature extraction, *StreamCSD* employs a lightweight *streaming K-means* algorithm for feature clustering and stream assignment. *Learned Streaming* based on the page compression ratio also faces the challenge from two aspects. First, pages sharing the same compression ratio might have different lifetimes. Second, pages with different compression ratios may have similar lifetimes. These limitations compromise the goal of separating data with different lifetimes. We further propose GC-guided page-to-stream remapping (§IV-C) and secondary lifetime-assisted stream mapping (§IV-D) to complement *Learned Streaming*.

We implement *StreamCSD* with the *Learned Streaming* mechanism on both the FEMU emulator [20] and a PCIe 5.0 ×4-lane SSD with a built-in hardware compressor. Experimental results show that *StreamCSD* reduces WA by 63.4% and improves IO performance by 74% compared to baseline, matching or surpassing state-of-the-art methods including *PCStream* [19] and MiDAS [23]. Under multimodal generative AI workloads, StreamCSD lowers the WA from 1.7 to 1.06 while having a negligible effect on write bandwidth.

II. Background and Related Work

The block interface tax [9], [10] arises from the mismatch between the block interface and flash memory's characteristics. To bridge this gap, numerous mechanisms have been proposed. Open-channel [10], [24] and ZNS [9], [15] interfaces expose different levels of flash memory management to the host. The multi-stream [17] and FDP [27], [34] allow hosts to deliver hints of data placement to SSDs.

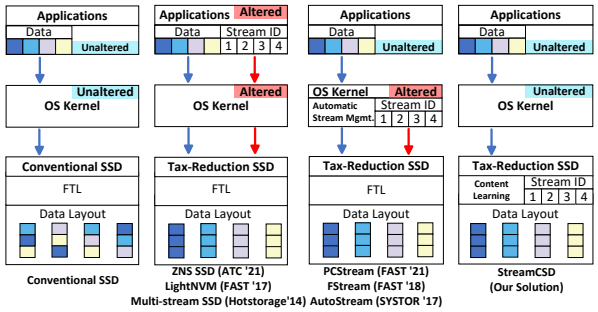


Fig. 1: SSD block interface tax reduction mechanisms. StreamCSD targets a host-transparent multi-stream SSD with a block interface.

Previous works have exploited various software hints through the multi-stream interface, such as multi-level semantics of LSM-tree [17], data types in file systems [25], and program contexts at the application layer [19]. LBA-based clustering methods classify page hotness by analyzing historical LBA write patterns, including recency, frequency, sequentiality and so on.

Figure 1 illustrates the host dependencies of existing mechanisms. Open-channel [10] and ZNS [9] interfaces require significant modifications to the application and OS kernel. The open-channel interface exposes raw NAND details, forcing the host to manage tasks like data mapping, garbage collection, and wear leveling. ZNS requires the host to manage sequential writes, zone resets, and garbage collection while the SSD firmware handles wear leveling. Similarly, Multi-stream [17] and FDP [27] require the application and OS kernel to assign stream or placement IDs, demanding application-specific knowledge to abstract different write streams. Although techniques like PCStream [19], FStream [25], and AutoStream [39] can autonomously extract application streams, they still necessitate OS kernel modifications. LBA-based clustering can be integrated directly into SSDs, independent of the host. However, these methods often require extensive statistics, such as LBA update intervals and block GC survival counts, or even develop complex models [23], [39] for clustering. These methods consume non-trivial amounts of valuable CPU and memory resources in SSDs, and rely on LBA access locality, making them unsuitable for frequently changing IO patterns and log-structured systems (e.g., RocksDB and F2FS) [19].

III. Motivation: Content-based Stream Management

A. Opportunities and Challenges

Content-based clustering, a robust unsupervised learning technique, is extensively employed in recommendation systems and big data analysis to group objects like documents [33], images [12], and videos [7] based on similarity or relevance. Techniques such as the *K-means* algorithm exploit term frequency-inverse document frequency (TF-IDF) vectors for clustering documents [8]. Inspired by these methods, we investigate the application of content learning within SSDs to enhance data placement and reduce write amplification.

However, implementing content-based clustering for intra-SSD stream management faces significant challenges. Current algorithms, designed for specific types of data such as documents and images, cannot be universally applied across the varied data handled by SSDs. Furthermore, the computational overhead of these clustering methods, particularly the high-dimensional processing and vectorization in TF-IDF, imposes a substantial CPU burden, demonstrated by a latency of 0.24 seconds for clustering 64 pages (or 4ms per page). This can severely impact SSD performance. Therefore, a universal and efficient content-based clustering algorithm is essential to effectively differentiate data streams without overwhelming the SSD resources.

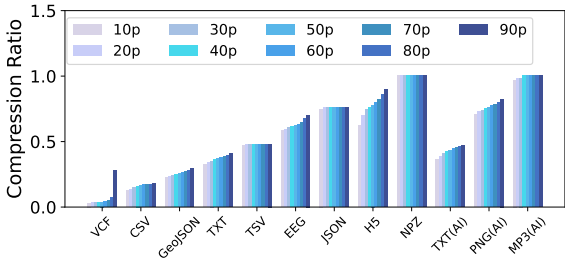


Fig. 2: Percentile distribution of 4KB page compression ratio in real datasets. '10p, 20p' refers to the 10th, 20th percentile CR and etc. These datasets are also employed to fill the microbenchmark as described in Table I and will be published.

B. Compression Ratio: A Good Multi-Stream Indicator?

Compression ratio feature. Typically, different types of data content are often managed independently, resulting in distinct write streams. For instance, file systems and databases typically separate user data, system metadata, and recovery logs into different streams. Similarly, multimodal generative AI applications produce a variety of content types—such as text, images, audio, and video—each stored in separate streams based on its unique structural characteristics. These structural differences, or content patterns, play a critical role in determining data compressibility. Compression algorithms exploit these patterns to minimize data size by replacing repeated sequences with pointers (e.g., LZ [41]), using shorter codes for frequent symbols and longer ones for rare symbols (e.g., entropy coders [16]). Therefore, the compression ratio is potentially a valuable indicator of the internal pattern and structure of data content.

Compression ratio and stream ID. To demonstrate whether compression ratio or entropy is a good indicator for distinguishing streams, we analyze 9 domain-various datasets from a public repository [1] and 3 AI-generated datasets [3]. Each dataset corresponds to a write stream and is split into 4KB pages, which are compressed to quantify percentile distributions of compression ratios. As depicted in Figure 2, the CRs of pages within each stream converge closely, with a mean deviation of only 0.02, which indicates *compressibility locality*. A similar observation is made in a recent work [36]. We also observe that the CRs or entropy values of pages diverge across different write streams. As Figure 3 illustrates, 9 of the 12 datasets exhibit significant distinction in either compression ratio or entropy value, highlighting a strong correlation between compression ratio and entropy. From the above results, we can see both the compression ratio and entropy are universal and representative features effective enough for content-based data page clustering.

Compression ratio and data lifetime. As shown in Figure 4, entropy strongly correlates with data lifetime in many important scenarios, and this also applies to the compression ratio. RocksDB's Write-Ahead Logs (WALs) exhibit lower entropy values or compression ratios and shorter lifetimes compared to Sorted String Tables (SSTables) [19]. WALs avoid compression to minimize latency [5], while SSTables use Snappy compression to reduce storage usage. Similar patterns are seen in Cassandra's flushing and compaction SSTables [2]. The redo and DB files in MySQL are difficult to distinguish based on entropy. However, for the overall data, those with lower entropy values tend to have shorter lifetimes, while data with higher entropy values tend to have longer lifetimes. In GCC compilation, intermediate files like preprocessed and assembly files have different entropy and compression ratios compared to long-lived object files. The correlation between file extensions and compression ratios has also been proven effective for stream separation [17], [25].

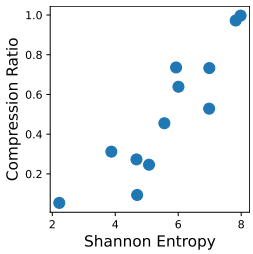


Fig. 3: Compression ratios and Shannon entropy.

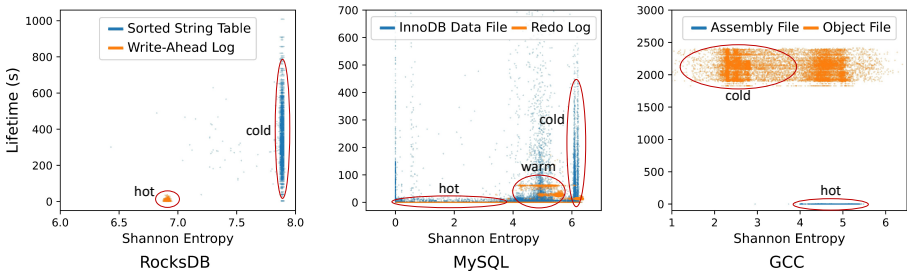


Fig. 4: Correlation between Shannon entropy and data lifetime in representative databases and Linux kernel source code compiling applications.

IV. Design of StreamCSD

A. Overview

To alleviate the block interface tax, we propose a computational SSD, called *StreamCSD*, which employs a *learned streaming* mechanism. It bears the following design goals:

- **G1: Host-transparent in-storage multi-streaming.** StreamCSD presents a standard block device and infers stream ID based on page content for automatic multi-streaming to reduce GC overhead. No application-specific expertise and host software modifications are required.
- **G2: Real-time multi-streaming.** StreamCSD can perform real-time identification of write streams among incoming data pages on the write IO path, eliminating the need to track and analyze historical access patterns of LBAs.
- **G3: Comparable efficacy to existing approaches.** StreamCSD should be comparable to or even outperform host-guided multi-stream approaches and LBA-based clustering approaches in representative application scenarios.
- **G4: Free from side effects.** StreamCSD should be lightweight with insignificant overheads. Even when the workload is unfriendly for StreamCSD, the performance and WA should not degrade.

Figure 5 presents the design overview of StreamCSD, which integrates a content learning accelerator for real-time page feature extraction. The *Shannon entropy* or *compression ratio* can be used as the learned feature, which is consumed by the streaming K-means algorithm to separate incoming data pages into different learned streams (§IV-B). The basic workflow of StreamCSD is as follows. ❶ The host NVMe write request is transmitted into the SSD via the PCIe interface. ❷ The NVMe controller inside the SSD parses the request. ❸ Each data page is transferred into the data cache of the content learning accelerator via a high-speed bus for content learning and feature extraction. These feature values are processed through clustering to obtain the corresponding stream ID (LearnedID). ❹ The data pages along with their stream IDs are written into the SSD write buffer. ❺ The FTL flushes data pages with different stream IDs to the corresponding physical blocks or reclaim units.

However, pages originating from different applications, with varying lifetimes but similar compression ratios, may be clustered into the same stream. StreamCSD employs a *GC-guided stream remapping* mechanism (§IV-C) to overcome this issue. Moreover, multi-stream SSDs support a limited number of physical streams, typically 4 to 16 [9], [19], [39], but high-concurrent application scenarios can generate more learned streams than available physical streams. To resolve this mismatch, we propose a *secondary lifetime-assisted stream mapping* mechanism that efficiently merges learned streams to fit the available physical streams (§IV-D).

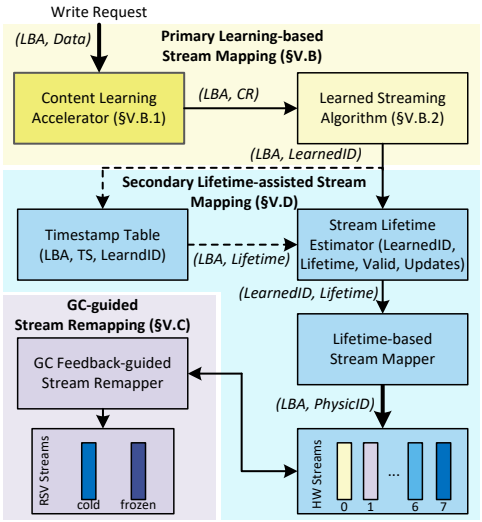


Fig. 5: The design overview of StreamCSD. Note, the secondary lifetime-assisted stream mapping mechanism is conditional. When it is not activated, learned IDs directly map to physical stream IDs.

B. Primary Page Content Learning-based Stream Mapping

Compression ratio and Shannon entropy, as universal data-characterizing features, yield similar classification accuracy. However, relying on the SSD controller's ARM cores for compression or Shannon entropy calculation can create performance bottlenecks in the write IO path. To address this, we propose using hardware acceleration and a learned streaming algorithm within the SSD as an effective solution.

1) Content Learning Accelerator: Accurately determining the stream for each page is critical for host-transparent multi-stream SSD. Efficiency is crucial in designing the feature extraction algorithm to avoid a significant impact on SSD write performance.

Piggyback feature extraction on compressors. Some SSDs have integrated hardware compressors [29], [31] for storage savings. In these instances, *StreamCSD* can utilize the SSD's internal compressor to extract page features, thereby eliminating the need for an additional content-learning accelerator. Consequently, *StreamCSD* leverages the page compression ratio as a primary metric in identifying page stream ID, without incurring extra computational costs.

Shannon entropy accelerator. In SSDs without built-in compressors, implementing a Shannon entropy accelerator for feature extraction is recommended, as it is significantly simpler than a compression engine [6], [18]. The hardware engine for entropy computation efficiently analyzes symbol (byte) frequency in input data. It consists of a data buffer for input storage, a counter array to monitor symbol frequencies, a probability calculator, a logarithm unit

utilizing an approximation technique, and an accumulator to handle the multiplication of probabilities and logarithms. In our simulation on a *Vivado XCVU19P FSVA3824* FPGA, the entropy module is remarkably compact, taking up less than 1% of the area relative to embedded processor cores. The computation for a 4KB page is accomplished in 1024 cycles, corresponding to 1μs at a 1GHz core frequency. Notably, a 6.25% sampling of 256 bytes achieves 98% entropy accuracy with a calculation time of only 0.05μs, ensuring minimal impact on SSD performance.

2) Learned Streaming Algorithm: **Streaming K-means clustering.** For real-time efficiency in SSD multi-stream identification, we employ a streaming K-means algorithm [30] to cluster learned streams. Unlike traditional K-means, which processes data in batches, streaming K-means continuously updates clusters and assigns stream IDs as new data arrives. The algorithm begins by specifying the number of clusters (K) and initializing centroids to maximize the minimum distance between them [14]. As new pages arrive, the algorithm calculates distances based on accelerator-generated features, assigns stream IDs to each page, and updates centroids when the predefined batch size is reached.

Wait-free clustering for single-page. Page features and stream IDs are cached until a predefined threshold triggers a batch update to the K-means centroids. Notably, our customized K-means algorithm assigns stream IDs on a per-page basis, eliminating clustering wait time. This separation of model updates from stream ID assignments ensures immediate stream ID retrieval, which is crucial for continuous data influx. As a result, tasks such as FTL address translation and backend data flushing can proceed without delay. Combining hardware-accelerated features of compression ratio or entropy, this approach ensures immediate page flushing to NAND flash with readily available stream IDs.

C. GC-guided Stream Remapping

Learning-based stream mapping is effective for streams with consistent lifetimes, such as WAL or SSTables. However, non-uniform workload distributions (e.g., *Zipfian*, *latest*) [13], and limitations of unsupervised clustering could result in the mixing of hot and cold data. To handle this problem, StreamCSD employs GC-guided remapping to reduce lifetime variance within each physical stream. GC-guided remapping reallocates outlier pages to dedicated streams based on their hotness.

Identification of outlier pages in a stream. Intra-SSD multi-stream management enables StreamCSD to detect page hotness by utilizing GC statistics. The typical GC process involves: 1) selecting a victim block; 2) migrating valid pages from the victim block to a new block; and 3) erasing the victim block. To identify outlier pages, StreamCSD tracks the migration count of each valid page. If a page undergoes excessive migrations, indicating its longevity and outlier status, it should be moved to a more appropriate stream. Empirical evaluation shows that 4 bits per migration count are sufficient, requiring only 128MB for a 1TB SSD.

Remapping state transition. StreamCSD designates two streams (*cold* and *frozen*) for outlier pages (refer to Figure 6). Pages are initially assigned to normal streams based on their stream IDs. If a page is migrated during GC, its *migration count* is incremented. When the count exceeds the *T1* threshold, the page is remapped to the cold stream. Cold stream pages are either returned to the original stream (with the *migration count* set to 0) or moved to the frozen stream. Pages in the frozen stream may move to the cold stream, setting their *migration count to T1*. The default *T1* and *T2* values are 1 and 2, based on empirical testing across various workloads.

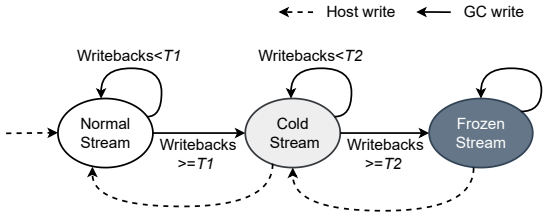


Fig. 6: The remapping state transition. *T1* and *T2* are configurable writeback thresholds.

D. Secondary Lifetime-assisted Stream Mapping

In multi-application environments, the number of learned streams generated by the primary stream mapping may exceed the available physical SSD streams [39], [40]. We propose secondary lifetime-assisted stream mapping to tackle this issue. The main idea is to merge learned streams into the same physical stream based on their lifetime estimates.

***Learned stream* lifetime.** The learned stream's lifetime is estimated from its pages' lifetimes. However, tracking each page's lifetime requires significant memory, especially for high-capacity SSDs. To balance accuracy and memory consumption, we track lifetime at the chunk level. A chunk contains 256 consecutive pages, which share a timestamp table entry containing *chunk ID*, *learned stream ID*, and *timestamp*. If pages within a chunk belong to different learned streams, the chunk will have multiple entries, each corresponding to a different learned stream. Upon page updates or deletions, the timestamp table entry is adjusted, and new entries are created if none exist. Each update computes the chunk's lifetime by subtracting the *timestamp* from the current time, which is then used to update the learned stream's lifetime. By averaging the chunk lifetimes, we estimate the learned stream's lifetime.

Stream mapping mechanism. We designed a mapping mechanism to align learned streams with physical streams. Learned streams with similar lifetimes are grouped into the same physical stream, reducing intra-stream lifetime variance. A stream mapping table records the learned-to-physical mapping and other fields. Due to space limitations, we omit details.

V. Evaluation

A. Experimental Setup

We implemented StreamCSD on FEMU [20], a widely used NVMe SSD emulator for full-system research [35], [37]. StreamCSD is configured similarly to [35], with 32GB storage capacity, 8 channels, 8 planes per channel, 512 blocks per plane, 256 pages per block, and a 4KB page size. The Greedy policy [26], [38] is adopted as the default for GC victim selection. We also developed a real system prototype based on a PCIe 5.0 7.68TB CSD with a hardware compressor to further validate the functionality and efficiency of StreamCSD. We primarily use the FEMU emulator for comprehensive evaluation. The

TABLE I: Microbenchmark and representative real-world applications that are employed for evaluation.

Benchmark	Description
Microbench (with public datasets)	Eight threads read data from real-world datasets and append to files with various data lifetimes.
RocksDB (DbBench)	LSM-tree based key-value store known for handling a large amount of sequential writes.
MySQL (TPC-C)	Relational database known for handling a large number of random writes.
SQLite (TPC-C)	Widely used embedded relational database with a lot of random writes.

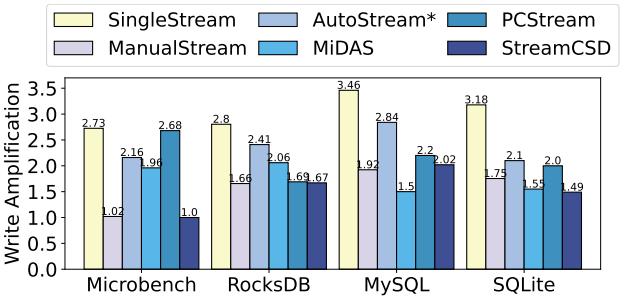


Fig. 7: Write amplification factor (WAF) (lower is better).

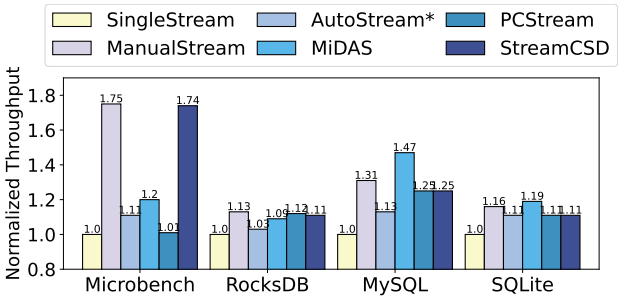


Fig. 8: SSD performance (higher is better).

hardware platform is used to evaluate AI workload performance and the actual overhead of learned stream management.

Benchmarks. We benchmark StreamCSD using a synthetic benchmark and three representative real-world applications listed in Table I. Microbench is designed with 8 threads, each sequentially writing datasets with varying entropy and lifetime. RocksDB performs random writes of 64 million key-value pairs, flushing WAL every 4 KB and compressing SSTables with the ZSTD algorithm. MySQL and SQLite are evaluated with TPC-C workloads, using 60 and 170 warehouses, respectively. The total data written exceeds the SSD capacity: 280GB for RocksDB and 120GB each for MySQL and SQLite, all hosted on the Ext4 file system.

Counterparts. We dedicated significant efforts to port state-of-the-art multi-streaming schemes listed in Table II. Particularly, AutoStream [39] leverages historical IO patterns for LBA-based stream identification. PCStream [19] identifies streams by using program context. MiDAS [23] segregates data by age and dynamically adjusts groups. To ensure fairness, write amplification and throughput metrics are evaluated under consistent resource conditions and configurations.

B. WAF and Performance Evaluation

WAF. We begin the evaluation by comparing the Write Amplification Factor (WAF) improvements under Microbench, as shown in Figure 7. Significant WAF reductions are observed for ManualStream and StreamCSD, both achieving around 63.4% reduction compared to SingleStream. MiDAS and AutoStream* also show reductions of 28.2% and 20.9%, respectively. However, PCStream shows negligible improvement, as Microbench generates IO requests using multiple threads that share the same task handler, resulting in insufficient program context to distinguish datasets with different lifetimes.

For RocksDB, StreamCSD and ManualStream both demonstrate the highest WAF reductions of approximately 40%, compared to PCStream (39.6%), MiDAS (22.5%) and AutoStream* (13.9%). For MySQL, MiDAS achieves the highest WAF reduction of 56.64%, followed by ManualStream (44.5%), StreamCSD (41.6%), PCStream (36.4%), and AutoStream* (17.9%). MiDAS identifies hot data based on update intervals but struggles with out-of-place updates, as valid pages may be directly discarded. This limits MiDAS' efficiency in RocksDB, where out-of-place updates are more prevalent. However, it works effectively for MySQL, which has many in-place updates. In contrast, StreamCSD addresses this limitation by learning data content, enabling more accurate stream identification upon the first write. In SQLite, StreamCSD significantly outperforms ManualStream. Although ManualStream uses separate lifetime hints for rollback journals, WALs, and database files, it fails to distinguish between hot and cold data within a single database file. In contrast, StreamCSD employs GC-guided remapping, enabling more effective differentiation of intra-file hot and cold data.

Performance. There is a strong correlation between WAF and performance. A lower WAF means lower GC overhead. As illustrated in Figure 8, StreamCSD significantly outperforms SingleStream in various applications, delivering performance improvements of 74%, 11%, 25%, and 11% for Microbench, RocksDB, MySQL, and SQLite, respectively. Compared to MiDAS, StreamCSD provides higher throughput for Microbench and RocksDB with extensive out-of-place updates, whereas MiDAS performs better for MySQL and SQLite with numerous in-place updates.

C. In-depth Analysis

Effect of primary learning-based stream mapping. We use RocksDB as a case study. As shown in Figure 7, we see that both StreamCSD and PCStream achieve near-optimal WAF reduction. To investigate further, we analyze the underlying mechanisms. RocksDB generates several types of files, including the Write-Ahead Log, level 0 SSTables (L0), and level 1 to n SSTables (L1–Ln). ManualStream assigns unique labels to these file types, allowing SSDs to manage them with dedicated Stream IDs for optimal performance. Similarly, PCStream identifies these streams by deriving program context from function call stacks, effectively differentiating WAL, L0, and L1–Ln. StreamCSD achieves a similar categorization by perceiving upper-level compression strategies: no compression for WAL, lightweight compression for L0, and standard compression for L1–Ln.

Effect of GC-guided stream remapping. We compare the WAF results with GC-guided remapping enabled and disabled across different workloads. Figure 9 shows that for MySQL and SQLite, where database files inherently mix hot and cold data, remapping reduces WAF by 22.9% and 21.2%, respectively. For RocksDB, remapping achieves a moderate WAF reduction of 5.6%, addressing SSTable level identification challenges. In contrast, for Microbench, remapping has minimal impact as data lifetimes within each learned stream have already converged. Futhermore, as shown in Figure 10, we analyze the remapping mechanism's sensitivity to threshold *T1*

TABLE II: Description of various multi-streaming schemes and a regular SSD as the baseline.

Scheme	Description
SingleStream (Regular SSD)	Mix pages from all streams causing serious write amplification. (**Baseline**)
ManualStream [17]	Multi-stream, programmer manually tag stream identification. (**Programmer-dependent**)
AutoStream*	LBA-based multi-streaming. A ported FEMU implementation of AutoStream [39]. (**Historical IO pattern-dependent**)
MiDAS [23] (State-of-the-art)	Self-adjusting data grouping, segregating data blocks by age. (**Historical IO pattern-dependent**)
PCStream [19] (State-of-the-art)	Multi-stream, program context-based stream identification. (**Host-dependent**)
StreamCSD (**Our solution**)	Multi-stream, content learning-based stream identification. (**Host-transparent**)

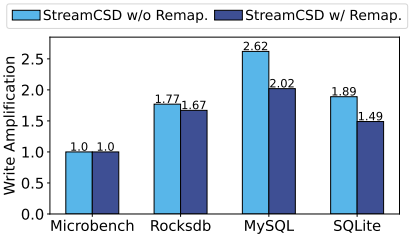


Fig. 9: Effect of GC-guided stream remapping and threshold sensitivity.

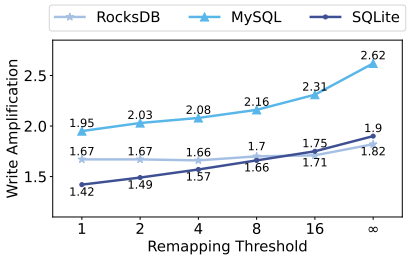


Fig. 10: Page copy-back ratio within physical streams.

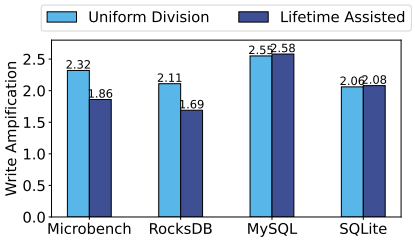


Fig. 11: Effect of secondary lifetime-assisted stream mapping.

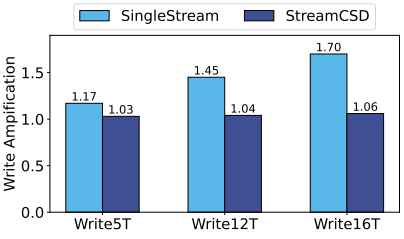


Fig. 12: WAF of a hardware StreamCSD with AI workloads.

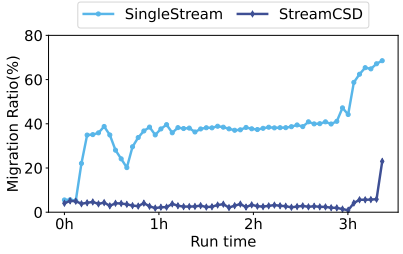


Fig. 13: GC migration ratio of a hardware StreamCSD with AI workloads.

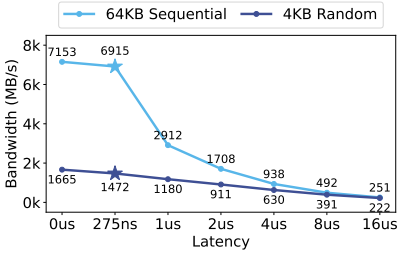


Fig. 14: Impact of stream management overhead on SSD write performance.

for RocksDB, MySQL and SQLite, respectively, with *T2* always set to *T1* + 1. Varying the threshold clearly influences WAF. Smaller thresholds lead to significant WAF reduction. At a *T1* threshold of 1, almost all workloads achieve the minimum WAF, which supports our default threshold settings.

Effect of secondary lifetime-assisted stream mapping. In a physical stream-limited scenario with 4 physical streams and 8 learned streams, we compare two merging strategies: one based on stream lifetime and the other on uniform division, where learned streams are evenly divided by IDs and mapped to physical streams. Figure 11 shows that, for Microbench and RocksDB, lifetime-assisted merging reduces WAF by 19.8% and 20.3%, respectively, highlighting the necessity of secondary mapping. However, this strategy does not improve other applications. For MySQL and SQLite, the relatively large page lifetime variance within learning streams limit the effectiveness of the lifetime-assisted strategy. It is worth noting that secondary mapping does not introduce significant side effects.

D. Hardware StreamCSD for Generative AI Workloads

We notice that the compression ratio-based stream management is very effective in emerging generative AI multimodality applications. We evaluate the StreamCSD hardware prototype using data from *OpenAI* [3] and *Runway* [4]. We emulate multimodality applications generating and concurrently flushing *ChatGPT* txt, *DALL-E* png, *OpenAI TTS* mp3, and *Gen-2* mp4 data into a 7.68TB SSD. Without prior large-scale file lifetime knowledge, we assign the lifetime of these data streams in a 1:2:4:8 ratio. The SSD's built-in compressor is used to obtain the data compression ratio; therefore, the cost of feature extraction is negligible. As shown in Figure 12, the WAF of SingleStream increases with the growth in generative AI file writes, while StreamCSD's WAF remains almost unchanged. Figure 13 demonstrates the variation in GC migration ratio during the *Write-16T* test. GC process resulted in an additional 10.68 TB of data being written to SingleStream, with an average 38% migration ratio, leading to a WAF of 1.7. However, StreamCSD significantly reduces GC-written data. It categorizes the four types of AI files based on their compression ratios and directs them to distinct streams. With an average 4% migration ratio, StreamCSD notably reduces WAF compared to SingleStream.

E. Impact of Learned Streaming on StreamCSD Performance

We evaluate the impact of learning-based stream management on write performance with the real PCIe 5.0 hardware SSD prototype. Learning-based stream mapping involves two components: a content-learning module for feature extraction and a clustering module for stream ID determination. We focus on the clustering module's impact, as the overhead of feature extraction is negligible due to the SSD's compression accelerator. Our customized streaming K-means algorithm takes *275ns* to assign a stream ID, resulting in a moderate performance impact—3.3% decrease for coarse-grain sequential writes and 11.6% for fine-grain random writes, as shown in Figure 14. These findings highlight the importance of lightweight feature extraction, efficient clustering algorithms, and hardware acceleration in intra-SSD stream management design.

VI. Conclusion

Traditionally, reducing SSD write amplification and enhancing performance have depended on changes to interfaces or host-assisted stream ID labeling. However, in the age of AI and computational storage, it is crucial to advance SSD intelligence toward device-autonomous stream management for WA reduction. StreamCSD addresses this need by utilizing unsupervised content learning to eliminate dependency on the host. It autonomously assigns stream IDs to IO requests through content clustering. StreamCSD has been implemented and validated on both the FEMU emulator and a production-grade SSD hardware prototype equipped with a data compression accelerator, showcasing the feasibility and effectiveness of autonomous multi-stream SSDs. Although the current use of compression ratio as the feature representation poses limitations, preventing the complete elimination of the block interface tax, StreamCSD still significantly reduces this tax. Overall, StreamCSD exemplifies the transformative role of AI in storage technology, with significant implications for the future of SSD design.

Acknowledgement

This work was supported by the National Key R&D Program of China (No.2023YFB4502901), the National Natural Science Foundation of China (under Grants 62372197, U22A2071, 62102155) and the Shenzhen Key R&D Program (No.KJZD20240903102459001). We sincerely thank anonymous reviewers and Prof. *Yu Hua* from HUST for their valuable comments.

References

[1] "Awesome public datasets," https://github.com/awesomedata/awesome-public-datasets, accessed: 2023-11-15.

[2] "Compression — apache cassandra documentation," https://cassandra.apache.org/doc/stable/cassandra/operating/compression.html, (Accessed on 05/14/2024).

[3] "Text generation models," https://platform.openai.com/docs/guides/text-generation, accessed: 2023-12-20.

[4] "Video generation models," https://runwayml.com/, accessed: 2024-01-16.

[5] "Write ahead log (wal) · facebook/rocksdb wiki," https://github.com/facebook/rocksdb/wiki/Write-Ahead-Log-%28WAL%29, (Accessed on 05/14/2024).

[6] B. Abali, B. Blaner, J. Reilly, M. Klein, A. Mishra, C. B. Agricola, B. Sendir, A. Buyuktosunoglu, C. Jacobi, W. J. Starke, H. Myneni, and C. Wang, "Data compression accelerator on ibm power9 and z15 processors : Industrial product," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 1–14.

[7] H. Alwassel, D. Mahajan, B. Korbar, L. Torresani, B. Ghanem, and D. Tran, "Self-supervised learning by cross-modal audio-video clustering," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9758–9770, 2020.

[8] P. Bafna, D. Pramod, and A. Vaidya, "Document clustering: Tf-idf approach," in *Proceedings of the International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016.

[9] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. Le Moal, G. R. Ganger, and G. Amvrosiadis, "Zns: Avoiding the block interface tax for flash-based ssds," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 689–703.

[10] M. Bjørling, J. Gonzalez, and P. Bonnet, "Lightnvm: The linux open-channel ssd subsystem," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017, pp. 359–374.

[11] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan *et al.*, "Polardb meets computational storage: Efficiently support analytical workloads in cloud-native relational database," in *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*, 2020, pp. 29–41.

[12] J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan, "Deep adaptive image clustering," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5879–5887.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing (SoCC'10)*, 2010, pp. 143–154.

[14] A. Grønlund, K. G. Larsen, A. Mathiasen, J. S. Nielsen, S. Schneider, and M. Song, "Fast exact k-means, k-medians and bregman divergence clustering in 1d," *arXiv preprint arXiv:1701.07204*, 2017.

[15] K. Han, H. Gwak, D. Shin, and J. Hwang, "Zns+: Advanced zoned namespace interface for supporting in-storage zone compaction," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 147–162.

[16] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[17] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive," in *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'14)*, 2014.

[18] S. Karandikar, A. N. Udipi, J. Choi, J. Whangbo, J. Zhao, S. Kanev, E. Lim, J. Alakuijala, V. Madduri, Y. S. Shao, B. Nikolic, K. Asanovic, and P. Ranganathan, "Cdpu: Co-designing compression and decompression processing units for hyperscale systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–17.

[19] T. Kim, D. Hong, S. S. Hahn, M. Chun, S. Lee, J. Y. Hwang, J. Lee, and J. Kim, "Fully automatic stream management for multi-streamed ssds using program contexts." in *FAST*, 2019, pp. 295–308.

[20] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The case of femu: Cheap, accurate, scalable and extensible flash emulator," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018, pp. 83–90.

[21] Y. Lu, J. Shu, and W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 257–270.

[22] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "Sfs: Random write considered harmful in solid state drives," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST'12. USA: USENIX Association, 2012, p. 12.

[23] S. Oh, J. Kim, S. Han, J. Kim, S. Lee, and S. H. Noh, "MIDAS: Minimizing write amplification in Log-Structured systems through adaptive group number and size configuration," in *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, 2024.

[24] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, and Y. Wang, "Sdf: Software-defined flash for web-scale internet storage systems," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, 2014, pp. 471–484.

[25] E. Rho, K. Joshi, S.-U. Shin, N. J. Shetty, J. Hwang, S. Cho, D. D. Lee, and J. Jeong, "Fstream: Managing flash streams in the file system," in *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018, pp. 257–264.

[26] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 1, pp. 26–52, 1992.

[27] C. Sabol and R. Stenfort, "Flexible data placement mode (fdp)," https://nvmexpress.org/wp-content/uploads/Hyperscale-Innovation-Flexible-Data-Placement-Mode-FDP.pdf, accessed: 2023-12-18.

[28] M. Satyanarayanan, "A study of file sizes and functional lifetimes," *ACM SIGOPS Operating Systems Review*, vol. 15, no. 5, pp. 96–108, 1981.

[29] ScaleFlux, "CSD 2000 Data Sheet," https://oss.scaleflux.com/202105/file_20210527_125844179.pdf, 2021, online; accessed 3 March 2022.

[30] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 1177–1178.

[31] Seagate, "Nytro 1000 SSD Series Datasheet)," https://www.seagate.com/www-content/datasheets/pdfs/nytro-1351-1551-sata-ssdDS1992-4-1907US-en_US.pdf, 2022, online; accessed on September 1, 2022.

[32] SNIA, "Computational storage architecture and programming model version 1.0," https://www.snia.org/sites/default/files/technical-work/computational/release/SNIA-Computational-Storage-Architecture-and-Programming-Model-1.0.pdf, 2022, online; accessed on September 3, 2022.

[33] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," 2000.

[34] R. Stenfort, "Hyperscale storage perspective," https://storageconference.us/2023/StenfortPresentation.pdf, Santa Clara University, 2023.

[35] S. Wang, Z. Lin, S. Wu, H. Jiang, J. Zhang, and B. Mao, "Learnedftl: A learning-based page-level ftl for reducing double reads in flash-based ssds," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 616–629.

[36] Y. Wang, Z. Sun, Y. Zhou, T. Lu, C. Xie, and F. Wu, "Balloon-zns: Constructing high-capacity and low-cost zns ssds with built-in compression," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[37] Y. Wen, X. Zhao, Y. Zhou, T. Zhang, S. Yang, C. Xie, and F. Wu, "Eliminating storage management overhead of deduplication over ssd arrays through a hardware/software co-design," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 320–335.

[38] M. Wu and W. Zwaenepoel, "envy: a non-volatile, main memory storage system," *ACM SIGOPS Operating Systems Review*, vol. 28, no. 5, pp. 86–97, 1994.

[39] J. Yang, R. Pandurangan, C. Choi, and V. Balakrishnan, "AutoStream: Automatic stream management for multi-streamed SSDs," in *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR'17)*, 2017.

[40] H. Yong, K. Jeong, J. Lee, and J.-S. Kim, "vStream: Virtual stream management for multi-streamed SSDs," in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[41] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

Look Before You Leap: A Self-Review Bayesian Optimization Method for Constrained High-Dimensional Design Space Exploration

Xuyang Zhao[1†], Yiyang Zhao[1†], Zheng Wu[1], Tianning Gao[2], Zhaori Bi[1*], Changhao Yan[1], Dian Zhou[12], Xuan Zeng[1*]
[1]State Key Laboratory of Integrated Chips & Systems, School of Microelectronics, Fudan University, China
[2]Department of Electrical Engineering, University of Texas at Dallas, U.S.A

***Abstract*—The parameterizable and synthesizable RISC-V processors enable the automatic generation of customized CPU cores through EDA tools. However, current methods often explore the extensive design space with significant model errors while neglecting design constraints, which are critical for practical implementations. To address these limitations, we propose a Self-Review Bayesian Optimization method (SRBO). This method integrates a teacher-student paradigm within a local Bayesian optimization framework to reduce model errors and enhance exploration efficiency. Additionally, it employs deep ensembles for effective constraint handling. Experimental results demonstrate that our approach outperforms state-of-the-art methods within a limited time budget, significantly enhancing exploration efficiency.**

***Index Terms*—High-dimensional Design Space Exploration, Teacher-Student Paradigm, Deep Ensembles, RISC-V Processor**

I. Introduction

As RISC-V processor agile design emerges and diverse application demands increase, CPU customization has become increasingly popular. Open-source frameworks, such as Chipyard [1], automatically generate Register-Transfer Level (RTL) files based on user-defined design configurations written in Chisel [2]. With the support of HAMMER framework [3] and Electronic Design Automation (EDA) tools, the entire Very Large Scale Integration (VLSI) flow, which is from Chisel to GDSII, can be automated. Designers can easily modify their designs by adjusting various design parameters, including microarchitecture parameters and EDA tool options, to achieve improved Performance, Power, and Area (PPA) outcomes. Consequently, traditional challenges of CPU design can be redefined as design parameter optimization or Design Space Exploration (DSE) problems.

However, conflicts often arise among various Quality of Results (QoRs) metrics. For instance, better performance often necessitates greater power consumption or increased area. Traditional optimization methods employ weighted approaches to balance PPAs, converting the problem into a Single-Objective Optimization (SOO) framework. Nonetheless, with the advent of hierarchical design methodologies, module specifications may need adjustment in response to upper-level design requirements. In a SOO scenario, any changes in weights assigned to QoR metrics render the problem fundamentally different, necessitating re-optimization. On the other hand, conducting design verification for each adjustment incurs significant time costs, often requiring several weeks to complete the entire VLSI flow. Consequently, Multi-Objective Optimization (MOO) methods have been introduced into the DSE problems. MOO results in a set of Pareto solutions, each representing an optimal design that reflects a trade-off among different QoRs. When design requirements change, designers can select a design from the Pareto set according to their demands, eliminating the need for re-optimization. This significantly reduces the repeated optimizations and enhances design efficiency.

Traditional MOO methods, such as NSGA-II [4] and MOEA/D [5], are based on evolutionary algorithms. However, they require a substantial number of simulation points, which is not suitable for DSE problems that necessitate time-consuming simulation. In contrast, Bayesian optimization (BO) is a model-based optimization method that can yield effective results with relatively few actual simulation points. BOOM-Explorer [6], [7] and GRL-DSE [8] utilize BO in conjunction with features of RISC-V microarchitecture to facilitate effective exploration of the microarchitecture design space. However, these methods necessitate the evaluation of entire design space using surrogate models to identify the next data point for simulation, even when only a subset of design points requires simulation. Therefore, they consider only a limited number of microarchitecture design parameters and neglect the EDA tool parameters, which also significantly influence the design process. In contrast, REMOTune [9] and ROI-HIT [10] employ local BO [11], [12] and dimension reduction techniques [13], [14], effectively handling the high-dimensional DSE problem including the parameters of both microarchitecture and EDA tools. However, they ignore the constraint handling. In practical design scenarios, although a design yields excellent PPA results after logical synthesis, it cannot work if it violates timing constraints. Consequently, it is essential to incorporate constraints into the DSE process.

In addition, the number of parameters within the design space, encompassing both microarchitecture parameters and EDA tools (such as logic synthesis tools), can exceed 50, with the scale potentially reaching 10^{30}. Simultaneously, completing the processes of RTL generation, RTL simulation, and logic synthesis for a specific design parameter configuration

†Equal contribution
*Corresponding authors: {zhaori_bi, xzeng}@fudan.edu.cn

can take from 1 to 6 hours. Consequently, DSE methods can only model this vast design space based on a very limited dataset, inevitably introducing errors. Furthermore, if simulation resources are wasted on an inappropriate candidate point, it incurs significant time costs that substantially impact the effectiveness of exploration. Therefore, we propose a Self-Review Bayesian Optimization (SRBO) method for constrained high-dimensional DSE. It integrates a teacher-student paradigm within a local BO framework to reduce model errors and employs deep ensembles for effective constraint handling. Our contributions are summarized as follows:

- We propose a local Bayesian optimization based on teacher-student paradigm. To address the challenges of inaccurate modeling and low exploration efficiency within a vast design space, we concentrate on the current optimal local region and employ variable selection to identify key design parameters. Subsequently, we utilize Gaussian Process Regression (GPR) models as "students" and employ Multi-Layer Perceptron (MLP) models as "teachers" to mitigate model misdirection, thereby enhancing optimization efficiency.
- We propose a constraint classifier based on deep ensemble. To assess whether candidate design points satisfy the design constraints, we employ multiple MLPs for classification. It reduces the need for actual simulation of infeasible design points, thereby further enhancing optimization efficiency.
- The experimental results demonstrate that SRBO outperforms the state-of-the-art methods, achieving $1.93\times \sim 41.47\times$ hypervolume increment within the same time budget.

The remainder of this paper is structured as follows: Section II introduces the problem formulation. Section III details the SRBO algorithm. Section IV presents experimental results and compares performances of our proposed algorithm with state-of-the-art algorithms. Section V concludes this article.

II. Problem Formulation

The DSE problem can be formulated as a constrained MOO problem, as shown in (1).

$$\begin{aligned} \text{minimize} \quad & f_1(\boldsymbol{x}), \cdots, f_M(\boldsymbol{x}), \\ \text{s.t.} \quad & g(\boldsymbol{x}) \geq C, \end{aligned} \tag{1}$$

where $\boldsymbol{x}$ is design parameters and $f_i(\cdot)$ represents PPAs. $g(\cdot)$ denotes constraints, such as setup and hold time requirements.

To assess the quality of MOO, the concept of *domination* is introduced. For arbitrary data points $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a} \prec \boldsymbol{b}$ ($\boldsymbol{a}$ *dominates* $\boldsymbol{b}$) if

$$\begin{aligned} & \forall i \in \{1, \cdots, M\} \quad f_i(\boldsymbol{a}) \leq f_i(\boldsymbol{b}) \\ \text{and} \quad & \exists i \in \{1, \cdots, M\} \quad f_i(\boldsymbol{a}) < f_i(\boldsymbol{b}). \end{aligned} \tag{2}$$

The MOO yields a Pareto Set PS and the corresponding performance space is referred to as Pareto front PF.

$$\begin{aligned} & PS = \{\boldsymbol{x} \in X_C \mid \nexists\, \boldsymbol{x'} \in X_C : \boldsymbol{f}(\boldsymbol{x'}) \prec \boldsymbol{f}(\boldsymbol{x})\}, \\ & PF = \{\boldsymbol{f}(\boldsymbol{x}) \mid \boldsymbol{x} \in PS\}, \; X_C = \{\boldsymbol{x} \in X \mid g(\boldsymbol{x}) \geq C\}. \end{aligned} \tag{3}$$

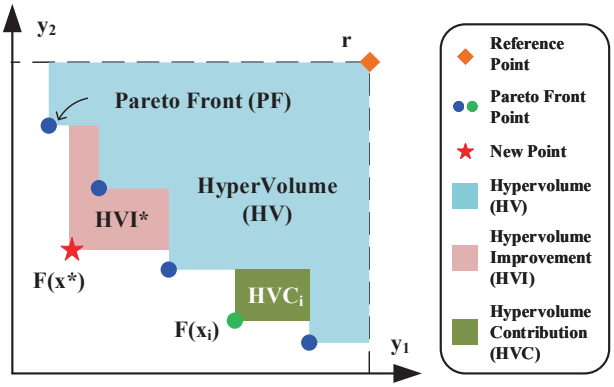


Fig. 1. The illustration of HyperVolume (HV), HyperVolume Improvement (HVI), and HyperVolume Contribution (HVC). A larger HV means a better PF.

In MOO, HyperVolume (HV) is a common performance indicator to assess the quality of the PF. As Fig. 1 shows, given a reference point $\boldsymbol{r}$ and a finite PF, the HV is the M-dimensional Lebesgue measure $\mathcal{M}$ of the space dominated by PF, as defined in (4).

$$\mathrm{HV}(PF, \boldsymbol{r}) = \mathcal{M}(\bigcup_{j=1}^{|PF|} [\boldsymbol{r}, \boldsymbol{y}_j]), \tag{4}$$

where $[\boldsymbol{r}, \boldsymbol{y}_j]$ denotes the hyper-rectangle bounded by $\boldsymbol{r}$ and $\boldsymbol{y}_j$. A larger HV value indicates a better PF, which means a better MOO result.

To compare the points in the PS, HyperVolume Contribution (HVC) is defined as (5). Similarly, to select the best candidate point, HyperVolume Improvement (HVI) of a candidate point $\boldsymbol{x}^*$ is defined as (6). As shown in Fig. 1, the green area is the HVC of $\boldsymbol{x}_i$, and the red area is the HVI of $\boldsymbol{x}^*$.

$$\mathrm{HVC}(\boldsymbol{x}_i | PF, \boldsymbol{r}) = \mathrm{HV}(PF, \boldsymbol{r}) - \mathrm{HV}(PF \setminus \boldsymbol{f}(\boldsymbol{x}_i), \boldsymbol{r}), \tag{5}$$

$$\mathrm{HVI}(\boldsymbol{x}^* | PF, \boldsymbol{r}) = \mathrm{HV}(PF \cup \boldsymbol{f}(\boldsymbol{x}^*), \boldsymbol{r}) - \mathrm{HV}(PF, \boldsymbol{r}). \tag{6}$$

III. Self-Review Bayesian Optimization

A. Overview of SRBO Framework

Our proposed SRBO framework is shown in Fig. 2. The SRBO method is built upon a local BO framework. It identifies key design parameters through a variable selection technique and explores designs near the current optimal design point with the highest HVC. The surrogate models leverage a teacher-student paradigm, where MLP models serve as the teachers and GPR models act as the students. This interaction reduces model error and yields more accurate prediction points. Additionally, SRBO employs a constraint classifier based on deep ensembles to model constraints, allowing it to assess the feasibility of candidate design points and filter out potentially invalid ones. Subsequently, the local BO computes the HVIs of the feasible candidate points based on the predicted values from the surrogate models, and the point with the largest HVI is selected for simulation. The results from the simulation will be incorporated into the explored dataset, thereby updating each model.

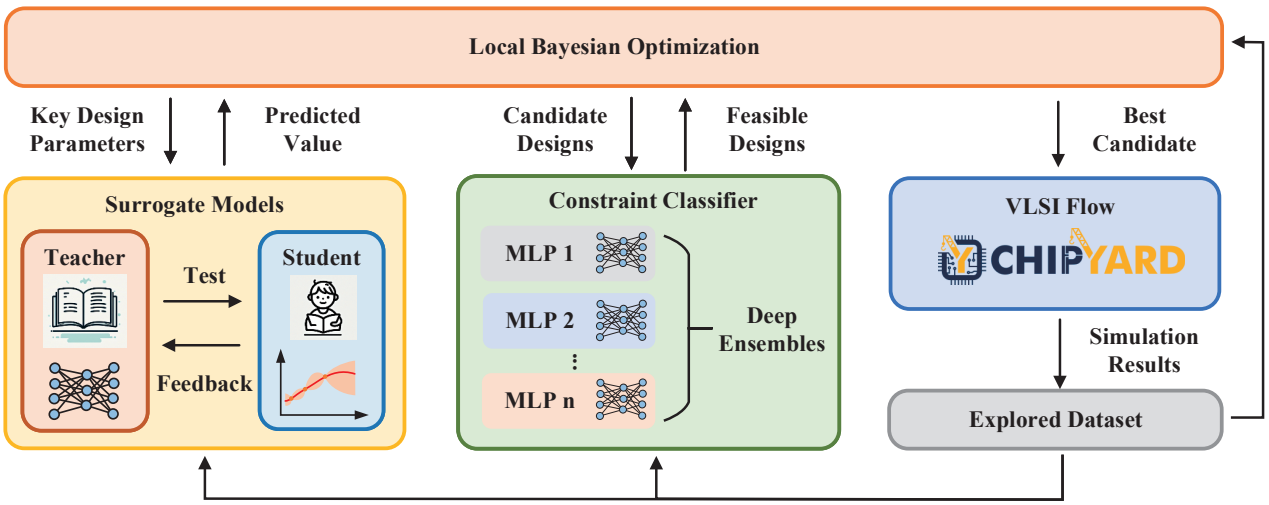


Fig. 2. An illustration of our proposed SRBO method. It utilizes a local BO framework to identify the key design variables and explore designs near the current optimal design point. The surrogate models based on the teacher-student paradigm model the PPAs in the key design space and provide the predicted value after self-review. Simultaneously, the constraint classifier based on deep ensemble assesses the constraints, filtering all candidate designs generated by the local BO to obtain the predicted feasible design points. The local BO ranks the candidates according to the predicted values provided by the surrogate models, selects the optimal candidate for simulation, and incorporates the results from the simulation into the explored dataset, updating each model with this new design point.

B. Teacher-Student Paradigm

The design space for high-dimensional DSE problems is extensive. For instance, the design space encompassing the parameters of the RISC-V BOOM microarchitecture and the Genus logical synthesis tool consists of 53 design variables with a range of 10^{30}. Consequently, modeling such a large design space using a limited number of sampled points inevitably introduces errors. These model errors may misguide the exploration process. Therefore, we employ a Teacher-Student Paradigm (TSP) to enhance model accuracy through self-review, improving overall exploration efficiency.

Traditional BO models the design space using a GPR model, which provides the mean and variance of predictions. Although the acquisition function can be employed to balance these mean and variance estimates, selecting an appropriate acquisition function and setting the hyperparameters often require sophisticated techniques, as the optimal choices can vary across different cases. This variability diminishes the robustness and generalization of BO.

TSP incorporates a teacher model and a student model. In our SRBO, the MLP model acts as the teacher $T(\cdot;\boldsymbol{\theta}_T)$, and the GPR model serves as the student $S(\cdot;\boldsymbol{\theta}_S)$. In addition to training with the dataset, the MLP incorporates the error between the predictions of the GPR model and the true outcomes into its loss function. By leveraging the GPR model's strong expressiveness while mitigating its misleading uncertainties, the MLP achieves more accurate predictions through a process of self-review, eliminating the need for experienced tricks.

As the Fig. 3 shows, firstly, the teacher (the MLP model) predicts the values of the candidate points (the red stars) based on the explored points D (the blue points).

$$\hat{\boldsymbol{y}}_{cand}(\boldsymbol{\theta}_T) = T(\boldsymbol{X}_{cand};\boldsymbol{\theta}_T), \tag{7}$$

where $\boldsymbol{X}_{cand}$ denote the candidate points.

Then, the teacher provides the candidate points and a selection of the explored points to the student for modeling.

$$\boldsymbol{\theta}_S^*(\boldsymbol{\theta}_T) = \arg\min_{\boldsymbol{\theta}_s} L_S(D_{test}(\boldsymbol{\theta}_T);\boldsymbol{\theta}_S), \tag{8}$$

$$D_{test}(\boldsymbol{\theta}_T) = \{\boldsymbol{X}_{cand} \cup \boldsymbol{X}_{valid}, \hat{\boldsymbol{y}}_{cand}(\boldsymbol{\theta}_T) \cup \boldsymbol{y}_{valid}\}, \tag{9}$$

where $\{\boldsymbol{X}_{valid}, \boldsymbol{y}_{valid}\}$ are extracted from the explored dataset D. L_S represent the loss function of the student. The student evaluates the explored points and returns the error between the predicted values and the actual values to the teacher.

$$L_{test}(D;\boldsymbol{\theta}_S^*(\boldsymbol{\theta}_T)) = MSE(\boldsymbol{\mu}_{\boldsymbol{\theta}_S^*(\boldsymbol{\theta}_T)}(\boldsymbol{X}), \boldsymbol{y}). \tag{10}$$

After that, the teacher is retrained in accordance with the loss function of the teacher L_T and the loss function L_{test} derived from the student's feedback to develop a more accurate model.

$$\boldsymbol{\theta}_T^* = \arg\min_{\boldsymbol{\theta}_T} L_T(D;\boldsymbol{\theta}_T) + \lambda L_{test}(D;\boldsymbol{\theta}_S^*(\boldsymbol{\theta}_T)). \tag{11}$$

In this work, our teacher model is set up as follows:

$$\begin{aligned} T(\boldsymbol{x};\boldsymbol{\theta}_T) : &Input\ \boldsymbol{x} \rightarrow Linear(n,256) \rightarrow ReLU \\ &\rightarrow Linear(256,64) \rightarrow ReLU \\ &\rightarrow Linear(64,32) \rightarrow ReLU \\ &\rightarrow Linear(32,1) \rightarrow Output\ y \end{aligned} \tag{12}$$

Acknowledging the potential for misleading pseudo-label predictions by the teacher, the student is evaluated against the existing ground truth labeled data, and its performance is consequently communicated back to the teacher. Upon the completion of the teacher-student interaction, the teacher, refined through regularization, produces reliable pseudo-labels for training the GPR model, thereby enhancing its data querying capability, even with limited labeled training data.

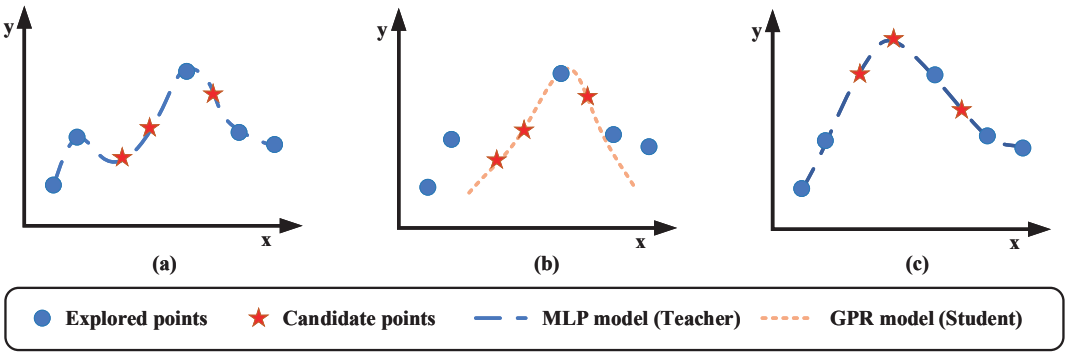


Fig. 3. An illustration of the Teacher-Student Paradigm (TSP). The blue points represent explored points, while the red stars denote candidate points. In the TSP of SRBO, the MLP model acts as the teacher, and the GPR model serves as the student. (a) Initially, the teacher predicts the values of the candidate points based on the explored points. (b) Subsequently, the teacher provides the candidate points and a selection of the explored points to the student for modeling. The student evaluates these explored points and returns the error between the predicted values and the actual values to the teacher, which is utilized as a loss function. (c) The teacher is then retrained in accordance with the loss function derived from the student's feedback to develop a more accurate model.

C. Constraint Classifier based on Deep Ensembles

In RISC-V processor design, it is crucial to adhere to the specified design constraints. For instance, if the timing constraints are not satisfied, even a design with optimal PPAs may result in functional errors, rendering it infeasible. Existing DSE methods often overlook constraint handling and typically only investigate design spaces that comply with these constraints. However, in the vast, high-dimensional DSE, there may exist design points that do not satisfy timing constraints, necessitating effective constraint management.

A common approach for handling constraints is to penalize design points that deviate from the required constraints. However, in digital IC design, a lack of adherence to constraints like timing can result in substantial changes across the entire design. At times, fine-tuning parameters may fail to yield a design that meets these constraints, rendering the distance from the constraint requirements ineffective as a guide for exploration direction.

To address these challenges, we propose using deep ensembles to transform the satisfaction of design constraints into classification problems. This involves utilizing multiple MLPs as classifiers and aggregating their predicted outputs to derive final results.

Deep Ensembles (DE) consists of a collection of neural networks, commonly MLPs, and are employed to model epistemic uncertainty. By training multiple independent MLPs and harnessing their diversity, DE improves accuracy and enhances uncertainty estimation. Each MLP in the ensemble is trained on the same dataset but initialized with different random weights. During inference, predictions from the ensemble are derived by averaging or combining the outputs of the individual models.

In the SRBO, we independently train N identical MLPs on all explored points D to model the probability of each design being either feasible or infeasible. The output of each MLP is a probability value between 0 and 1 for a given design point $\boldsymbol{x}$. Our constraint classifier computes the mean of the predicted values, utilizing the trained weights θ_i^* from each model.

$$C(\boldsymbol{x}) = \frac{1}{N}\sum_{i}^{N} M_{\theta_i^*}(\boldsymbol{x}). \tag{13}$$

In this work, we propose that if $C(\boldsymbol{x}) > 0.5$, $\boldsymbol{x}$ is considered a feasible design candidate for subsequent optimization. We utilized 5 MLPs to construct the DE, where each MLP comprises 2 hidden layers, each containing 64 neurons.

D. Local Bayesian Optimization

In addition to the vast design space associated with high-dimensional DSE problems, the time required for design simulation and verification is substantial. For instance, generating RTL files, performing RTL simulation, and executing logic synthesis for a single RISC-V BOOM design can take between 1 and 6 hours. Consequently, achieving a global optimum within a limited time budget is impractical. In this context, identifying a better solution is often more valuable than pursuing the absolute best. Therefore, our approach prioritizes the exploration of current optimal design points in the vicinity and emphasizes key design parameters, which significantly reduces the problem's dimensionality and enhances optimization efficiency.

Like ROI-HIT [10], we apply a sensitivity matrix S [15] to measure the importance of variables.

$$S = \{s_{ij} \mid i = 1, \cdots, n,\ j = 1, \cdots, M\}, \tag{14}$$

$$\begin{aligned} s_{ij} &= \frac{Var[E(y_j|x_i)]}{Var(y_j)} \approx \frac{\sum_{l=1}^{r_i} |A_i^l|[(\bar{y}_j)_i^l - \bar{y}_j]^2}{\sum_{\boldsymbol{x}\in X}[y_j(\boldsymbol{x}) - \bar{y}_j]^2}, \\ \bar{y}_j &= \frac{1}{|X|}\sum_{\boldsymbol{x}\in X} y_j(\boldsymbol{x}), \quad (\bar{y}_j)_i^l = \frac{1}{|A_i^l|}\sum_{\boldsymbol{x}\in A_i^l} y_j(\boldsymbol{x}), \end{aligned} \tag{15}$$

where X represents the dataset, r_i represents the number of value which x_i can be and A_i^l represents the set of points for which the variable x_i is set to its l-th value.

Given that the current optimal point in our work satisfies the constraints, we posit that the key design parameters should exhibit sensitivity to PPAs while remaining insensitive to the

constraints. Consequently, we assign equal weights to PPAs and assign negative values to the weights of the constraints. The key design variables are obtained as follows.

$$V_i = \sum_{j=1}^{m} (w_j \cdot s_{ij}) + C_p\sqrt{1/n_i}, \tag{16}$$

$$var_{key} = \{var_i \mid V_i \geq p \cdot V_{max}\}, \tag{17}$$

where n_i is the number that the i-th variable has been selected as the key variable, and C_p and p are hyper-parameters. In this work, $C_p = 0.01$ and $p = 0.5$.

We select the explored point with the highest HVC, which can be calculated using (5), as the optimal design. The design space is then restricted to the vicinity of this current optimum. For non-key variables, the values remain consistent with the current optimum, while for the key variables, only the values within the neighboring interval are considered as candidate points. Surrogate models and constraint classifier are trained exclusively for the key variables. Candidate points are firstly filtered from infeasible points using the constraint classifier. Subsequently, surrogate models are employed to predict their PPAs, followed by the calculation of HVI using (6). The design exhibiting the maximum HVI will be simulated.

IV. Experimental Results

A. Experimental Setup

We explore the design space for a RISC-V BOOM [16], [17] core and a Rocket [18] core, both configured as RV64GC. The PPA results are evaluated using System-on-Chip (SoC) designs in all experiments, each equipped with either the BOOM core or the Rocket core as its processor. The peripheral modules of these SoCs consist of standard components provided by the Chipyard framework. A 7nm ASAP7 Process Design Kit (PDK) [19] is utilized within the VLSI flow. Since ASAP7-based designs are not manufacturable, this paper focuses solely on the front-end design for the experiments. The design parameters are shown in TABLE I.

Firstly, the RTL design is generated by Chipyard framework. Then, an open-source simulator Verilator [20] simulates the design to obtain Cycle Per Instruction (CPI). 13 benchmarks are selected to evaluate the performance of the design: *dhrystone, median, mm, mt-matmul, mt-vvadd, multiply, qsort, rsort, spmv, towers, vvadd, fir2dim*, and *whetstone*. The average CPI of benchmarks is taken as the performance metric. Finally, Cadence Genus Synthesis Solution 19.10 is used as the logic synthesis tool to report the timing violations and estimate power and area metrics. The power metric is the total power. The clock frequency is set as 500MHz. Design points that fail to meet the timing constraints are categorized as failure points and are excluded from the resulting statistics.

We compare SRBO with 4 state-of-the-art methods: (1) BOOM-Explorer [6], a BO-based method for microarchitecture DSE, (2) GRL-DSE [8], another BO method for microarchitecture DSE, (3) REMOTune [9], a local BO method for high-dimensional DSE that by the trust regions and random embedding, (4) ROI-HIT [10], a recent local BO method for high-dimensional DSE that combines the ROIs and variable selection. To verify the effectiveness of TSP and DE, we implement two versions of SRBO: (1) SRBO w/o TSP, a SRBO method without TSP, (2) SRBO, the full version of SRBO.

All algorithms are implemented in Python. All algorithms are initialized with the same set of 100 designs. The experiments are executed within a maximum time budget of one day (86400 seconds). The HV indicator serves as a benchmark

TABLE I
Design Parameters

Class	N	Name	Value
BOOM	22	BPD	Boom2,TAGEL,Alpha21264
		fetchWidth	4,8
		numFetchBufferEntries	8,16,24,32,35,40
		numRasEntries	16,24,32
		maxBrCount	8,12,16,20
		decodeWidth	1,2,3,4,5
		numRobEntries	32,64,96,128,130
		numIntPhysRegisters	48,64,80,96,112
		memIssueWidth	1,2
		intIssueWidth	1,2,3,4,5
		numLdqEntries	8,16,24,32
		enablePrefetching	false,true
		enableSFBOpt	false,true
		numRXQEntries	2,4,8
		numRCQEntries	8,16,32
		nL2TLBEntries	128,256,512,1024
		nL2TLBWays	1,2,4,8
		nICacheWays	2,4,8
		nICacheTLBWays	8,16,32
		nDCacheWays	2,4,8
		nDCacheMSHRs	2,4,8
		nDCacheTLBWays	8,16,32
Rocket	9	mulUnroll	1,2,4,8
		divUnroll	1,2,4,8
		mulEarlyOut	false,true
		divEarlyOut	false,true
		nDCacheWays	1,2,4
		nDCacheTLBWays	4,8,16
		nDCacheMSHRs	1,2,4
		nICacheWays	1,2,4
		nICacheTLBWays	4,8,16
Genus	31	time_recovery_arcs	false,true
		auto_partition	true,false
		dp_analytical_opt	off,standard,extreme
		dp_area_mode	false,true
		dp_csa	basic,none
		dp_rewriting	basic,advanced,none
		dp_sharing	advanced,basic,none
		exact_match_seq_async_ctrls	false,true
		exact_match_seq_sync_ctrls	false,true
		iopt_enable_floating_output_check	false,true
		iopt_force_constant_removal	false,true
		iopt_lp_power_analysis_effort	low,medium,high
		lbr_seq_in_out_phase_opto	false,true
		optimize_net_area	true,false
		retime_effort_level	low,medium,high
		retime_optimize_reset	false,true
		syn_generic_effort	low,medium,high
		syn_map_effort	low,medium,high
		syn_opt_effort	low,medium,high
		leakage_power_effort	low,medium,high
		lp_power_analysis_effort	low,medium,high
		enable_latch_max_borrow	false,true
		latch_max_borrow	1:500:1
		enable_max_fanout	false,true
		max_fanout	2:64:1
		enable_lp_clock_gating_max_flops	false,true
		lp_clock_gating_max_flops	8,16,32
		enable_lp_power_optimization_weight	false,true
		lp_power_optimization_weight	0.1:0.9:0.1
		max_dynamic_power	3.0:150.0:0.1
		max_leakage_power	5.0:100.0:0.1

TABLE II
OPTIMIZATION RESULTS FOR RISC-V BOOM DSE

Algorithm	Δ HV	Ratio	# Fail.	# Sim.	Rate
BOOM-Explorer [6]	2	0.00×	**0**	12	**0.00%**
GRL-DSE [8]	726	0.00×	2	9	22.22%
REMOTune [9]	21,597	1.00×	6	13	46.15%
ROI-HIT [10]	295,011	13.66×	4	16	25.00%
SRBO w/o TSP	577,879	26.76×	3	16	18.75%
SRBO	**895,622**	**41.47×**	3	**17**	**17.65%**

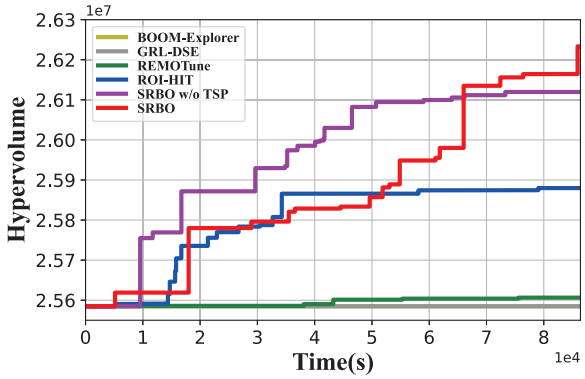


Fig. 4. The optimization results for RISC-V BOOM DSE.

TABLE III
OPTIMIZATION RESULTS FOR ROCKET DSE

Algorithm	Δ HV	Ratio	# Fail.	# Sim.	Rate
BOOM-Explorer [6]	65	0.00×	**0**	16	**0.00%**
GRL-DSE [8]	1,741	0.11×	5	24	20.83%
REMOTune [9]	15,822	1.00×	1	**29**	3.45%
ROI-HIT [10]	55,272	3.49×	2	**29**	6.90%
SRBO w/o TSP	72,618	4.59×	**0**	27	**0.00%**
SRBO	**107,470**	**6.79×**	**0**	26	**0.00%**

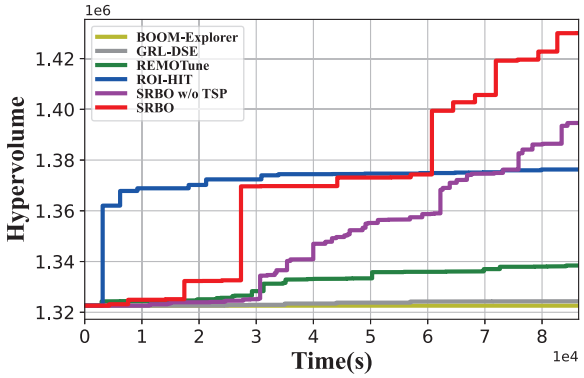


Fig. 5. The optimization results for Rocket DSE.

metric for evaluating the quality of the solutions generated by the algorithms. For HV computation, the reference point is defined as the maximum observed values of CPI, power, and area within the initial dataset. The experiments are conducted five times, and the average results are subsequently reported.

B. RISC-V BOOM Design Space Exploration

The optimization results are shown in TABLE II and Fig. 4. In the table, Δ *HV* denote the increase in HV compared to the HV of the initial dataset, while *# Fail.* and *# Sim.* represent the number of failure points and simulation points, respectively. Compared with the state-of-the-art methods, SRBO obtains the largest Δ HV and achieves 41.47× and 3.04× HV increment over REMOTune and ROI-HIT.

BOOM-Explorer and GRL-DSE are well-suited for microarchitecture design space exploration (DSE) problems with relatively few design variables. However, for high-dimensional DSE problems characterized by extensive design spaces, both BOOM-Explorer and GRL-DSE exhibit less increase in HV. The introduction of the local BO method significantly enhances the performance of ROI-HIT, SRBO w/o TSP, and SRBO during the early stages of optimization. Additionally, the implementation of a constraint classifier allows SRBO w/o TSP and SRBO to achieve a lower failure rate. Although the BOOM-Explorer has the zero failure rate, it has little HV increase, so its failure rate has little significance. Furthermore, thanks to the teacher-student paradigm, the accuracy of the SRBO model continually improves, enabling it to escape from local optima in the ROI-HIT method.

C. Rocket Design Space Exploration

The optimization results are shown in TABLE III and Fig. 5. Compared with the state-of-the-art methods, SRBO obtains the largest Δ HV and achieves 6.79× and 1.94× HV increment over REMOTune and ROI-HIT.

The optimization results for Rocket are generally consistent with those of RISC-V BOOM. However, Rocket is relatively easier to optimize within the specified constraints, allowing other unconstrained methods to also achieve favorable optimization results. As illustrated in the Fig. 5, both SRBO and ROI-HIT became trapped in the same local optimum region. But, due to the self-review model, SRBO successfully escapes from this region.

V. CONCLUSION

In this paper, we propose a self-review Bayesian optimization method for constrained high-dimensional DSE. To address the challenges posed by the vast design space and time-consuming simulations, SRBO integrates a teacher-student paradigm within a local BO framework to reduce model errors and enhance the exploration efficiency. Given the importance of design constraints in digital IC designs, SRBO utilizes deep ensembles for effective constraint handling. Experimental results demonstrate that our approach outperforms state-of-the-art methods within a limited time budget. Our proposed SRBO method is a general framework that we anticipate will be applicable to other high-dimensional, expensive optimization problems with constraints in the future.

ACKNOWLEDGMENTS

This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711901, partly by National Natural Science Foundation of China (NSFC) research projects 92473206, 62304052, 62474050, 92473207, 62474051.

REFERENCES

[1] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[2] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *2012 Design Automation Conference (DAC)*, 2012, pp. 1212–1221.

[3] H. Liew, D. Grubb, J. Wright, C. Schmidt, N. Krzysztofowicz, A. Izraelevitz, E. Wang, K. Asanović, J. Bachrach, and B. Nikolić, "Hammer: a modular and reusable physical design flow tool," in *2022 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1335–1338.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[5] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[6] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[7] ——, "Boom-explorer: Risc-v boom microarchitecture design space exploration," *ACM Transactions on Design Automation of Electronic Systems*, 2023.

[8] X. Yi, J. Lu, X. Xiong, D. Xu, L. Shang, and F. Yang, "Graph representation learning for microarchitecture design space exploration," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[9] S. Zheng, H. Geng, C. Bai, B. Yu, and M. D. Wong, "Boosting vlsi design flow parameter tuning with random embedding and multi-objective trust-region bayesian optimization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 5, pp. 1–23, 2023.

[10] X. Zhao, T. Gao, A. Zhao, Z. Bi, C. Yan, F. Yang, S.-G. Wang, D. Zhou, and X. Zeng, "Roi-hit: Region of interest-driven high-dimensional microarchitecture design space exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4178–4189, 2024.

[11] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable global optimization via local bayesian optimization," *Advances in neural information processing systems (NeurIPS)*, vol. 32, 2019.

[12] S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy, "Multi-objective bayesian optimization over high-dimensional search spaces," in *Uncertainty in Artificial Intelligence (UAI)*. PMLR, 2022, pp. 507–517.

[13] L. Song, K. Xue, X. Huang, and C. Qian, "Monte carlo tree search based variable selection for high dimensional bayesian optimization," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 28 488–28 501, 2022.

[14] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Feitas, "Bayesian optimization in a billion dimensions via random embeddings," *Journal of Artificial Intelligence Research*, vol. 55, pp. 361–387, 2016.

[15] L. Adjengue, C. Audet, and I. Ben Yahia, "A variance-based method to rank input variables of the mesh adaptive direct search algorithm," *Optimization Letters*, vol. 8, pp. 1599–1610, 2014.

[16] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167*, 2015.

[17] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020, pp. 1–7.

[18] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[19] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[20] W. Snyder, "Verilator 4.0: open simulation goes multithreaded," in *Open Source Digital Design Conference (ORConf)*, 2018.

An Input-Aware Sparse Tensor Compiler Empowered by Vectorized Acceleration

Xianhao He*, Haotian Wang*✉, Jiapeng Zhang*, Wangdong Yang*,
Anthony Theodore Chronopoulos†‡ and Kenli Li*✉
*College of Computer Science and Electronic Engineering, Hunan University
† Department of Computer Science, The University of Texas at San Antonio
‡ Department Computer Engineering and Informatics, University of Patras
{hexianhao, wanghaotian, zhangjp, yangwangdong}@hnu.edu.cn, antony.tc@gmail.com, lkl@hnu.edu.cn

***Abstract*—Sparsity is widely prevalent in real-world applications, yet existing compiler optimizations and code generation techniques for sparse computations remain underdeveloped. Sparse matrix-matrix multiplication (SpMM) is a representative operator in sparse computations, whose performance is often limited by the design of sparse formats and the extent of hardware architecture optimization. Most existing solutions achieve high-performance SpMM through two approaches: (1) meticulously designed kernels and specialized sparse formats, which require extensive manual effort, or (2) tensor compilers that support code generation, though these typically offer limited support for sparse patterns, making it challenging to adapt to complex sparsity patterns in practical applications. This paper presents SpMMTC, an input-aware sparse tensor compiler. Given a sparse matrix as input, SpMMTC analyzes its non-zero distribution and generates a vectorized kernel optimized for SpMM on the specific matrix. We evaluated SpMMTC on various workloads. It achieves speedups of 1.21x to 2.97x over state-of-the-art methods such as TACO, TVM, and ASpT on different multi-core processors. It also provides a speedup of up to 1.52x for sparse MobileNetV1 inference on the edge device.**

I. Introduction

Sparsity is prevalent in deep learning applications, such as image classification [1], natural language processing [2], and object detection [3]. As DNN models grow, pruning weights becomes essential to reduce storage and computation, especially on resource-constrained devices [4]. Unstructured sparsity, a common pruning method, removes weights with minimal impact on accuracy, resulting in matrices with many zeros. Likewise, graph neural networks [5] often feature sparse adjacency matrices. Sparse matrix-matrix multiplication (SpMM) is a fundamental operator in sparse computation, multiplying a sparse matrix A with a dense matrix X to produce a dense matrix Y. SpMM is widely applied in areas like PageRank [6], recommender systems [7], and graph clustering [8].

The performance of SpMM is primarily influenced by two factors: 1) ***sparse pattern*** and 2) ***hardware architecture***. The sparse pattern impacts memory access during computation, as the indices pointing to the dense matrix are non-contiguous, leading to a low data reuse ratio. The hardware architecture affects SpMM performance by enabling the development of high-performance parallel execution strategies that fully leverage the hardware's capabilities. Existing solutions for high-performance SpMM predominantly rely on runtime libraries. These libraries requires a carefully designed SpMM kernel and typically reorders the sparse matrices to improve data reuse before the kernel is executed. However, this often introduces cumbersome reordering algorithms and similarity measurements between rows or columns of the matrices [9]. Additionally, previous works, such as TC-GNN [10], aim to exploit dense blocks in sparse matrices to better utilize the tensor core units in NVIDIA GPUs, but this approach inevitably leads to unnecessary computations on zero elements, known as *redundant FLOPs* [11]. To achieve a high-performance SpMM kernel without extensive manual effort, tensor algebra compilers, such as TACO [12] and TVM [13], [14], have been proposed to automatically generate SpMM kernels. However, these compilers rarely analyze the sparse pattern of matrices, and TVM-style compilers often rely on long-term auto-tuning processes.

Therefore, to take both the sparse pattern and hardware architecture into consideration, we propose SpMMTC, an input-aware sparse tensor compiler for SpMM on multi-core CPUs. It takes the sparse matrix A and dimensions of dense matrix X as input, analyzes the non-zeroed distribution of A, then determines the optimal tile sizes along the direction of the rows of A and the columns of X based on the analysis results and backend architecture. Since the number of non-zeros in each row of A varies, the tiles of A may have unequal dimensions, which differs from the tiling techniques used in GEMM optimization [15]. SpMMTC generates high-performance kernels for these tiles that leverage SIMD units without redundant FLOPs. To accommodate the vectorized kernels, SpMMTC also generates transformation code to reorganize the data layout of the tiles. The generated code is assembled into several functions that are compiled into a library and invoked at runtime as needed.

In summary, our contributions in this paper can be concluded as the following:

- We propose SpMMTC, a sparse tensor compiler designed to automatically generate high-performance SpMM kernels tailored to specific sparse patterns, dimensions, and hardware architectures.
- SpMMTC analyzes the non-zero distribution of sparse matrices, optimizes memory access, and generates high-performance SpMM kernels with vectorized execution.
- We conducted experiments to evaluate SpMMTC on SpMM across various workloads on multi-core proces-

✉Corresponding Author

sors. The experimental results show SpMMTC achieves speedups of 1.21x to 2.97x over state-of-the-art baselines such as TACO, TVM, and ASpT on Intel and ARM multi-core processors. SpMMTC also provides a speedup of up to 1.52x for sparse MobileNetV1 inference on the widely-used edge device.

II. Background and Motivation

SpMM is formulated as $Y = AX$, where A is sparse matrix with dimension $M \times K$, X is a dense matrix with dimension $K \times N$, and their calculation results are stored in dense matrix Y with dimension $M \times N$. The sparse A is usually stored in compressed sparse format like CSR and CSC. Algorithm 1 illustrates SpMM computation using the CSR format. A key challenge in SpMM is the indirect memory access in the dense matrix X (Line 4). Because the column indices of matrix A are non-contiguous, multiple rows of X are skipped as the iteration over the K-dimension progresses, leading to poor cache locality.

Algorithm 1 SpMM (Sparse Matrix Matrix Multiplication)

Input: CSR $A[M][K]$, float $X[K][N]$
Output: float $Y[M][N]$
1: **for** $i = 0$ **to** $A.num_rows - 1$ **do**
2: **for** $k = A.\text{ptr}[i]$ **to** $A.\text{ptr}[i+1] - 1$ **do**
3: **for** $j = 0$ **to** $N - 1$ **do**
4: $Y[i][j] += A.\text{values}[k] * X[A.\text{col_idx}[k]][j]$

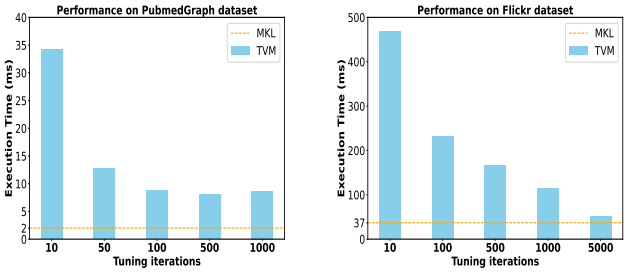


Fig. 1: Performance comparison between MKL and TVM for single-precision SpMM on x86 architecture

Recently, tensor algebra compilers such as TACO and TVM have been introduced to automatically generate high-performance kernels. These compilers generally adhere to the philosophy of separating the computation from the scheduling [16]. They provide a Domain-Specific Language (DSL) [17]–[19] that allows developers to independently define both the computation and the scheduling primitives of operations such as parallelizing, reordering, and unrolling for-loops. Moreover, TVM provides the auto-tuning ability to generate optimized, high-performance kernels. We compare the MKL [20] with TVM, using different tuning iterations, on the PubmedGraph and Flickr datasets[1] for SpMM on x86 architecture. The results are shown in Figure 1. As the number of iterations increases, the performance of TVM improves, but it remains lower than MKL; additionally, the tuning process is time-consuming (e.g., tuning 5,000 iterations on the Flickr dataset takes over 10 hours). While compiler-based methods free developers from dealing with intricate hardware architecture and complex kernel designs, they struggle to analyze sparse patterns, which hinders further optimization of SpMM kernels.

[1]available at https://networkrepository.com

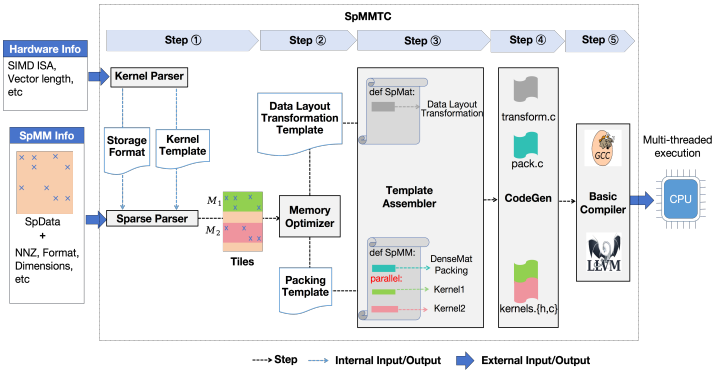


Fig. 2: Overview of SpMMTC. SpMMTC takes SpMM information, including the sparse matrix, storage format, dimensions of the two involved matrices, and other details as input, then optimizes and generates code tailored for the specific SpMM.

Motivated from the above analysis, we design SpMMTC, an input-aware sparse tensor compiler to generate SpMM kernels and its overview is shown in Figure 2. SpMMTC follows five steps to generate SpMM kernels: 1) It takes SpMM and hardware information as input, analyzes the sparse patterns, and determines the storage format, kernel template, and sparse tiles for the computation; 2) It optimizes memory access for sparse and dense matrices and outlines the corresponding templates; 3) It assembles these separate templates into functions; 4) It generates C/C++ source code to instantiate the templates and implement these functions; and 5) All the code is compiled and executed on multi-core CPUs.

III. SpMMTC Design

In this section, we detail SpMMTC, which is composed of the Kernel Parser, Sparse Parser, Memory Optimizer, Template Assembler, and CodeGen modules.

A. Kernel Parser

Kernel Parser determines how computations are performed on SIMD units and sketches out the sparse kernel template. Inspired by BLIS [15], the template uses an outer-product approach as shown in Figure 3. The advantages of outer product include a high computation-to-memory ratio (CMR) [21] and the elimination of redundant FLOPs.

1) Storage Format: The template utilized by Kernel Parser operates on tiles in A and X. To maintain the outer-product computation, it is essential to organize the sparse tiles of A in column-major order using three arrays: **tiles**, **indices**, and **data**, which are consistent with CSR or CSC formats. Each tile occupies a few entries in **tiles** that **tiles**$[i][0]$ represent the offset in **indices** and **data**, **tiles**$[i][1]$ indicates the starting row of the i-th tile. The indices of non-zero elements of this

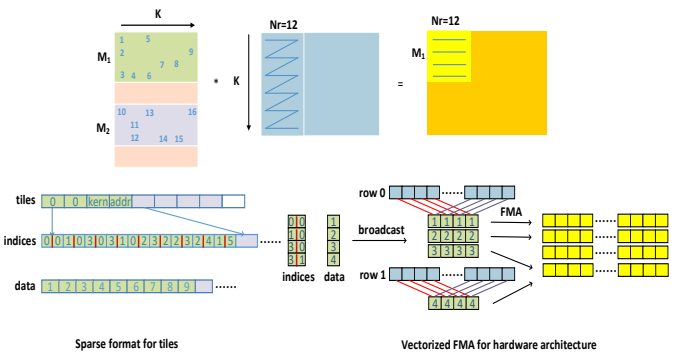


Fig. 3: Sparse kernel template. The numbers in the matrix represent non-zero elements. The black grids represent vector registers and we assume each vector register holds 4 elements. The sparse matrix is organized by three arrays: **tiles**, **indices** and **data**.

tile store from **indices**[**tiles**$[i][0]$] to **indices**[**tiles**$[i+1][0]$], and **tiles**$[i][2:]$ stores the address of the kernel for this tile. Slightly different from CSR, each entry in **indices** is divided into two parts bitwise. The higher part stores the row index of the non-zero element within the tile, while the lower part stores its column index, facilitating the retrieval of the vector register used to store computation results. In our experiments, the type of **indices** is **int32_t** for single precision data and **int64_t** for double precision data, with the upper 5 bits used for the row index and the remaining lower bits for the column index. **data** stores the value of each non-zero element in the tile and corresponds to the entries in **indices**.

```
// template parameters
const int Mi = 4, N = 64, Nr = 12, d = 4, si = 2;
int j;
for (j = 0; j <= (N - Nr); j += Nr) {
int32x4_t loc;
float32x4_t sp, bd0, bd1;
// Following registers are determined by Mi and Nr
float32x4x3_t ds0, ds1, res0, res1, res2, res3;
int col_prev = -1;
for (int t = tiles[i][0]; t < tiles[i + 1][0]; t += d) {
    sp = load(&data[t]), loc = vload(&indices[t]);
    for (int m = 0; m < d; m += si) {
        int idx0 = loc[m] >> 27;
        int c0 = loc[m] & 0x7ffffff;
        bd0 = broadcast(sp[m]);
        if (c0 > col_prev) ds0 = vload(&Xpack[c0][j:j+Nr]);
        switch (idx0) case 0: res0 = vFMA(res0, bd0, ds0);
        // omit other cases: 1, 2, 3
        // next FMA
        int idx1 = loc[m + 1] >> 27;
        int c1 = loc[m + 1] & 0x7ffffff;
        bd1 = broadcast(sp[m + 1]);
        if (c1 > c0) ds1 = vload(&Xpack[c1][j:j+Nr]);
        else ds1 = ds0;
        switch (idx1) case 0: res0 = vFMA(res0, bd1, ds1);
        // omit other cases: 1, 2, 3
        ds0 = ds1, col_prev = c1;  // next iteration
    }}
// write back to output
int r = tiles[i][1];
vstore(&Y[r + 0][j], res0); vstore(&Y[r + 1][j], res1);
vstore(&Y[r + 2][j], res2); vstore(&Y[r + 3][j], res3); }
// handling the boundary of remaining 4 = (N - j)
...
```

Listing 1: An instantiation of kernel template on **FP32** precision with an unrolled vectorized FMA

2) Kernel Template: Kernel Parser can determine the SIMD ISA and vector register length of the backend CPU architecture in advance. To implement high-performance sparse kernel, Kernel Parser outlines kernel template that utilizes SIMD instructions. For convenience, we assume d represents the number of elements each vector register can hold, the size of the i-th tile in A is (M_i, K), and the tile size in X is (K, N_r). Since the non-zeros of each tile are stored contiguously, every d non-zeros can be loaded into a vector register with a single load instruction. The computation involves multiplication of two tiles, requiring one vector register to store the broadcast values and N_r/d vector registers to store a row of the dense tile, with results maintained in $M_i \times N_r/d$ vector registers. N_r/d vectorized FMA operations are then performed on these three types of vector registers. Moreover, the corresponding indices are loaded into a vector register to determine which row of the dense tile to load and where to store the results in the vector registers. To maximize instruction-level parallelism, the template unrolls the for-loop associated with FMA operations by a factor of s_i, as shown in Listing 1.

3) Optimization Goal: The sparse kernel for the i-th tile is derived from kernel template by CodeGen module and is determined by the parameters M_i, N_r, d and s_i. Taking Listing 1 as an example, we assume the iteration over non-zeros of the i-th tile goes through T times (Line 10), the CMR can be calculated as follows:

$$\begin{aligned} \text{CMR} &= \frac{\sum_{t=1}^{T} N_r \cdot d}{\sum_{t=1}^{T} [1 + 1 + N_r \cdot c(t)/d]} \\ &= \frac{d^2}{2d/N_r + \sum_{t=1}^{T} c(t)/T} \end{aligned} \tag{1}$$

where $c(t)$ represents at the t-th iteration, the number of distinct column indices in vector register loaded from **indices** array (Line 11), which determines the frequency of memory accesses to the rows of the dense tile (Lines 16 and 23). Additionally, since the number of vector registers is limited in each core, the vector registers used in our sparse kernel must satisfy the following constraints:

$$\begin{cases} 1 + 1 + s_i + \frac{N_r}{d} s_i + \frac{M_i N_r}{d} \leq \#\text{vec_reg} \\ s_i \geq 1, d \ \% \ s_i == 0, N_r \ \% \ d == 0 \end{cases} \tag{2}$$

where "1+1" indicates vector registers to store elements in **indices** and **data**, and the number of vector registers to store broadcasting values and dense tiles is multiplied by s_i due to the unrolling of for-loop (Lines 6 and 8). The optimization goal of sparse kernel is to maximize CMR while also maximizing s_i, subject to the constraints in (2). From equation (1), larger N_r and smaller $\sum_{t=1}^{T} c(t)/T$ contribute to greater CMR. In our experiments, we set $N_r = 16$ and 12 on x86 and ARM processors, respectively. The value of $\sum_{t=1}^{T} c(t)/T$ is related to the sparse tile size M_i and will be discussed later. After determining M_i and N_r, the maximum s_i is calculated as follows:

$$\begin{cases} s_i \leq (\#\text{vec_reg} - 2 - \frac{M_i N_r}{d})/(1 + \frac{N_r}{d}) \\ s_i \geq 1, d \ \% \ s_i == 0 \end{cases} \tag{3}$$

Algorithm 2 Tile selection

Input: CSR $A[M][K]$
Output: #tiles N_t and row ranges $\{(S_1, E_1),...,(S_{N_t}, E_{N_t})\}$

1: $nnz[i] \leftarrow$ **ptr**$[i+1] -$ **ptr**$[i]$ for $i = 0, 1, ..., M-1$
2: $N_t \leftarrow 0$, R $\leftarrow \emptyset$
3: vis$[M] \leftarrow \{0, ..., 0\}$ ▷ We assume vis[-1] = vis[M] = 1
4: **repeat**
5: $N_t \leftarrow N_t + 1$
6: $r \leftarrow \arg\max_i(nnz[i]$ & vis$[i] == 0)$
7: vis$[r] = 1$
8: $S_{N_t} \leftarrow r$, $E_{N_t} \leftarrow r + 1$
9: best_val $\leftarrow \sum_{t=1}^{T} c(t)/T$ based on the sparse tile with shape $(E_{N_t} - S_{N_t}, K)$ and row range (S_{N_t}, E_{N_t})
10: **repeat**
11: $S = (\text{vis}[S_{N_t} - 1] == 0)\ ?\ S_{N_t} - 1 : S_{N_t}$
12: $E = (\text{vis}[E_{N_t}] == 0)\ ?\ E_{N_t} + 1 : E_{N_t}$
13: $s_1 \leftarrow \sum_{t=1}^{T} c(t)/T$ based on the sparse tile with shape $(E_{N_t} - S, K)$ and row range (S, E_{N_t})
14: $s_2 \leftarrow \sum_{t=1}^{T} c(t)/T$ based on the sparse tile with shape $(E - S_{N_t}, K)$ and row range (S_{N_t}, E)
15: **if** $s_1 \geq \max\{s_2, \text{best_val}\}$ **then**
16: best_val $\leftarrow s_1$, $S_{N_t} \leftarrow S$, vis$[S_{N_t}] = 1$
17: **else if** $s_2 \geq \max\{s_1, \text{best_val}\}$ **then**
18: best_val $\leftarrow s_2$, $E_{N_t} \leftarrow E$, vis$[E_{N_t} - 1] = 1$
19: **else if** best_val $> \max\{s_1, s_2\}$ **then**
20: break
21: **until** $M_{N_t} \leftarrow E_{N_t} - S_{N_t} + 1$ violates the constraint (2) and (3)
22: R.append(S_{N_t}, E_{N_t})
23: **until** $\forall i \in [0, M-1]$, vis$[i] = 1$
24: **return** N_t, R

```
const int NNZ = 16, Nt = 2;
int t_info[Nt][4] = {{0, 4, 0xb23e0854, 0xaaaa}, {6, 9, 0
    xb23e086c, 0xaaaa}};
int tiles[Nt][4], indices[NNZ], off = 0;
float data[NNZ];
for (int i = 0; i < NTile; i++) {
    tiles[i][0] = off; tiles[i][1] = t_info[i][0];
    tiles[i][2] = t_info[i][2]; tiles[i][3] = t_info[i][3];
    // first: idx of csr_indices, second: row idx
    // comparing order: csr_indices[first], row idx
    MinHeap<int, int> heap;
    for (int j = t_info[i][0]; j < t_info[i][1]; j++) {
        heap.push({csr_ptr[j], j});
    }
    while (!heap.empty()) {
        int idx, row = heap.pop();
        indices[off] = csr_indices[idx];
        data[off++] = csr_data[idx];
        if (idx + 1 < csr_ptr[row + 1]) {
            heap.push({idx + 1, row});
        }
    }
}
```

Listing 2: An instantiation of data layout transformation template

B. Sparse Parser

Since kernel template is based on tiles from sparse matrix, Sparse Parser aims to select the sparse tiles so that the i-th tile, with shape (M_i, K), minimizes $\sum_{t=1}^{T} c(t)/T$ in equation (1) to achieve optimized CMR.

To address this, Sparse Parser first calculates the number of non-zeros in each row (e.g., **ptr**[i+1]-**ptr**[i] in CSR format), then adopts a greedy strategy to choose tiles, as shown in Algorithm 2. The algorithm takes sparse storage format like CSR as input and returns the number of tiles N_t along with their row ranges $\{(S_1, E_1),...(S_{N_t}, E_{N_t})\}$, where (S_i, E_i) indicate the range of the i-th tile. Intuitively, a tile composed of rows with more non-zero elements is likely to reduce the number of distinct columns in the vector register and increase the number of for-loop iterations, resulting in a smaller $\sum_{t=1}^{T} c(t)/T$. Thus, the algorithm begins with the available row that has the maximum number of non-zeros when identifying a new tile (Lines 6∼8). The algorithm tentatively expands the current tile by adding its last row or next row, then simulates the procedure of sparse kernel on the expanded tile to calculate $\sum_{t=1}^{T} c(t)/T$. The algorithm records the best result and expands the tile accordingly. If the calculations from the tentatively expanded tile are less than the recorded best result, or if expanding by one row would violate the constraint in (2), the shape and row range of the current tile are determined, and the algorithm proceeds to find the next tile.

C. Memory Optimizer

1) Data Layout Transformation: Conventionally, sparse matrices are stored in CSR or CSC formats. To adapt the storage format to our sparse kernel, Memory Optimizer outlines a template for data layout transformation, which is later instantiated by the CodeGen module. Listing 2 illustrates the instantiation of transformation template from CSR format shown in Figure 3. Variables such as **NNZ, Nt, t_info** are template parameters populated at CodeGen stage. **t_info**[i][0] and **t_info**[i][1] store the row range of the i-th sparse tile, while **t_info**[i][2:] stores address of the specific sparse kernel to be invoked during the computation of the sparse tile. Memory optimizer employs min heap to ensure the non-zeros in the tile are organized in column-major reorder.

2) Dense Matrix Packing: The above descriptions focus on the optimization of the sparse matrix, but optimizing the data layout of the dense matrix is also important. Memory Optimizer also outlines a template for packing dense matrix into tiles with shape (K, N_r). The benefit of packing is to improve the locality of dense tiles in sparse kernel. The template packs dense matrix as shown in Figure 4. Specifically, each thread is responsible for packing one row. Since both the dense matrix and tiles are typically stored in row-major order, this row packing effectively leverages cache locality.

D. Template Assembler

So far, there are three types of templates proposed by SpMMTC. Since these templates are separate, Template Assembler assembles them into two types of functions: sparse matrix definition (**SpMat**) and **SpMM** kernel, as shown in Figure 2. The specific code for these functions is generated by the CodeGen module. The **SpMat** function internally utilizes the

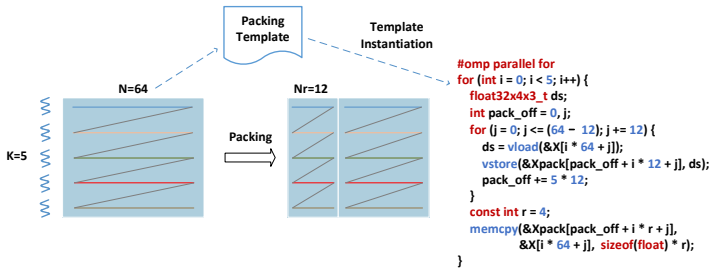


Fig. 4: Packing of a dense matrix and instantiation of a template with dimensions 5 × 64 and **FP32** precision. Each thread is responsible for packing one row.

data layout transformation code to convert the original sparse format (e.g., CSR, CSC) into the structures **tiles**, **indices**, and **data**, as previously described, and returns a handler specifying these structures. Users must call this function before executing the **SpMM**. When invoking the **SpMM** function, users provide the handler and the dense matrix as inputs. The function first packs the dense matrix and then calls the appropriate sparse kernel based on the current tile being processed.

E. Code Generation

The CodeGen module generates code to implement the **SpMat** and **SpMM** functions defined by the Template Assembler. For dense matrix packing, the code can be directly generated based on the dimensions of the dense matrix and the packing template. For sparse matrices, the sizes of each tile are determined using Algorithm 2, where the size of the i-th tile with a row range (S_i, E_i) is calculated as $M_i = E_i - S_i$. CodeGen then generates transformation code using the data layout transformation template proposed by the Memory Optimizer, as shown in Listing 2. Furthermore, based on the constraint in (3), SpMMTC selects the maximum unroll factor s_i to generate code with kernel template. The generated code consists of C/C++ source files, and the corresponding functions are compiled into a dynamic library, enabling users to invoke them at runtime.

IV. EXPERIMENT

A. Setup

We evaluate SpMMTC on x86 and ARM multi-core architectures: Intel Xeon Gold 5120, Kunpeng 920 (KP920) and Raspberry Pi 4 (RPi 4). In our experiments, the two servers are equipped with 56 and 64 cores, respectively, each with 256 GB of RAM, while the embedded device has 4 cores and 4 GB of RAM. We implement SpMMTC in C/C++, and the generated code is compiled using GCC with the `-O3` optimization flag.

B. Benchmarks

We conduct SpMM evaluations on two workloads: (I) sparse matrices in scientific computation and (II) sparse matrices in deep learning. For SpMM in scientific computation, we select datasets as shown in Table I, and for SpMM in deep learning, we use sparse matrices of ResNet50 model with magnitude pruning and random pruning [22] provided by deep learning

TABLE I: Sparse matrices for SpMM evaluation. These datasets are available at https://sparse.tamu.edu/ and https://networkrepository.com

Name	rows	nnz
Flickr	89,250	899,756
com-Youtube	1,134,890	5,975,248
GAP-road	23,947,347	57,708,624
Reddit	232,965	114,615,892
Bump_2911	2,911,419	127,729,899
vas_stokes_4M	4,382,246	131,577,616
nlpkkt160	8,345,600	225,422,112
Queen_4147	4,147,110	316,548,962
mycielskian19	393,215	903,194,710
uk-2005	39,459,925	936,364,282

matrix collection (DLMC) [23]. The following state-of-the-art libraries or compilers are used as baselines: 1) MKL, a linear algebra library optimized for x86 architectures by Intel; 2) ASpT [9], a high-performance sparse computation library that enhances data reuse by reordering rows; 3) TACO, a tensor compiler for linear algebra; 4) TVM, a tensor compiler with auto-tuning capabilities; and 5) XNNPack [24], a deep learning library providing SpMM interface.

C. Performance Analysis

1) Workload I: We evaluate these methods on SpMM using both FP32 and FP64 precisions, setting the columns of dense matrices (N) to 512. The experiments are performed in a multi-thread environment (56 threads on Intel and 64 threads on KP920) and their results are shown in Figure 5. ASpT demonstrates superiority over other baselines by reordering rows of sparse matrices before execution, thereby improving data reuse. Among the two tensor compilers, TVM outperforms TACO in most datasets due to its auto-tuning capability for generated kernels. On the Intel server, SpMMTC achieves up to 419.52 GFLOPs (FP32) and 200.78 GFLOPs (FP64), surpassing the baselines with an average speedup of 2.46x over MKL, 1.30x over ASpT, 3.11x over TACO, and 2.97x over TVM. On the ARM server, where XNNPack is used instead of MKL, SpMMTC achieves up to 299.65 GFLOPs (FP32) and 177.79 GFLOPs (FP64), outperforming all baselines with an average speedup of 1.99x over XNNPack, 1.48x over ASpT, 2.08x over TACO, and 1.95x over TVM, similar to its performance on the x86 architecture. Additionally, SpMMTC selects each optimal tile in at most 28 iterations on x86 architectures and 8 iterations on ARM architectures (Lines 10∼21 in Algorithm 2), making it more efficient than other tensor compilers' auto-tuning process.

2) Workload II: Unlike scientific computation, matrix dimensions in deep learning are smaller. We also evaluate SpMMTC performance under this scenario on Intel and KP920 servers using FP32 precision. Figure 6 shows the evaluation results on the sparse matrices of ResNet50. The model is pruned under varying sparsity levels with magnitude pruning and random pruning, comprising a total of 540 matrices.

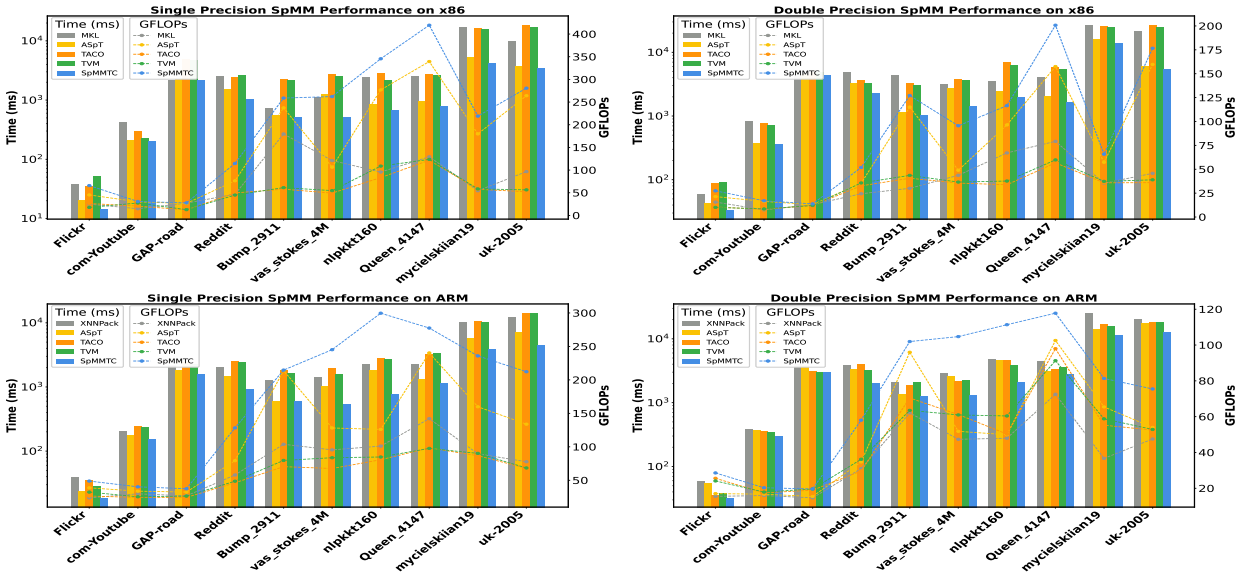


Fig. 5: SpMM performance of *Workload I* on x86 (Top) and ARM architectures (Bottom).

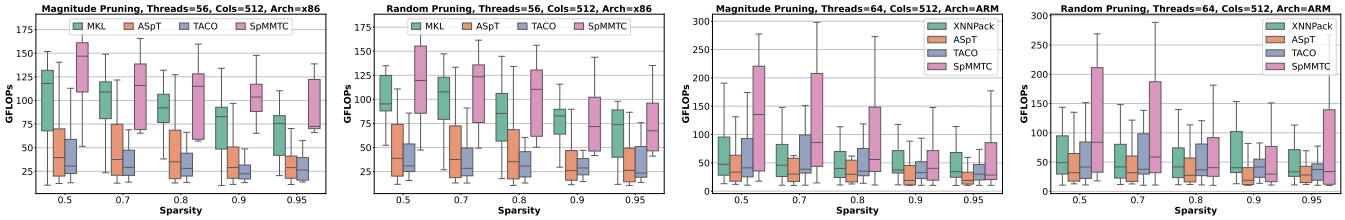


Fig. 6: SpMM performance of pruned ResNet50 on x86 and ARM architectures.

These matrices come from sparse filters, where convolution operations can be converted into matrix multiplications with im2col operation. The evaluations are conducted in a multi-thread environment, with the columns of the dense matrix set to 512 as well. The experimental results show that the throughput of these methods is affected by sparsity. As sparsity increases, their throughput decreases accordingly. In *Workload I*, ASpT demonstrates advantages over other baselines, but it is less efficient for deep learning computation workloads. SpMMTC outperforms all baselines on both architectures, achieving an average speedup of 1.21x over MKL, 1.84x over ASpT, 1.39x over XNNPack, and 2.28x over TACO.

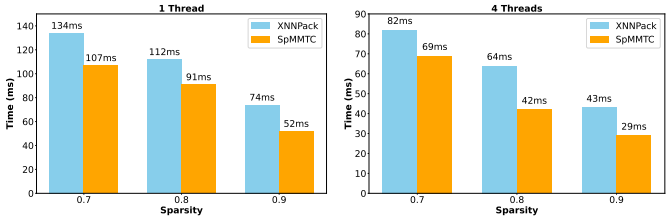


Fig. 7: End-to-end inference time of pruned MobleNetV1 on RPi 4 device.

3) MobileNetV1 on edge: MobileNetV1 is a well-known DL model widely used in image classification and object detection on mobile or edge devices. Optimizing the model's performance can help improve user experience and battery energy consumption. To evaluate SpMMTC performance on MobileNetV1, we replace sparse convolution operations with im2col operations using our SpMM and evaluate end-to-end inference under different sparsity levels on the RPi 4 device. We compare SpMMTC with XNNPack, and the results are shown in Figure 7. SpMMTC achieves a speedup of up to 1.52x over XNNPack under sparsity levels ranging from 0.7 to 0.9.

V. Conclusion

In this paper, we propose SpMMTC, a sparse tensor compiler designed to automatically optimize and generate SpMM code. SpMMTC takes a sparse matrix and the dimensions involved in SpMM as inputs, analyzes the sparse patterns of the matrix, and generates high-performance kernels specialized for SpMM with identical dimensions and sparse matrix, leveraging vectorized acceleration on multi-core architectures. In our experiments, SpMMTC outperforms state-of-the-art methods, including MKL, ASpT, TACO, and TVM.

Acknowledgement

This work was partially supported by National Key R&D Program of China (Grant No. 2023YFB3001805), CCF-Huawei Populus Grove Fund (CCF-HuaweiSY202407), Programs of NSFC (Grant No. U21A20461, U24A20235, 62321003, 62227808), China Postdoctoral Science Foundation (2024M750879), China National Postdoctoral Program for Innovative Talents (BX20240111), and Changsha Natural Science Foundation (kq2502277).

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Red Hook, NY, USA, 2017, p. 6000–6010.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[4] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 907–922.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[6] M. Yang, H. Wang, Z. Wei, S. Wang, and J.-R. Wen, "Efficient algorithms for personalized pagerank computation: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[7] H. Chen, Y. Bei, Q. Shen, Y. Xu, S. Zhou, W. Huang, F. Huang, S. Wang, and X. Huang, "Macro graph neural networks for online billion-scale recommender systems," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 3598–3608.

[8] A. Hajiveiseh, S. A. Seyedi, and F. A. Tab, "Deep asymmetric nonnegative matrix factorization for graph clustering," *Pattern Recognition*, vol. 148, p. 110179, 2024.

[9] C. Hong, A. Sukumaran-Rajam, I. Nisa, K. Singh, and P. Sadayappan, "Adaptive sparse tiling for sparse matrix multiplication," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019, pp. 300–314.

[10] Y. Wang, B. Feng, Z. Wang, G. Huang, and Y. Ding, "{TC-GNN}: Bridging sparse {GNN} computation and dense tensor cores on {GPUs}," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 149–164.

[11] L. Wilkinson, K. Cheshmi, and M. M. Dehnavi, "Register tiling for unstructured sparsity in neural network inference," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 1995–2020, 2023.

[12] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–29, 2017.

[13] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[14] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen *et al.*, "Ansor: Generating {High-Performance} tensor programs for deep learning," in *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, 2020, pp. 863–879.

[15] F. G. Van Zee and R. A. Van De Geijn, "Blis: A framework for rapidly instantiating blas functionality," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 3, pp. 1–33, 2015.

[16] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *Acm Sigplan Notices*, vol. 48, no. 6, pp. 519–530, 2013.

[17] P. Liu, A. J. Root, A. Xu, Y. Li, F. Kjolstad, and A. J. Bik, "Compiler support for sparse tensor convolutions," *Proc. ACM Program. Lang.*, vol. 8, no. OOPSLA2, 2024. [Online]. Available: https://doi.org/10.1145/3689721

[18] A. Dias, L. Anderson, K. Sundararajah, A. Pelenitsyn, and M. Kulkarni, "Sparseauto: An auto-scheduler for sparse tensor computations using recursive loop nest restructuring," *Proc. ACM Program. Lang.*, vol. 8, no. OOPSLA2, Oct. 2024. [Online]. Available: https://doi.org/10.1145/3689730

[19] J. Liu, Z. Zhao, Z. Ding, B. Brock, H. Rong, and Z. Zhang, "Unisparse: An intermediate language for general sparse format customization," *Proc. ACM Program. Lang.*, vol. 8, no. OOPSLA1, Apr. 2024. [Online]. Available: https://doi.org/10.1145/3649816

[20] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*, pp. 167–188, 2014.

[21] W. Yang, J. Fang, D. Dong, X. Su, and Z. Wang, "Libshalom: Optimizing small and irregular-shaped matrix multiplications on armv8 multi-cores," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.

[22] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 359–371.

[23] T. Gale, M. Zaharia, C. Young, and E. Elsen, "Sparse gpu kernels for deep learning," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14.

[24] Google, "Xnnpack," https://github.com/google/XNNPACK.

Synthesis of CFET Cell Library Leveraging Backside Metal Routing

Ting-Xin Lin, Yih-Lang Li

Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

jack25946705@gmail.com, ylli@cs.nycu.edu.tw

ABSTRACT

As technology nodes continue to shrink, Complementary FET (CFET) structures, which stack PMOS and NMOS together, have emerged as a promising candidate for next-generation technology. Due to the reduction in routing tracks, the insertion of dummy polys to increase routing resources and the use of M2 during the synthesis of CFET standard cells have become more inevitable. These two factors make block-level routing significantly more challenging. To address this, we introduce the methods to utilize the backside (BS) routing resources at the CFET standard cell synthesis stage and efficiently handle CFET transistor folding. To the best of our knowledge, this is the first work to consider BS routing resources at the cell synthesis stage and efficiently address transistor folding in the CFET stacked structure. In the transistor folding and placement stages, we utilize Euler paths to estimate the lower bound of contacted poly pitch (CPP) and apply dynamic programming (DP) to calculate the frontside minimum required tracks (FMRT) of BS-resource-aware placements. The subsequent satisfiability modulo theories (SMT) approach determines which tracks the devices occupy and completes cell routing. Experimental results show that compared to previous work [11], we achieve reductions of 1%, 45%, and 19% in #CPP, #M2 tracks, and runtime, respectively.

1. INTRODUCTION

Due to the scaling of semiconductor technology, device structures like FinFET face limitations due to the separation between two types of devices. Therefore, CFET structures, which stack both device types together, have become a mainstream approach for the next generation technology. However, the reduced number of available routing tracks and complex design rules impose significant challenges on both cell-level and block-level routing in the frontside (FS) back end of line (BEOL). Backside power delivery networks (BS PDN) [1] have demonstrated IR-drop reduction in both 2D and 3D designs. [2] further introduces BS signal routing to enhance performance and power efficiency at the 2nm technology node. Inspired by [2], we apply BS routing resources, including nano-Through-Silicon Vias (nTSVs) and backside metals (BSMs), during the CFET cell synthesis stage to alleviate routing congestion on the FS of the chip. Fig. 1 illustrates the impact of utilizing BS resources. The signal *Y* is routed through M0 metals, nTSVs and BSMs (Fig. 1(a)) without the use of M1 and M2 wires, whereas in Fig. 1(b), two M1 wires and one M2 track are required to complete the routing. Compared to Fig. 1(b), Fig 1(a) saves two M1 wires and one M2 track.

In CFET structures, the contact allocation for the bottom device signal can be blocked by the top device signal in the same column when they differ and partially overlap in the *z*-direction, with each source, drain, or gate signal occupying a separate column. This issue becomes more severe when the top device obstructs more routing tracks, and it is evident that a larger top device blocks more tracks in a column. Transistor folding, which divides a larger transistor into multiple smaller

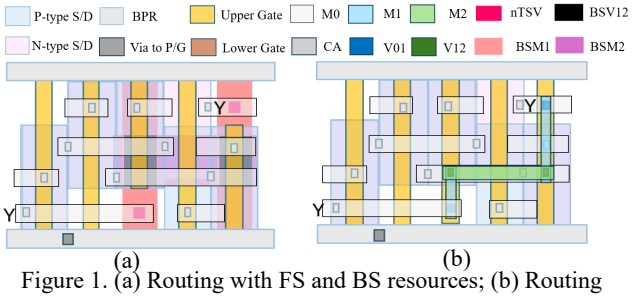


Figure 1. (a) Routing with FS and BS resources; (b) Routing with only FS resources.



Figure 2. (a) CFET folding; (b) Simple folding.

transistors (fingers), helps release more routing resources for the bottom device. However, transistor folding in CFET stacked structures is simplified, with previous works mainly focusing on planning structures. In the non-stacked structures, [3] addresses both transistor folding and placement simultaneously through search tree exploration. [4] applies ILP to determine the size of each finger and minimize cell area. [5] estimates the cell width using an Euler graph and presents search tree pruning techniques. The main difference of folding problems between in stacked and non-stacked structures is that, in stacked structures, we need to account for the space required to allocate contacts for bottom signals to access M0. Besides, measuring this space is critical, which is not a concern in non-stacked structures. Fig. 2 highlights the significance of the folding issue in CFET. In Fig. 2(a), the bottom device signal *S* is successfully routed via M0 through the diffusion contact access. However, as shown in Fig. 2(b), the contact location at the cross position for the bottom signal S fails due to being blocked by the top device, ultimately causing routing failure.

Previous works have addressed transistor placement with various approaches. For instance, [6] applies a bipartite graph and graph-based chaining processes to maximize diffusion sharing, while [7] uses DP to generate routability-aware transistor placement. For cell routing, common algorithms like maze, Boolean satisfiability (SAT), integer linear programming (ILP), mixed integer linear programming (MILP) and network flow are employed. The proposed ILP M0 planning model in [7] systematically selects M0 segments, followed by implicitly adjustable grid map maze routing to produce higher-quality cell layouts that are compliant with ASAP 7nm technology compared to the handcrafted cell library. The SAT approach in [8] generates cell layouts free of design-rule violations and minimizes undesired patterns while considering design-for-manufacturing. In [9], a spare M1 track is reserved to enhance

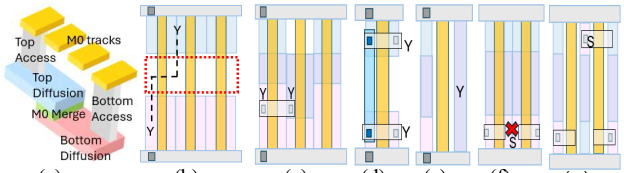


Figure 3. (a) A 3D view; (b) (c) FinFet and CFET structures; (d) With real MOS widths; (e) With simplified MOS widths; (f) Our routing model; (g) The simplified routing model

pin accessibility, and MILP is used to concurrently complete cell routing and pin allocation. In previous CFET cell library synthesis works, [10] simultaneously handles placement and routing using SMT. However, due to the high complexity of the model, generating large cells faces scalability challenges. Moreover, since placement and routing are explored simultaneously, the process may overlook placements that could lead to more efficient routing due to time constraints. The approach [11] demonstrates that traditional two-stage cell synthesis, cell placement and cell routing, can not only outperform [10] in terms of the cell area, but also additionally resolve the scalability issue. Nonetheless, these CFET works ignore transistor widths specified in SPICE netlists, which will be discussed in more detail in Section 2. Synthesized standard cell layouts must comply with SPICE netlists to ensure cell functionality and characteristics, such as performance, and power consumption.

This is the first work to introduce BS routing resources during the cell synthesis and to handle the folding issue for transistors with large width in CFET stacked structures. Our main contributions are as follows: (1) The CPP lower bound of a folding outcome is calculated using Euler paths, followed by a DP-based FMRT calculation of BS-resource-aware placements; (2) The proposed SMT formulation decides the tracks used by the devices and performs cell routing considering BS resources; and (3) Compared to previous work [11], we achieve reductions of 1%, 45%, and 19% in #CPP, #M2 tracks, and runtime, respectively.

In Section 2, we introduce the CFET cell structure and highlight the differences in the routing model between our approach and previous works. Section 3 addresses CFET folding, BS-resource-aware placement, and the SMT formulations for cell routing. Section 4 presents the experimental results and Section 5 concludes this work.

2. PRELIMINARY

2.1 CFET Stacked Structures

In CFET with buried power rails (BPRs), the middle region disappears due to stacking PMOS and NMOS together. The following two CFET routing features simplify connecting signals between two different types of devices, making it easier than in FinFET. First, a vertical wire connecting two diffusion signals, which overlap in the *z*-direction, is replaced by an M0-merge to interconnect them. Second, a Z-shaped wire can be eliminated by using a horizontal M0 wire to connect signals from both types of devices, provided their contact allocations are on the same routing track. Fig. 3(a) shows a 3D view of the M0-merge. The red dotted rectangle in Fig. 3(b) represents the middle region in FinFET structures, where the signal *Y* requires vertical wires. In contrast, Fig. 3(c) shows that no vertical wires are needed in CFET structures.

2.2 The Previous Routing Models

The previous work [11] simplifies both the routing model and MOS widths. There are at most two signals in a column. Since the two signals in a column are distinct and the bottom

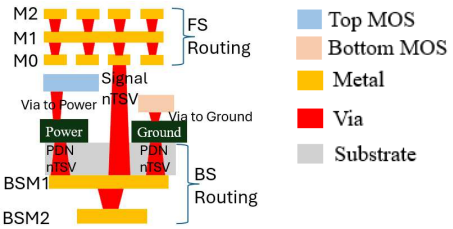


Figure 4. Layers on front and back sides.

signal requires a contact to access the upper M0 layer, contact placement for this signal is limited to either the top or bottom track to avoid the obstruction caused by the top device. However, if the top signal is VDD or VSS, and connects to power or ground rail, the bottom signal has only one available track. If the two signals in a column are identical, all tracks remain available for the bottom signal. On the contrary, the top signal can allocate the contact on any available track, regardless of whether the two signals in a column are distinct or not.

The layout style in [11] prioritizes routing simplicity over strict adherence to transistor width specifications. Transistor widths are treated as flexible and adjusted as needed to facilitate easier routing. For example, the DFFHQN cell in [11] does not follow the transistor width specification. Figs. 3(d) to 3(g) demonstrate why the cell design complying with transistor width specification is much harder than that in [11]. The layout style in Figs. 3(d) and 3(f) are ours, while those in Figs. 3(e) and 3(g) are from previous work [11]. Fig. 3(d) shows that the layout of two non-overlapping devices with small widths requires two M0 and one M1 wires to realize signal *Y*'s connection. M0-merge cannot be used in this case. In contrast, Fig. 3(e) demonstrates the flexible MOS widths allow the use of an M0-merge for routing signal *Y*. The assumption in [11] that two identical signals in the same column can always be routed with an M0-merge may fail under real MOS width specification.

In Fig. 3(f), top devices occupy the top three tracks, leaving only one bottom free track for the bottom signals, which produces a design rule violation (DRV) between signal *S* with two other M0 wires during M0 access. In Fig. 3(g), the routing model in [11] encourages signal *S* accessing the top M0 track, which results in misalignment between two diffusions in the top device; however, they should be aligned.

2.3 BS Routing Resources

BS routing resources, which are manufactured separately, are located under the substrate, while FS routing resources are on the substrate. Therefore, the metal pitch, metal width, and design rules for the BS are independent of the FS. To connect to the frontside metals (FSMs), specifically M0 in this work, BSMs utilize nTSVs that pass through the substrate. Careful placement of these nTSVs is crucial to avoid damaging any existing devices, such as poly or diffusion layers. In this work, we assume nTSVs can be placed on any track and in any column as long as there is no overlap with devices in the *z*-direction. According to our assumption, an nTSV occupies one grid on the FS and retains its size before passing through the substrate. As the number of BSM layers is configurable, there are two BSM layers in this work: backside metal 1 (BSM1) and backside metal 2 (BSM2). We assume that the BSM1 pitch is twice the CPP, and its width equals three times the M1 width. There is one BSM2 routing track, with a width three times that of M2 metal and spacing rules the same as M2. Fig. 4 shows a cross-sectional view of the layers on front and back sides.

2.4 Balanced Design-Rule-Check-Aware Remaining Pin Access (BDARPA)

The proposed BDARPA in [11] is an enhancement of remaining pin access (RPA) in [12]. BDARPA optimizes the RPA value of each IO pin evenly, while ensuring no violations occur when an M1 access point connects to an M2 wire during block-level routing. BDARPA provides a more accurate estimation of pin accessibility compared to RPA.

2.5 Graph-based Transistor Placement

In [13], there are four steps in transistor placement: clustering, pairing, chaining, and chain placement. Clustering uses common signals in PMOS and NMOS to divide the entire netlist into several sub-netlists. Pairing pairs the PMOS and NMOS with the same gate signal within the same clustering group. Chaining maximizes the number of shared diffusions using a depth-first search with a pruning technique. Finally, chain placement determines the permutation and orientation of the chains established in the previous step.

3. METHODOLOGY

The following discussion focuses on stacking P-type transistors on N-type transistors. The analysis can be extended to the configuration where N-type transistors are stacked on P-type transistors. Given a SPICE netlist, we first address CFET transistor folding and estimate the CPP lower bound of the folding results. The folding results with the lowest estimated value will proceed to the BS-resource-aware transistor placement stage. We utilize graph-based transistor placement from [13] to obtain placement results. Additionally, a diffusion column is considered undesirable if the top device's width in that column is not smaller than the number of routing tracks minus one. During the placement stage, we minimize CPP, FMRT, undesired diffusion columns, and then total wirelength. We also predict whether a column can accommodate an nTSV and calculate FMRT considering BS resources using a shortest path algorithm and DP. Finally, the SMT solver processes placement candidates, determining which tracks the devices occupy and completing cell routing.

3.1 CFET Transistor Folding

Transistor folding is the process of dividing a larger transistor into several smaller ones, called fingers. [11] briefly addresses CFET folding. Let tr_no_{M0} represent the number of M0 routing tracks. In [11], a transistor with a size greater than or equal to tr_no_{M0} is folded so that each finger has a size less than tr_no_{M0}, and the number of fingers is minimized. In our work, we not only fold transistors with sizes greater than or equal to tr_no_{M0}, but also fold top transistors with sizes greater than or equal to tr_no_{M0} minus one to explore more folding solutions. Since top devices block bottom devices, our goal is to create more room for the bottom by breaking down the top transistors. Additionally, the number of fingers in our approach ranges from $\lceil \frac{sz_{tran}}{tr_no_{M0}} \rceil$ to $\lceil \frac{sz_{tran}}{x} \rceil$ (where sz_{tran} represents the transistor size, and x is user-defined; in this work, x equals two). Each finger size is equal to or less than tr_no_{M0}.

We construct a folding table by enumerating all possible finger configurations for each transistor and estimate the CPP lower bound for each folding result. When a transistor is folded into several fingers, its signal is split into the corresponding number of signals. We then sum the number of each diffusion signal across all transistors, as only diffusion signals affect subsequent estimations. A signal is considered odd when its count is odd. For example, two P-type transistors, (*net*1, *A*, *Y*) and (*VDD*, *B*, *net*1), are both folded into three fingers. The number of diffusion signals for *net*1, *Y*, and *VDD* is six, three, and three, respectively. Using Euler graphs, the following formula to compute the number of diffusion breaks (n_{df_bk}) is obtained:

$$n_{df_bk} = \begin{cases} 0, \ if \ n_{o_sg} < 3 \\ \left\lfloor \frac{n_{o_sg}-1}{2} \right\rfloor, if \ n_{o_sg} \geq 3 \end{cases}, \quad (1)$$

where n_{o_sg} is the number of odd signals. We sum the number of fingers and diffusion breaks to estimate the CPP lower bound. P-type and N-type devices are estimated separately, and we choose the larger estimated value as the CPP for the entire cell, as the larger value obviously dominates. We proceed with transistor placement only for the folding results that have the lowest estimated value in the folding table.

It is also worth mentioning that during the pairing stage, we exclude two illegal pairing results. One occurs when the NMOS width occupies all tracks and the *VDD* signal has no room to connect down to the power rail. The other occurs when the PMOS width occupies all tracks, leaving the NMOS signal no room to connect up to M0 if necessary.

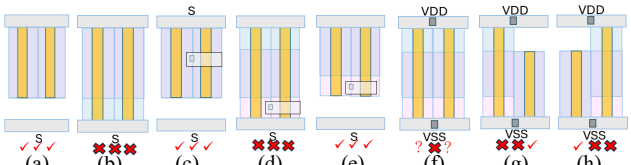


Figure 5. (a) - (e) n_{rm_trk} is 1, 0, 1, 0, and 1, respectively; (f) In case 4, n_{rm_trk} can only be used to predict the middlemost column, while the other two remain uncertain. (g) – (h) The left and right columns must be predicted by other abutting pairs. An nTSV can be placed by shifting the right (g) and left (h) devices, respectively.

3.2 BS-resource-aware MOS Placement

3.2.1 nTSV Column Prediction

After the pairing stage, we predict whether an nTSV can be placed on a column when abutting two transistor pairs. A local net refers to a net that is not *VDD*, *VSS*, or an IO signal and completes its routing without using metal, relying only on diffusion sharing.

We define the notations used in this section first. The number of remaining tracks on a column is n_{rm_trk}. wid_T and wid_B represent the larger width value of the two abutting top and bottom devices, respectively. n_{rm_trk} is calculated by subtracting the number of tracks that devices occupy on a column from the total number of M0 tracks, as placing an nTSV on devices will damage them. An n_{rm_trk} value greater than 0 implies that an nTSV can be placed on this column; otherwise, an nTSV cannot be placed.

In Fig. 5, a tick indicates that an nTSV can be placed in the column, a cross means the column cannot accommodate an nTSV, and a question mark represents uncertainty. We focus on discussing the three middle columns since the leftmost and rightmost columns must be predicted based on other abutting pairs.

Case 1: In Figs 5(a) and 5(b), the abutting bottom signal *S* is a local net. Since *S* does not need to connect up to M0, the bottom device can hide under the top device, potentially freeing up space for nTSV placement. In this case, $n_{rm_trk} = tr_no_{M0} -$

$max(wid_T, wid_B)$. In Fig. 5(b), there is no available column for placing an nTSV since n_{rm_trk} is 0.

Case 2: In Fig. 5(c), the abutting signals for top and bottom devices are the same (*S*), and the bottom *S* is not local. Even though S is not local, it also can use an M0-merge to connect to the top device, which then accesses M0. The bottom device can still hide under the top device. Here, $n_{rm_trk} = tr_no_{M0} - max(wid_T, wid_B)$.

Case 3: In Figs 5(d) and 5(e), the abutting signals for top and bottom devices differ, and the abutting bottom signal *S* is not local. Since *S* needs to allocate a contact to connect up to M0, at least one track must be reserved for contact allocation for *S*. In this case, $n_{rm_trk} = tr_no_{M0} - 1 - max(wid_T, wid_B - 1)$.

Case 4: In Figs 5(f) to 5(h), the abutting top and bottom signals are power and ground. Since both the power and ground signals need to connect to their respective power and ground rails, at least one top device and one bottom device must occupy the top and bottom tracks. n_{rm_trk} is calculated separately for both device types, with $n_{rm_trk} = tr_no_{M0} - wid_T - wid_B$.

To verify the accuracy of our nTSV column prediction, we also enumerate all possible track occupations of an MOS to check whether an nTSV can be placed on a column. We directly incorporate information about MOS track occupation in the following stages: pairing, chaining, and chain placement, which enlarges the placement solution space. Fig. 6 demonstrates the enumeration, pairing, and pairing abutment rules. In Fig. 6, a cross indicates an invalid pattern. Types 1 – 3 illustrate the partial enumeration with the top diffusion on different tracks. During pairing, the pairing pattern in Type 4 is invalid because the bottom device blocks the top device from connecting to the power rail, while Type 5 is valid. When abutting two transistor pairs, Type 6 is invalid as the top devices do not share their diffusion, while Type 7 is valid.

3.2.2 DP-based and BS-resource-aware FMRT

After the chaining and chain placement stages, we calculate the FMRT of a placement candidate (PC), considering BS resources to further reduce FMRT and ease M2 track usage in the subsequent routing stage. Fig. 7 depicts the routing results of two PCs, where Fig. 7(b) utilizes the BS resources while Fig. 7(a) cannot. Without considering BS resources, the evaluation of FMRT of two PCs are the same. However, after introducing BS resources, the FMRT in 7(b) becomes less than that in 7(a). Even though both consider BS resources, 7(b) further reduces FMRT by routing one signal using BS resources, while 7(a) cannot. Factors that impact FMRT when considering BS resources include the position of signals, and the number and position of nTSVs. We calculate the FMRT of a PC using a shortest path algorithm and DP.

For a PC, each signal has two ways to realize its shortest path. The first way uses only M0, while the second can utilize both M0 and BS resources. The shortest paths for a signal are determined in the absence of the shortest paths for other signals. All shortest paths are used in enumerating the FMRT. To identify the second shortest path, evaluating whether an nTSV can be placed on a column can help link the M0 and BS resources. Additionally, to ensure the BS resources are explored, their cost is set lower than that of the M0 resources. The path exploration space for each signal ranges from the leftmost to rightmost columns of its pins with slight expansion. The source and target for paths are set on the M0 layer.

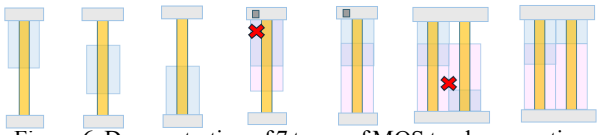
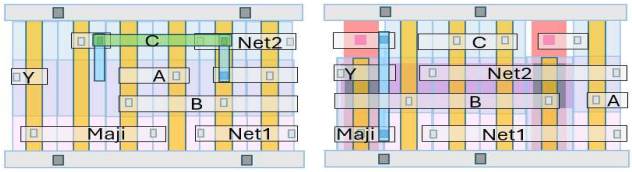


Figure 6. Demonstration of 7 types of MOS track occupations and the pairing and pair abutment rules, considering the information about MOS track occupation.



(a) (b)

Figure 7. Different PCs in a cell. (a) uses one M2 track; (b) uses one BSM2 track and no M2 usage.

Algorithm 1. BS-resource-aware FMRT calculation

Input: a given transistor placement candidate *p*
Output: FMRT of the given candidate *p*
1. Construct graph $\mathcal{G}_\mathcal{R}$ of the candidate *p*
2. **for** *i*=1 to $2^{|V(\mathcal{G}_\mathcal{R})|}$
3. $dp[i] \leftarrow S$
4. $M \leftarrow \emptyset$
5. FMRT_Recursive_Process($\mathcal{G}_\mathcal{R}$, *M*)

Figure 8. Algorithm of BS-resource-aware FMRT.

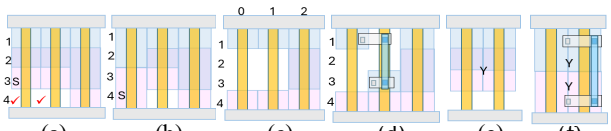
Algorithm 2. FMRT_Recursive_Process

Input: a graph $\mathcal{G}_\mathcal{R}$ and a node set *M*
Output: FMRT of the given graph $\mathcal{G}_\mathcal{R}$
1. **if** $\mathcal{G}_\mathcal{R}$ is $\emptyset$
2. **return** FMRT according to *M*
3. Pick a node *n* with the least node value and #neighbors
4. Push node *n* and its neighbors into queue *q*
5. **for** each node *s* in queue *q*
6. $\mathcal{G}_\mathcal{R}' \leftarrow \mathcal{G}_\mathcal{R}$ removes node *s*, its neighbors, and their edges
7. $M' \leftarrow M$ adds node *s*
8. $dp[\mathcal{G}_\mathcal{R}] \leftarrow$min($dp[\mathcal{G}_\mathcal{R}]$, FMRT_Recursive_Process($\mathcal{G}_\mathcal{R}'$, *M'*))
9. **return** $dp[\mathcal{G}_\mathcal{R}]$

Figure 9. FMRT_Recursive_Process.

After finding the paths, we construct a resource constraint graph, $\mathcal{G}_\mathcal{R}$, for the PC. If a signal uses BS resources in its second path, the signal has its respective node in $\mathcal{G}_\mathcal{R}$. If not, no node is created. If two signals' second paths overlap in the use of BS resources, an edge is created to link their respective nodes in $\mathcal{G}_\mathcal{R}$, indicating that the signals cannot use the BS resources simultaneously. We calculate the FMRT for each node in $\mathcal{G}_\mathcal{R}$ as its node value, assuming the corresponding signal uses its second path while the other signals choose their first paths for signal connections. The FMRT calculation focuses on the leftmost to rightmost columns of the corresponding signal's pins and equals the number of signal overlaps in the y-direction on M0.

DP is then applied to $\mathcal{G}_\mathcal{R}$ to determine which signals should utilize BS resources to minimize the FMRT of the PC. Fig. 8 and Fig. 9 illustrate our algorithm. In Fig. 8, we construct $\mathcal{G}_\mathcal{R}$, and $V(\mathcal{G}_\mathcal{R})$ represents the node set of $\mathcal{G}_\mathcal{R}$. There are $2^{|V(\mathcal{G}_\mathcal{R})|}$ possible combinations of BS signal usage. Each element of the array *dp* represents the FMRT when considering only the corresponding signals in the graph. We initialize all *dp* values to a large number *S* and set *M* as an empty node set. After initialization, a recursive procedure is called to calculate the FMRT of the PC. If $\mathcal{G}_\mathcal{R}$ is



(a) (b) (c) (d) (e) (f)

Figure 10. Device track occupation and SMT demonstration.

TABLE 1. NOTATIONS

S^{SN}/S^{N}	The subnet/net space
$TD_{i,t}/BD_{i,t}/G_{i,t}$	The bool variable indicates whether the i^{th} top diffusion/bottom diffusion/gate occupies track t.
$\mu_v^S(c,t,l,n)/\mu_e^S(c,t,l,n)$	The bool variable indicates whether the vertex/edge in c^{th} column, t^{th} track, l layer and S space is used by net n. A horizontal edge connects the c^{th} and $(c+1)^{th}$ columns. A vertical edge connects the t^{th} and $(t+1)^{th}$ tracks. A via edge connects the l^{th} and $(l+1)^{th}$ layers. S can be either SN or N.
$deg^{SN}(v)$	The integer variable indicates the number of used edges of vertex v in the SN space.

empty, we calculate and return the FMRT for the entire placement when the corresponding signals in M use their second paths, while other signals use their first paths for connections (Fig. 9, Lines 1-2).

We select a node n in $\mathcal{G}_{\mathcal{R}}$ with the smallest node value and the fewest neighbors to reduce recursion calls (Line 3). Node n and its neighbors are pushed into a queue, and we iterate through each node in the queue to call the recursive procedure, minimizing the dp value at each iteration (Lines 4-8). Node n and its neighbors are not selected simultaneously due to the overlap in BS resources. We iterate through all nodes in the queue to explore all possible solutions. Once a node s is selected, $\mathcal{G}_{\mathcal{R}}'$ ($\mathcal{G}_{\mathcal{R}}$ with node s, its neighbors, and their edges removed) and M' (M with node s added) are passed down to the next recursive level (Lines 6-8).

3.3 SMT-based Cell Routing

There are two important tasks for our SMT algorithm: determining the track occupation of devices and managing signal connections while considering BS resources, both of which are determined simultaneously. Due to the additional consideration of transistor width, which was simplified in previous work, the proposed SMT formulation needs to accurately handle device track occupation, diffusion sharing, diffusion merge, and contact allocation. The diffusion track occupation differs between Fig. 10(a) and Fig. 10(b), which shows that device track occupation impacts signal connections on both sides. For the FS, contacts can only be allocated on devices. Therefore, in Fig. 10(a), the signal S can only access M0 on the third track, whereas in Fig. 10(b), it can access both the third and fourth tracks. For the BS, nTSVs cannot overlap with devices in the z-direction. As a result, in Fig. 10(a), there are two ticks indicating positions where an nTSV can be placed, whereas in Fig. 10(b), no such ticks are present.

Table 1 lists the notations used in the SMT formulations. When a net is a multiple-pin net, we decompose it into several two-pin nets, arranged from the leftmost to the rightmost column. Even if the top and bottom signals in a column are the same, we still use a two-pin net to represent them. The solution space for a two-pin net ranges from its leftmost to rightmost columns, with slight expansion. We connect each two-pin net in the subnet space, and the routing results in the subnet space are projected onto the net space. To expedite the process, we first define a net order, which is sorted in descending order based on the signal span length. All nets are routed using an M0 pattern in the initial iteration, and one more net is routed using multi-commodity flow (MCF) in subsequent iterations until the routing is complete. T represents the M0 routing track set, and $|T|$ is four in this work. Our optimization objectives, in lexicographic order, are to minimize the number of M2 tracks, maximize BDARPA as defined in [11], and minimize the total wirelength.

3.3.1 MOS Constraints

wid represents the number of tracks that the diffusion of a device occupies in a column, with one set of *wid* continuous variables being true for the device. We enumerate all possible track occupations for the diffusion. Eq. (2) defines the diffusion of the i^{th} top device, with its width being two. Other devices and widths are treated similarly. In Fig. 10(c), the encoded result for the top diffusion number zero/two is {true, false, false, false}/{true, true, true, false}.

$$(TD_{i,0} \wedge TD_{i,1} \wedge \sim TD_{i,2} \wedge \sim TD_{i,3}) \vee (\sim TD_{i,0} \wedge TD_{i,1} \wedge TD_{i,2} \wedge \sim TD_{i,3}) \vee (\sim TD_{i,0} \wedge \sim TD_{i,1} \wedge TD_{i,2} \wedge TD_{i,3}) \quad (2)$$

In a device, if the diffusion occupies a track, the gate will also occupy the corresponding track in its column. We represent i^{th} gate using Eq. (3). Both the top and bottom devices are considered due to the same gate signal. Track occupation for the gate is also enumerated.

$$(TD_{i,t} \rightarrow G_{i,t}) \wedge (BD_{i,t} \rightarrow G_{i,t}) \quad (3)$$

3.3.2 Connectivity Constraints

M0, M2, and BSM2 are used for horizontal routing, while M1 and BSM1 are used for vertical routing. Eq. (4) defines a horizontal edge, with vertical and via edges treated similarly.

$$\mu_e^{SN}(c,t,l,n) \rightarrow \mu_v^{SN}(c,t,l,n) \wedge \mu_v^{SN}(c+1,t,l,n) \quad (4)$$

$S^{SN(a,b)}$ represents a two-pin net connecting pins a and b in the subnet space. Using the MCF, we restrict the degree of vertices in Eq. (5) and Eq. (6) to complete the two-pin net connection. In Eq. (5), the pin may be a floating pin. V is the vertex set of $S^{SN(a,b)}$.

$$\wedge_{p=a,b}(\sum_{\{v|v\in p\}} deg^{SN(a,b)}(v) == 1) \quad (5)$$

$$\wedge_{v\in V, v\notin a, v\notin b}\left(deg^{SN(a,b)}(v) == 0 \vee deg^{SN(a,b)}(v) == 2\right) \quad (6)$$

3.3.3 Diffusion Sharing Constraints

There must be at least one track abutment between the diffusions of adjacent devices, or a two-pin net can be used to connect them. Figs. 10(c) and 10(d) illustrate these two options. Eq. (7) defines the first option for the connection between the i^{th} and $(i+1)^{th}$ top devices.

$$\vee_{t\in T} TD_{i,t} \wedge TD_{i+1,t} \quad (7)$$

3.3.4 Diffusion Merging Constraints

There must be at least one track overlap in the z-direction between the P-type and N-type diffusions (using an M0-merge to connect them), or a two-pin net can achieve connection. Figs. 10(e) and 10(f) illustrate two possible options for connecting the top and bottom signal Y. Eq. (8) defines the first option to connect the two types of diffusion between the i^{th} and $(i+1)^{th}$ devices.

$$\vee_{t\in T}\left(TD_{i,t} \vee TD_{i+1,t}\right) \wedge \left(BD_{i,t} \vee BD_{i+1,t}\right) \quad (8)$$

3.3.5 Contact Allocation Constraints

Contacts can only be allocated on devices. Additionally, when the two signals in a column differ, the contact allocation for the bottom signal will be blocked by the top signal. However, when the two signals are identical, the tracks occupied by two signals are available. Eq. (9) to Eq. (11) represent the contact allocation for the diffusion column. For the bottom diffusion, Eq. (9) and Eq. (10) represent the cases where the two signals in a column are different and identical, respectively. Eq. (11) defines for the top diffusion when the two signals in a column are different.

A top/bottom diffusion column consists of its two corresponding top/bottom devices due to diffusion sharing. i and j are the indexes of the corresponding devices for c^{th} diffusion column. The top/bottom signal in the c^{th} column is denoted as $TS(c)/BS(c)$. CA represents the contact layer.

$$\mu_v^N(c,t,CA,BS(c)) \rightarrow \sim(TD_{i,t} \vee TD_{j,t}) \wedge (BD_{i,t} \vee BD_{j,t}) \quad (9)$$

$$\mu_v^N(c,t,CA,BS(c)) \rightarrow TD_{i,t} \vee TD_{j,t} \vee BD_{i,t} \vee BD_{j,t} \quad (10)$$

$$\mu_v^N(c,t,CA,TS(c)) \rightarrow TD_{i,t} \vee TD_{j,t} \quad (11)$$

3.3.6 nTSV Constraints

An nTSV cannot overlap with devices in the z-direction. Eq. (12) applies to the diffusion, and the gate is treated similarly. $nTSV$ represents the nTSV layer. i and j are the indexes of the corresponding devices for c^{th} diffusion column.

$$\mu_v^N(c,t,nTSV,n) \rightarrow \sim(TD_{i,t} \vee TD_{j,t} \vee BD_{i,t} \vee BD_{j,t}) \quad (12)$$

4. EXPERIMENTAL RESULTS

This work is implemented in C++ and executed on a Linux platform with a 2.1 GHz twenty-core CPU and 252 GB of memory. We use the Z3 SMT solver [14] for the routing tasks. Our synthesis input is the ASAP7 SPICE netlist [15], and the cell structure is based on a 4.5T layout with buried power rails.

4.1 Cell Quality Comparison

All our cell libraries strictly adhere to the SPICE netlists. The design rules follow those in [11], with MAR/EOL/VR/PRL/SHR values set to 1/2/1/1/2. For transistor folding, F:CFET represents a library that considers the CFET stacked structure, while F:[11] represents a library that folds the way [11] does. Regarding routing resources, R:FB refers to a library that takes both FS and BS resources into account, while R:F refers to a library that only considers FS resources. For the nTSV column, N:P represents a library using the prediction method proposed in this work, while N:E represents a library that enumerates all possible device track occupations.

In Table 2, [11] successfully routes all cells. For each cell type, we count how many cells have a #CPP that is not larger than [11] (#CPP≤[11]), both #CPP and #M2 tracks that are not larger than [11] (#CPP≤[11]&#M2 T≤[11]), a #CPP that is lower than [11] (#CPP<[11]), and a #CPP that is not larger but with #M2 tracks lower than [11] (#CPP≤[11]&#M2 T<[11]). Our folding method benefits cells in the types INV BUF, AND OR, NAND NOR, AO AOI, OA OAI, and XOR XNOR. Due to our folding method, there are three/two/four cells with a lower #CPP than [11] in the first library for INV BUF/AND OR/NAND NOR. More than half (eleven out of twenty) of the OA OAI cells, and all XOR XNOR cells, have a #CPP that is not larger than [11]. BS resources benefit cells in the NAND NOR, AO AOI, OA OAI, MAJ MAJI, XOR XNOR, and DFF Latch types. Due to BS resources, in the first library, seven/ten cells have lower #CPP/#M2 tracks, two cells have lower #M2 tracks, and all cells have the same #CPP and #M2 tracks in AO AOI, MAJ MAJI, and DFF Latch types, respectively.

The reason for the larger #CPP in the OA OAI type is that [11] simplifies the routing model, and half of our OA OAI cells encounter the scenario shown in Fig. 3(f). Compared to [11], our library (F:CFET+R:FB+N:P in Table 2) contains six cells with one additional #CPP and three cells with two additional #CPP in the OA OAI type. Fig. 11 illustrates our DFFHQN layout with BS resources.

TABLE 2. LIBRARY COMPARISION AMONG [11] AND OURS.

Cell type/ #Cell	#CPP<=[11]/ #CPP<=[11]&#M2T<=[11]/ #CPP<[11]/ #CPP<=[11]&#M2 T<[11]			
	F:CFET+R: FB+N:P	F:CFET+ R:F+N:P	F:[11]+R :FB+N:P	F:[11]+R :F+N:P
INV BUF/12	12/12/3/0	12/12/3/0	12/12/0/0	12/12/0/0
AND OR/21	21/21/2/0	21/21/2/0	21/21/0/0	21/21/0/0
NAND NOR/23	23/23/4/0	23/23/4/0	21/21/0/0	21/20/0/0
AO AOI/36	36/36/7/10	33/31/0/0	35/35/7/10	32/30/0/0
OA OAI/20	11/10/0/0	11/8/0/0	9/8/0/0	9/7/0/0
MAJ MAJI/3	3/3/0/2	3/3/0/0	3/3/0/2	3/3/0/0
XOR XNOR/6	6/5/0/2	6/4/0/2	5/4/0/2	5/3/0/0
HA FA/2	2/2/0/0	2/2/0/0	2/2/0/0	2/2/0/0
DFF Latch/14	14/14/0/0	14/6/0/0	14/14/0/0	14/6/0/0

TABLE 3. NTSV PREDICTION, CELL QUALITY, AND RUNTIME COMPARISON.

Cell type/ #Cell	#CPP/ #M2 tracks/ Avg. Time		
	F:CFET+ R:FB+N:P	F:CFET+ R:FB+N:E	[11]
INV BUF/12	120/0/1.7	120/0/28.1	123/0/6.5
AND OR/21	194/1/4.7	194/1/16.4	196/1/4.5
NAND NOR/23	192/0/4.2	192/0/23.9	200/0/4.7
AO AOI/36	320/0/14.5	320/0/48.5	327/10/12.3
OA OAI/20	169/5/9.1	169/5/1317.7	157/1/3.8
MAJ MAJI/3	26/0/2.6	26/0/3.0	26/2/4.9
XOR XNOR/6	58/3/4.8	58/3/266.0	58/4/4.8
HA FA/2	22/2/11.0	22/2/13469.1	22/2/27.4
DFF Latch/14	208/0/14.9	208/0/1845.3	208/0/14.3
Sum	1309/11/67.4	1309/11/17018.2	1317/20/83.4

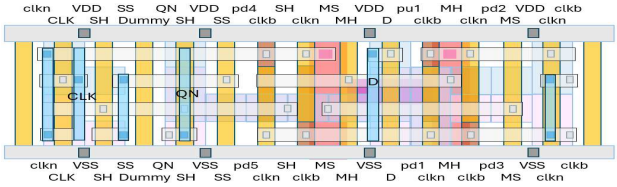


Figure 11. Our DFFHQN layout with BS resources.

4.2 NTSV Column Prediction, Cell Quality and Runtime Comparison

In Table 3, we sum the #CPP and #M2 tracks and calculate the average runtime for each cell type, with time measured in seconds. The first library predicts the nTSV columns, while the second enumerates all device track occupations to obtain the actual nTSV columns. The quality of both methods is the same, but the prediction method is 252 times faster on average than the enumeration method, reducing total runtime from 17018 seconds to 67 seconds. With total #CPP decreasing from 1317 to 1309, #M2 tracks dropping from 20 to 11, and the average runtime improving from 83.4 to 67.4 seconds, we achieve a 1%/45%/19% reduction in #CPP/#M2 tracks/runtime compared to [11]. Excluding the OA OAI type, we achieve an even greater reduction of 2% in #CPP and 68% in #M2 tracks.

5. CONCLUSIONS

To reduce the cell area and #M2 tracks, this is the first work to introduce BS resources and address stacked-structure transistor folding during the CFET cell synthesis. We fully adhere to SPICE netlists, while all previous works do not. Euler paths are utilized to estimate the CPP lower bound for folding results. In the placement stage, we calculate the FMRT with BS resources using DP. Finally, SMT decides MOS track occupation and completes cell routing. Compared to the state-of-the-art [11], we achieve superior results with reductions of 1%, 45%, and 19% in #CPP, #M2 tracks, and runtime, respectively.

6. REFERENCES

[1] G. Sisto et al. "IR-Drop Analysis of Hybrid Bonded 3D-ICs with Backside Power Delivery and μ- & n- TSVs." 2021 IEEE International Interconnect Technology Conference, pp. 1-3, 2021.

[2] R. Chen et al. "Design and Optimization of SRAM Macro and Logic Using Backside Interconnects at 2nm node." 2021 IEEE International Electron Devices Meeting (IEDM), pp. 22.4.1-22.4.4, 2021.

[3] K. Baek et al. "Simultaneous Transistor Folding and Placement in Standard Cell Layout Synthesis." 2021 IEEE/ACM International Conference On Computer Aided Design, pp. 1-8, 2021.

[4] A. Lu et al. "Simultaneous transistor pairing and placement for CMOS standard cells." 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 1647-1652, 2015.

[5] P. V. Cleeff et al. "BonnCell: Automatic Cell Layout in the 7-nm Era." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, Issue 10, pp. 2872-2885, 2020.

[6] C.-Y. Hwang et al. "A fast transistor-chaining algorithm for CMOS cell layout." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 9, Issue 7, pp. 781-786, 1990.

[7] Y.-L. Li et al. "NCTUcell: A DDA-aware cell library generator for FinFET structure with implicitly adjustable grid map." Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1-6, 2019.

[8] N. Ryzhenko et al. "Pin Access-Driven Design Rule Clean and DFM Optimized Routing of Standard Cells under Boolean Constraints." Proceedings of the 2019 International Symposium on Physical Design, pp. 41-47, 2019.

[9] B.-X. Song et al. "Routability Booster – Synthesize a Routing Friendly Standard Cell Library by Relaxing BEOL Resources." Proceedings of the 2024 International Symposium on Physical Design, pp. 185-193, 2024.

[10] C.-K. Cheng et al. "Complementary-FET (CFET) standard cell synthesis framework for design and system technology co-optimization using SMT." IEEE Transactions on Very Large-Scale Integration Systems, vol. 29, Issue 6, pp. 1178-1191, 2021.

[11] T.-W. Lee et al. "Scalable CFET Cell Library Synthesis with A DRC-Aware Lookup Table to Optimize Valid Pin Access." Proceedings of the 2025 International Symposium on Physical Design, 2025.

[12] J. Seo et al. "Pin accessibility-driven cell layout redesign and placement optimization." Proceedings of the 54th Annual Design Automation Conference, pp. 1-6, 2017.

[13] Y.-C. Hsich et al. "LiB: A CMOS cell compiler." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 10, Issue 8, pp. 994-1005, 1991.

[14] L. D. Moura et al. "Z3: An efficient SMT solver." Tools and Algorithms for the Construction and Analysis of Systems, pp. 337-340, 2008.

[15] L. T. Clark et al. "ASAP7: A 7-nm finFET predictive process design kit." Microelectronics Journal, vol. 53, pp. 105-115, 2016.

HH-PIM: Dynamic Optimization of Power and Performance with Heterogeneous-Hybrid PIM for Edge AI Devices

Sangmin Jeon†, Kangju Lee†, Kyeongwon Lee, and Woojoo Lee*
School of Intelligent Semiconductor Engineering, Chung-Ang University, Seoul, Korea
{jademin96, agl0312, since69se, space}@cau.ac.kr

***Abstract*—Processing-in-Memory (PIM) architectures offer promising solutions for efficiently handling AI applications in energy-constrained edge environments. While traditional PIM designs enhance performance and energy efficiency by reducing data movement between memory and processing units, they are limited in edge devices due to continuous power demands and the storage requirements of large neural network weights in SRAM and DRAM. Hybrid PIM architectures, incorporating non-volatile memories like MRAM and ReRAM, mitigate these limitations but struggle with a mismatch between fixed computing resources and dynamically changing inference workloads. To address these challenges, this study introduces a Heterogeneous-Hybrid PIM (*HH-PIM*) architecture, comprising high-performance MRAM-SRAM PIM modules and low-power MRAM-SRAM PIM modules. We further propose a data placement optimization algorithm that dynamically allocates data based on computational demand, maximizing energy efficiency. FPGA prototyping and power simulations with processors featuring HH-PIM and other PIM types demonstrate that the proposed HH-PIM achieves up to 60.43% average energy savings over conventional PIMs while meeting application latency requirements. These results confirm HH-PIM's suitability for adaptive, energy-efficient AI processing in edge devices.**

I. Introduction

With the advent of artificial intelligence (AI), real-world applications are rapidly expanding, fueling a trend to embed AI capabilities into IoT devices across diverse fields. However, traditional server-centric data processing, such as cloud computing, faces significant energy and latency challenges due to processing and communication overloads. Consequently, distributing AI workloads to edge devices has become a promising solution, with recent research focusing on enabling on-device AI through TinyAI models that support lightweight, local computations [1], [2].

In energy-constrained edge environments, integrating Processing-in-Memory (*PIM*) architectures has emerged as a promising approach for executing AI applications efficiently [3]–[8]. PIM enhances performance and energy efficiency in memory-intensive tasks, such as AI applications, by minimizing the overhead of data movement between processing and memory units. Early PIM designs primarily employed volatile memories like SRAM [9] and DRAM [10], but these designs faced challenges in storing large neural network weights due to the continuous power demands of volatile memory and issues such as SRAM leakage power and the periodic refresh cycles required by DRAM. To address these limitations, PIM architectures based on non-volatile memories (*NVMs*), such as MRAM [11], [12] and ReRAM [13], [14], were proposed. These designs achieve high energy efficiency in weight storage, especially when combined with power-gating techniques. However, NVMs may introduce additional read/write latency, potentially impacting overall neural network performance. Consequently, recent advances in PIM design have introduced hybrid architectures combining SRAM and NVM, known as Hybrid-PIM (*H-PIM*). These architectures use NVM to store weight data and SRAM as a buffer for input and output data, thereby enhancing both performance and energy efficiency [15]–[17].

However, H-PIM faces distinct limitations in achieving optimal energy efficiency during dynamic scenarios where inference loads fluctuate in real time on edge devices. For instance, an edge device running a YOLO model for real-time object detection experiences substantial variations in processing demand depending on the number of objects detected per video frame. Operating at a fixed performance level across all time intervals—typically set for peak computational load—inevitably leads to inefficient energy consumption. To address this, we observe that the mismatch between fixed computing resources and dynamically changing workloads has long been a challenge in traditional CPU-centric architectures. Established solutions, such as Dynamic Voltage and Frequency Scaling (*DVFS*) [18]–[21] and heterogeneous multi-processor architectures with high-performance and low-power cores [22]–[24], have been extensively researched for this purpose. Our focus centers on heterogeneous architectures, as DVFS continues to face significant challenges in edge devices due to added design complexities, such as DC-DC converters and real-time power monitoring. In contrast, heterogeneous architectures, exemplified by ARM's big.LITTLE architecture [25], are widely implemented in commercial processors, effectively improving energy efficiency by adapting to dynamic computational loads. Building on this insight, we propose configuring PIM modules, which integrate memory and Processing Elements (*PEs*) for independent computation, into high-performance and low-power configurations. This approach allows PIM architectures to dynamically balance performance and energy consumption throughout the application runtime.

To further elaborate this idea, we introduce Heterogeneous-Hybrid PIM (*HH-PIM*), an architecture designed to dynamically optimize performance and energy efficiency for AI applications on edge devices. HH-PIM integrates two distinct PIM modules: a High-Performance (*HP*) PIM module and a Low-Power (*LP*) PIM module. Each module's memory consists of a hybrid configuration of MRAM and SRAM banks, resulting in four types of memory: *HP-MRAM*, *HP-SRAM*, *LP-MRAM*, and *LP-SRAM*. Unlike conventional hybrid PIM architectures—where NVM is primarily allocated for weight storage and SRAM is reserved as an input-output buffer—HH-PIM adopts an adaptive approach. During periods of high computational demand, HH-PIM actively utilizes SRAM for weight storage as well, maximizing responsiveness to fluctuating inference loads. Additionally, to capitalize on HH-PIM's architecture, we propose an optimal data distribution algorithm that minimizes energy consumption by dynamically adjusting data allocation across the four memory types. This combinatorial optimization algorithm allocates weight data across HP-MRAM, HP-SRAM, LP-MRAM, and LP-SRAM to reduce energy use while balancing workload between HP-PIM and LP-PIM. With this design, HH-PIM efficiently adapts to the changing computational demands of AI applications, achieving significant energy savings without compromising performance.

To verify the functionality and evaluate the effectiveness of our proposed technique, we modeled the memory and PEs and performed an RTL-level design of the entire PIM processor, incorporating the HH-PIM architecture. We then conducted FPGA prototyping to confirm correct operation and measure performance of the developed PIM

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. 2022-0-00971, Logic Synthesis for NVM-based PIM Computing Architecture and No. RS-2023-00277060, Development of open edge AI SoC hardware and software platform), and in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00345668)

† Sangmin Jeon and Kangju Lee contributed equally to this work.

*Woojoo Lee is the corresponding author.

processor, while power measurements were obtained through synthesis using 45nm process technology and memory model simulations. Through various benchmark scenarios, we experimentally validated the energy-saving potential of the proposed HH-PIM architecture and data distribution algorithm. Results showed that our approach maximizes energy efficiency while satisfying the latency requirements of AI applications in edge computing environments. Specifically, the developed PIM processor demonstrated superior adaptability to real-time inference load variations compared to conventional PIM-based processors, achieving up to 60.43% average energy savings across different benchmark scenarios relative to the baseline processor. These outcomes validate the suitability of the HH-PIM architecture for maximizing efficiency in edge processors running AI applications.

II. HH-PIM Architecture for AI Edge Processors

Fig. 1 illustrates the proposed HH-PIM architecture, designed as a solution to dynamically maximize energy efficiency in response to changes in inference workload, while meeting the performance demands of AI applications on edge devices. Basically, HH-PIM adopts the general structure of near-memory computing architectures, as suggested in previous research [10], [26]. It comprises multiple PIM modules, each with PEs and memory banks, a controller that manages these modules, and an interface for external communication. Operating based on dedicated PIM instructions, commands received from the processor core are sequentially stored in the *PIM Instruction Queue*. Unlike conventional PIM architectures, where a single controller manages all PIM modules based on PIM instructions, HH-PIM incorporates two distinct controllers: *HP-PIM Controller* and *LP-PIM Controller*, as shown in the figure. This dual-controller setup is designed specifically to support HH-PIM's unique heterogeneous architecture, which consists of two types of clusters: HP-PIM module cluster operating at high performance with higher power consumption, and LP-PIM module cluster operating at low power with reduced performance. The HP-PIM and LP-PIM Controllers are responsible for controlling and synchronizing their respective module types.

Another notable feature of HH-PIM is its hybrid memory architecture, where the PIM module's memory consists of MRAM and SRAM with distinct characteristics and read/write latencies. The previous H-PIM architectures leverage the data characteristics of neural networks by storing large weight data in NVM to enhance energy efficiency, while using SRAM or DRAM for input/output data to improve computational performance by providing low-latency access. HH-PIM builds upon this approach, exploiting the performance and energy efficiency benefits of hybrid memory. However, to overcome the limitations of fixed memory allocation based solely on data characteristics, HH-PIM adds flexibility. When computational demand spikes and maximum PIM computing power is required, HH-PIM allows weight data to be stored in SRAM, thus enabling stable operation even in cases where traditional H-PIM architectures cannot meet the maximum latency requirements of certain applications. To realize this capability, the PIM module is designed to support variable operand counts retrieved from MRAM and SRAM during computation. Each PIM module's internal interface is controlled by the controller, which dynamically adjusts the load process based on data storage status. This design ensures synchronization of differing memory read cycles and access speeds between MRAM and SRAM in the LOAD state, guaranteeing reliable operation.

This heterogeneous and hybrid architecture of HH-PIM enables flexible adaptation to real-time variations in inference workloads for AI applications. When computational demand is high, the HP-PIM module and SRAM can be actively utilized to boost processor performance. Conversely, in low-demand scenarios, computations can be maximally allocated to the LP-PIM module to meet application latency requirements and maximize energy efficiency. To implement this strategy effectively, it is essential to design the PIM Controllers with precision, as they play a crucial role by synchronizing components that operate at different speeds and minimizing data movement overhead between PIM operations.

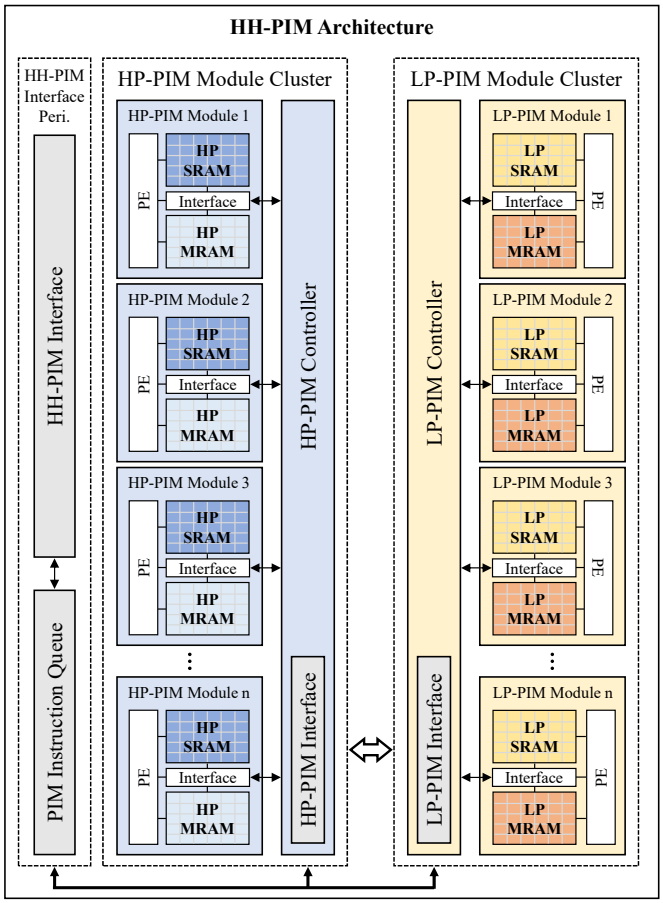


Fig. 1: Block diagram of the proposed HH-PIM architecture.

The architecture of HP-PIM and LP-PIM Controllers is fundamentally identical. Fig. 2 shows the architecture block diagram of the HP-PIM Controller, which consists of a *State Machine*, *Instruction Decoder*, *Command Encoder*, *Data Allocator*, and *Interface Logic*. The controller operates through the basic PIM instruction cycle, which includes the FETCH-DECODE-LOAD-EXECUTE-STORE phases, managed internally by the State Machine. The Instruction Decoder decodes the fetched instruction into components such as the instruction type (*Category*), specific operation or data movement details to be executed by the PIM module (*Instruction Field*), and the target module for the operation (*Module Select Signal*). The Command Encoder then generates command signals for each PIM module based on the decoded instruction details.

To minimize overhead from frequent data movement between HP-PIM and LP-PIM modules, as well as between MRAM and SRAM within each PIM module—resulting from the dynamic energy optimization of the HH-PIM architecture—the controller design of HH-PIM incorporates a Data Allocator and separates the Interface Logic as seen in Fig. 2. The Data Allocator manages data placement to minimize external data movement during PIM operations for various computations, such as convolution and Multiply-Accumulate (MAC) operations within AI applications. Additionally, the Interface Logic is divided into a CMD Interface Logic for delivering commands to the PIM modules and a MEM Interface Logic for data movement between PIM modules. When operand data is appropriately positioned in each PIM module's memory, computations can proceed efficiently through the CMD Interface Logic alone. Furthermore, the bandwidth of the MEM Interface Logic is scaled according to the number of PIM

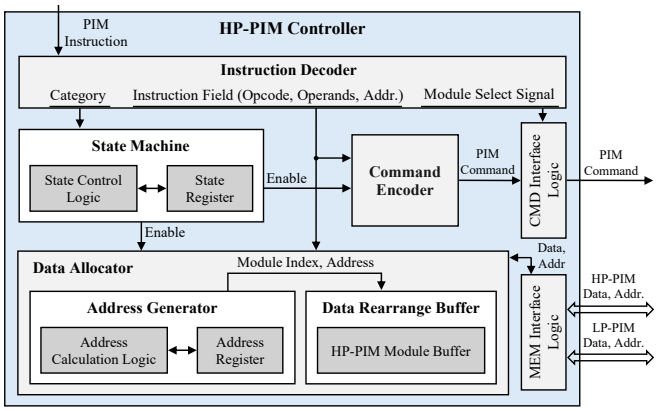


Fig. 2: Block diagram of the HP-PIM Controller architecture.

modules within each cluster, enabling parallel data transfers across clusters from multiple PIM modules simultaneously.

The controller's operation sequence for data movement between PIM modules is as follows. First, when a data placement instruction from the core prompts data transfer between HP-PIM and LP-PIM modules, the controller decodes the instruction and provides the relevant operation and address information to the Data Allocator. Based on the provided addresses, the Data Allocator accesses the memory in each PIM module via the MEM Interface Logic and stores the data to be transferred in the *Data Rearrange Buffer*. The Data Rearrange Buffer retains the data until the destination PIM module in the opposite cluster is ready for data writing, preventing data conflicts caused by the speed discrepancy between HP-PIM and LP-PIM modules. Next, the *Address Generator* within the Data Allocator generates the destination PIM module index and memory bank addresses in the opposite cluster to facilitate data movement. The generated addresses are then provided to the Data Rearrange Buffer, enabling it to place the buffered data into the memory banks of the designated PIM modules.

In summary, the PIM controllers in HH-PIM are meticulously designed to support data transfers between HP-PIM and LP-PIM modules and to effectively manage operations within each PIM module. This design maintains operational consistency and prevents data conflicts, supporting the HH-PIM architecture in achieving its design goals of real-time performance and energy efficiency optimization.

III. Dynamic Data Placement Strategy for Optimizing Energy Efficiency in HH-PIM

Optimal data placement in the HH-PIM architecture is essential for achieving maximum energy efficiency while ensuring smooth application execution without latency. In HH-PIM, each layer of a neural network is distributed across HP-PIM and LP-PIM modules for parallel computation, with the final output obtained by aggregating results from each module. When high performance is prioritized, it is critical to minimize idle times where the HP-PIM module completes its tasks quickly but must wait for the slower LP-PIM module. To achieve low output latency, a balanced distribution of data between HP-PIM and LP-PIM modules, as well as an optimal utilization ratio of MRAM and SRAM—each with different read/write latencies—must be carefully managed.

On the other hand, energy efficiency can be prioritized when AI applications do not always require the lowest possible inference latency. In such cases, HH-PIM can operate below peak performance by allocating more data to the energy-efficient LP-PIM module and increasing the usage of MRAM, thus improving overall energy efficiency while maintaining acceptable application latency. Based on this approach, we propose a dynamic data placement strategy that periodically adjusts the distribution of weight data during runtime. This strategy optimizes energy efficiency by shifting computational loads according to real-time demands, focusing on maximizing energy savings while meeting the minimum latency requirements of the application.

A. Problem Definition

In the proposed weight data placement strategy, data redistribution occurs every predefined interval T, referred to as the *time slice*. We define a set of PIM operations generated by a single inference process in an AI application as a *task*. These tasks, generated by inference, are stored in a task buffer, and HH-PIM sequentially processes the tasks stored from the previous *time slice* within the current *time slice*. This approach ensures that the operational latency of HH-PIM does not exceed $2T$, thereby maintaining the inference latency required by AI applications. In this framework, the number of tasks stored in the task buffer determines the minimum time $t_{constraint}$ needed for each task to be processed within the *time slice* without delay. Specifically, each task's processing time t_{task} must not exceed $t_{constraint}$ to guarantee that all tasks are completed within the allotted *time slice*, ensuring the system adheres to the required maximum latency.

The proposed method aims to achieve the most energy-efficient data distribution for each task within the time constraint $t_{constraint}$. The HH-PIM architecture includes four distinct storage spaces: HP-MRAM, HP-SRAM, LP-MRAM, and LP-SRAM, each with different power consumption and latency characteristics. Consequently, we are faced with a discrete combinatorial optimization problem of determining the optimal placement of each weight data in order to minimize energy consumption. This problem can be reduced to a well-known *knapsack problem* [27], with necessary adaptations for our specific context. Here, each storage space corresponds to an item i in the knapsack problem, and the time constraint $t_{constraint}$, task execution time t_{task}, and energy consumption per task E_{task} represent the knapsack's capacity, item weight, and item value, respectively. However, this problem is more complex than the standard knapsack problem, as it exhibits the characteristics of both an unbounded knapsack (due to the multiple selections of a particular storage space) and a multi-choice knapsack (due to the fixed total number of selected storage spaces, equivalent to the number of weight data).

The mathematical formulation of this problem is as follows:

$$\text{Minimize } E_{task} = \sum_{i=1}^{n} e_i \cdot x_i \tag{1}$$

$$\text{subject to: } \sum_{i=1}^{n} t_i \cdot x_i \le t_{constraint}, \quad \sum_{i=1}^{n} x_i = k,$$
$$x_i \in \mathbb{Z}_{\ge 0}, \quad \forall i = 1, 2, \cdots, n.$$

where n denotes the number of storage spaces, x_i represents the number of data assigned to storage space i, and k is the total number of weight data to be stored. Additionally, t_i and e_i denote the computation time per weight and the energy consumption per weight in storage space i, respectively. This fomula defines the objective function to minimize energy consumption per task, E_{task}, along with constraints on execution time and the total count of selected items.

B. Proposed Solution for Optimized Data Placement

Since this problem is inherently more complex than the standard knapsack problem, it is naturally NP-hard. To address the optimization problem defined in (1), we propose a bottom-up dynamic programming (DP) approach. This method uses a DP table to store intermediate results, avoiding redundant calculations and solving the problem incrementally. The recurrence relation is defined as follows:

$$dp[i][t][k] = \begin{cases} dp[i-1][t][k], & (t_i \cdot k > t) \\ \min\left(dp[i-1][t][k],\right. & \\ \left. dp[i][t-t_i][k-1] + e_i\right), & (t_i \cdot k \le t) \end{cases} \tag{2}$$

Algorithm 1 Finding Optimal Data Placement with DP.

1: **function** KNAPSACK_MIN_ENERGY
2: Set $dp[i][t][k] = \infty$ and $count[i][t][k] = 0$ for all i, t and k.
3: Set $dp[i][t][0] = 0$ for all i and t.
4: **for** i from 1 to $n/2$:
5: **for** k from 1 to K:
6: **for** t from 1 to T:
7: **if** $t_i \leq t$ **then**
8: Calculate the possible previous state time index $t-t_i$
9: $dp[i][t][k] = \min(dp[i-1][t][k], dp[i][t-t_i][k-1]+e_i)$
10: Update $count[i][t][k]$ based on the chosen minimum path.
11: **else**
12: $dp[i][t][k] = dp[i\text{-}1][t][k]$
13: $count[i][t][k] = count[i\text{-}1][t][k]$
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: **end function**

Algorithm 2 Finding Optimal (k_{hp}, k_{lp}).

1: **function** SET_ALLOCATION_STATE
2: **for** t from 0 to T:
3: Set $min_energy = \infty$, $k_opt_hp = 0$, $k_opt_lp = 0$
4: **for** k_{hp} from 1 to K :
5: $k_{lp} = K - k_{hp}$
6: **if** $dp_{hp}[n/2][t][k_{hp}] + dp_{lp}[n/2][t][k_{lp}] < \infty$ **then**
7: **if** $min_energy > dp_{hp}[n/2][t][k_{hp}] + dp_{lp}[n/2][t][k_{lp}]$ **then**
8: $min_energy = dp_{hp}[n/2][t][k_{hp}] + dp_{lp}[n/2][t][k_{lp}]$
9: $k_opt_lp = k_{lp}$
10: **end if**
11: **end if**
12: **if** min_energy remains ∞, **continue** to next t
13: $k_opt_hp = K - k_opt_lp$
14: Set $allocation_state[n][t]$ with current k_opt_hp and k_opt_lp
15: **end for**
16: **end for**
17: **end function**

where i, t, and k denote the current storage space being considered, the time constraint, and the number of weights, respectively. The value $dp[i][t][k]$ represents the minimum energy consumption required to store exactly k weight data across the first i storage spaces while satisfying the time constraint t. In this equation, when $t_i \cdot k > t$, storing k weights in the current storage space exceeds the time constraint. Therefore, in this case, the value from the previous storage space $dp[i-1][t][k]$ is carried forward. Conversely, if $t_i \cdot k \leq t$, the recurrence considers the minimum between $dp[i-1][t][k]$ and $dp[i][t-t_i][k-1]+e_i$, where the latter term represents the energy consumption if one additional weight is stored in the current storage space i, based on the previous results.

Meanwhile, in implementing the solution based on (2), it is essential to consider the architectural constraints of HH-PIM. HH-PIM allows for parallel execution of weight data distributed between HP-PIM and LP-PIM clusters; however, parallel processing is not feasible for weights stored across MRAM and SRAM within each module. To address this, we propose a partitioned approach where the DP algorithm is applied separately to HP-PIM and LP-PIM clusters, generating individual DP tables for each cluster. By combining these tables, we identify the optimal allocation of weights that minimizes energy consumption within the time constraint, yielding the best combination of k values, (k_{hp}, k_{lp}), where k_{hp} and k_{lp} represent the number of weights assigned to HP-PIM and LP-PIM clusters, respectively. This approach effectively accounts for both inter-cluster parallelism and intra-module serialization, optimizing energy efficiency while adhering to the operational constraints of HH-PIM.

First, Algorithm 1 presents the pseudo-code of the algorithm based on (2). Here, K denotes the total number of weight data to be stored in HH-PIM, and $count[i][t][k]$ is a variable used to trace the path to the optimal energy state. Lines 2 and 3 set the base conditions for the recurrence relation, while lines 7-13 correspond to the recurrence relation defined in (2). Notably, because Algorithm 1 is performed separately for both HP-PIM and LP-PIM, the iteration for i in line 4 only proceeds up to $n/2$.

Next, Algorithm 2 finds the optimal combination of k values that minimizes energy consumption using the results of the DP tables dp_{hp} and dp_{lp} generated for HP-PIM and LP-PIM, respectively. Here, *min_energy* denotes the minimum energy consumption, while *k_opt_hp* and *k_opt_lp* represent the number of weights assigned to HP-PIM and LP-PIM at this minimum. As a result, Algorithm 2 yields $allocation_state[n][t]$, representing the distribution state of weight data for each time constraint. In the proposed method, both Algorithm 1 and Algorithm 2 are performed only once during the application initialization phase to construct a Look-up Table (*LUT*) for the final output, *allocation_state*. This LUT allows rapid determination of the optimal weight placement state for varying $t_{constraint}$ values required at each *time slice* during application runtime. Furthermore, the calculation of $t_{constraint}$ at runtime incorporates the data movement overhead time needed for transitioning from the previous *time slice* data allocation to the new one, ensuring no inference delay arises due to data movement overhead.

TABLE I: Developed specifications for HH-PIM and other PIM architectures.

Architecture	PIM Module Configuration	Memory Types (per module)
Baseline-PIM	8 HP-PIM	128kB SRAM
Heterogeneous-PIM	4 HP-PIM + 4 LP-PIM	128kB SRAM
Hybrid-PIM	8 HP-PIM	64kB MRAM + 64kB SRAM
HH-PIM	4 HP-PIM + 4 LP-PIM	64kB MRAM + 64kB SRAM

The time complexities of Algorithm 1 and Algorithm 2 are $O(n \cdot T \cdot K)$ and $O(T \cdot K)$, respectively. Although these algorithms are executed only once during application initialization, the full *time slice* range for T and the large number of weight data K, often tens to hundreds of thousands in lightweight neural network models, may impose considerable overhead for energy optimization across all points in an edge processor. To mitigate this, we limit the resolution of optimization to ensure that the total computation time does not exceed 1% of each *time slice*, thereby avoiding excessive fine-grained calculations. The resulting optimal data distribution and improvements in energy efficiency are presented in the following section, along with detailed experimental settings.

IV. Experimental Work

A. Development and Evaluation of PIM Processors for Power and Performance Assessment

To evaluate the effectiveness of the proposed HH-PIM architecture and its data placement optimization strategy, we first implemented the HH-PIM architecture from Fig. 1 at the RTL level. The HP and LP clusters were each configured with four HP-PIM and four LP-PIM modules, with each PIM module equipped with 64kB of MRAM and SRAM. For a fair and precise comparison, we established Baseline-PIM, Heterogeneous-PIM (Hetero.-PIM), and Hybrid-PIM as comparison groups, each also implemented at the RTL level. Table I presents the specifications of each comparison group.

Subsequently, we developed processors equipped with the proposed HH-PIM and each of the comparative PIM architectures. RTL design and simulation were conducted using RISC-V eXpress (RVX) [28], [29]. The diagram on the right side of Fig. 3 illustrates the architecture of the processor equipped with HH-PIM, which is built with a single RISC-V Rocket [30] core. To facilitate efficient transmission and reception of large-scale AI application data, the HH-PIM communicates with the core through the AXI protocol, offering high bandwidth and

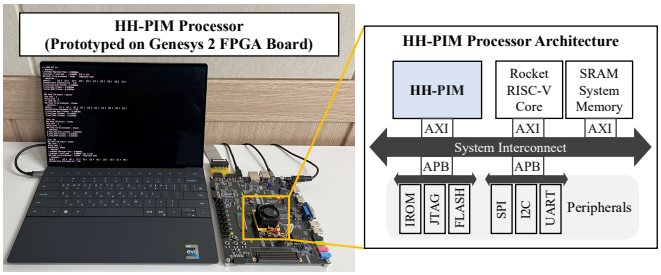


Fig. 3: Design and FPGA prototyping of the HH-PIM processor.

TABLE II: FPGA prototype resource utilization.

IPs	LUTs	FFs	BRAMs	DSPs
RISC-V Rocket Core	14,998	9,762	12	4
Peripherals	4,704	7,159	-	-
System Interconnect	5,237	7,720	-	-
HP-PIM Module	968	1,055	32	2
HP-PIM Module Controller	2,823	875	-	-
Total (HP-PIM module cluster)	6,951	5,460	128	8
LP-PIM Module	1,074	1,094	32	2
LP-PIM Module Controller	2,149	875	-	-
Total (LP-PIM module cluster)	6,680	5,616	128	8

low latency. For the system interconnect, we utilized μNoC [31], a lightweight Network-on-Chip optimized for edge devices.

Next, we performed functional verification and performance evaluation through FPGA prototyping of the developed processors. First, we modeled the operation and latency of SRAM at the RTL level, with Table III reporting the read/write latencies used in the model. These results were obtained using the NVSim simulation tool [32] at a 45nm process, with different operating voltages of 1.2V for HP-PIM and 0.8V for LP-PIM. The 0.8V operating voltage of LP-MRAM, in particular, is based on recent specifications of fabricated STT-MRAM chips [11], [33]. We then programmed the developed processors on a Genesys2 Kintex-7 FPGA board [34]. The operating clock frequency of the prototypes was set to 50 MHz, and memory latencies were scaled according to Table III. The photograph on the right side of Fig. 3 shows the prototype of the HH-PIM processor, while Table II reports the resource consumption of the prototype.

We verified the correct operation of the FPGA prototypes and measured performance metrics by running benchmark applications, expecting similar results to those observed on an SoC operating at 50 MHz [28]. For the AI benchmark applications, we used TinyML models based on CNN deep learning backbones such as EfficientNet-B0 [35], MobileNetV2 [36], and ResNet-18 [37], which are suitable for edge AI devices. We extracted the characteristics and operations of these models, calculated the proportion of PIM operations relative to the total operations, and incorporated these ratios into the benchmark applications. The *time slice* for performing the data placement algorithm in each TinyML model was set to allow up to 10 inferences per *time slice*, representing the scenario in which HH-PIM operates at maximum performance. Table IV reports the characteristics of the models incorporated into the benchmark applications. Additionally, to reflect dynamic computational load variations encountered at runtime, we configured six workload scenarios as shown in Fig. 4. These scenarios include a consistently low workload pattern (Case 1), a consistently high workload pattern (Case 2), periodic spike patterns (Case 3 and 4), a pulsing pattern with alternating high and low

TABLE III: Latency comparison of HP-PIM and LP-PIM modules.

Latency (*ns*)	STT-MRAM		SRAM		PE
	Read	Write	Read	Write	
HP-PIM (V_{dd} = 1.2*V*)	2.62	11.81	1.12	1.12	5.52
LP-PIM (V_{dd} = 0.8*V*)	2.96	14.65	1.41	1.41	10.68

TABLE IV: TinyML model specs and PIM operation ratios.

Model	# Param	# MAC	PIM Operation	–
EfficientNet-B0	95k	3.245M	85%	INT8 Quantized & Pruned
MobileNetV2	101k	2.528M	80%	
ResNet-18	256k	29.580M	75%	

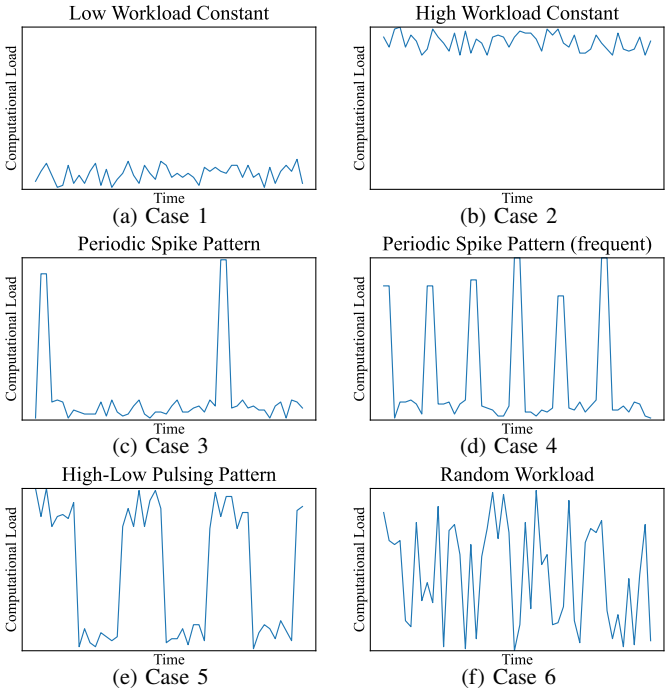


Fig. 4: Various workload scenarios of the AI benchmark app.

TABLE V: Power consumption (mW) across memory types in HP-PIM (1.2V) and LP-PIM (0.8V).

Power	STT-MRAM		SRAM		PE	
	Dynamic (Read/Write)	Static	Dynamic (Read/Write)	Static	Dynamic	Static
HP-PIM	428.48 / 133.78	2.98	508.93 / 500	23.29	0.9	0.48
LP-PIM	179.05 / 47.78	0.84	177.3 / 177.3	5.45	0.51	0.25

workloads (Case 5), and a random workload pattern (Case 6). The spike and pulse patterns simulate realistic scenarios in AI applications on edge devices, where computational demands periodically surge.

To measure the power consumption of the developed processors, we then synthesized them at a 45nm process technology. The RTL design, excluding memory, was synthesized using Synopsys Design Compiler with the 45nm Nangate PDK [38], while the memory power values were obtained through NVSim simulations at 45nm technology, as previously described. Table V details the power consumption of each memory type. These values were subsequently combined with benchmark-specific execution times, as measured from FPGA prototypes, to calculate total energy consumption. A detailed analysis of energy consumption, along with energy-saving results across various benchmark scenarios shown in Fig. 5, is discussed further in the following section.

B. Comprehensive Analysis of Energy Savings

Fig. 6 reports the results of applying the proposed data placement optimization algorithm to benchmark applications executed on HH-PIM. The x-axis represents the $t_{constraint}$ within a *time slice* (T), while the left y-axis indicates the memory utilization determined by the optimal data placement under the given $t_{constraint}$, and the right y-axis represents the corresponding energy consumption

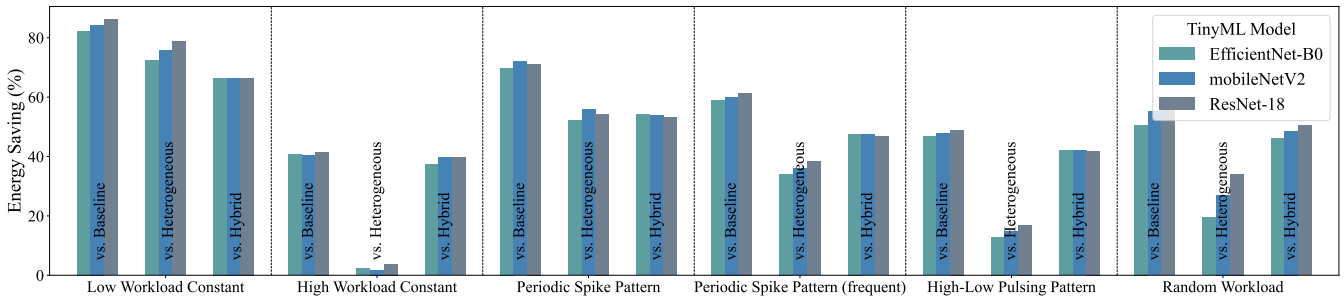


Fig. 5: Energy savings of the HH-PIM over the Baseline-, Heterogeneous-, and Hybrid-PIM across benchmark scenarios.

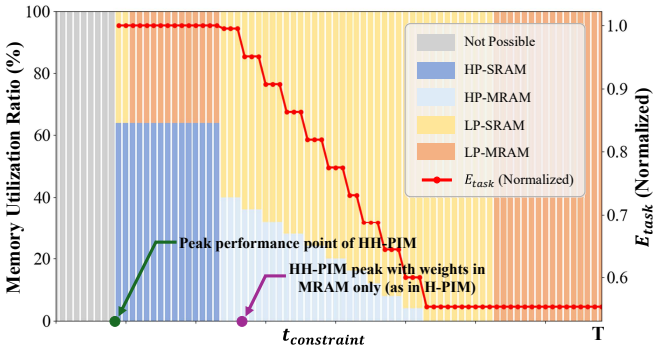


Fig. 6: Memory utilization and energy consumption across $t_{constraint}$ within *time slice* (T) based on HH-PIM's optimized data placement.

E_{task} of HH-PIM. The gray region in the graph denotes cases where $t_{constraint}$ is too small, i.e., the required performance is too high for HH-PIM to handle. The green dot marks the point where HH-PIM operates at peak performance, satisfying the application's performance requirements. At this point, HH-PIM actively utilizes SRAMs for data storage, with the total neural network data stored in a 16:9 ratio between HP-SRAM and LP-SRAM, minimizing PIM module idle times. The inference times at this point are measured as 31.06ms, 25.71ms, and 320.87ms for the EfficientNet-B0, MobileNetV2, and ResNet-18 benchmarks, respectively. As expected, energy consumption peaks at this point, and E_{task} is normalized to this value in the graph. Additionally, the purple dot indicates the peak performance point of HH-PIM when weights are stored only in MRAM (as in previous H-PIMs). At this point, the inference times for the EfficientNet-B0, MobileNetV2, and ResNet-18 benchmarks are 44.5ms, 36.84ms, and 459.74ms, respectively. This demonstrates that utilizing both SRAM and MRAM for weight storage, as proposed, outperforms the traditional method in terms of performance.

Fig. 6 demonstrates the sequential allocation of weight data across memory types as $t_{constraint}$ increases. Specifically, the distribution progresses through combinations of HP-SRAM and LP-MRAM, HP-MRAM and LP-SRAM, LP-SRAM alone, and finally LP-MRAM alone. This sequence reflects the balance between performance and energy efficiency across memory types. In the range where $t_{constraint}$ is longest, all weight data are stored exclusively in LP-MRAM, which operates at minimal power. In this configuration, other memory types are deactivated through power-gating, eliminating standby power and maximizing the processor's energy efficiency. Consequently, in this highly efficient region, HH-PIM achieves up to a 43.17% reduction in E_{task} compared to unoptimized data allocation. This significant reduction underscores the practical value of the proposed algorithm in real-world edge AI applications. Additionally, in the mid-range of $t_{constraint}$, E_{task} exhibits a quasi-linear decline with intermittent plateaus as $t_{constraint}$ increases. This pattern reflects the algorithm's capability to shift data progressively to lower-power memory while meeting inference latency constraints. These results demonstrate that the proposed solution achieves optimal energy efficiency within the constraints of application inference latency for each $t_{constraint}$, ensuring that HH-PIM can dynamically allocate data to balance energy savings and performance effectively.

TABLE VI: Energy Savings (ES) by HH-PIM for Cases 3–6.

ES (%) over	Baseline-PIM	Hetero.-PIM	H-PIM
Case 3: Periodic Spike	72.01	55.78	54.09
Case 4: Periodic Spike (frequent)	61.46	38.38	47.60
Case 5: High-Low Pulsing	48.94	16.89	42.10
Case 6: Random	59.28	34.14	50.52

Fig. 5 presents the energy savings achieved by HH-PIM compared to Baseline-PIM, Hetero.-PIM, and H-PIM (Hybrid-PIM) when executing the benchmark application across 50 *time slices* under various inference pattern scenarios from Fig. 4. The best-case scenario for HH-PIM is Case 1 (low workload constant), where the inference pattern remains low across all intervals. In this scenario, HH-PIM achieved energy savings of up to 86.23%, 78.7%, and 66.5% compared to Baseline-, Hetero.-, and H-PIM, respectively. Conversely, the worst-case scenario occurs in Case 2 (high workload constant), where the inference pattern remains consistently high. Here, HH-PIM still delivered notable energy savings of up to 41.46% and 39.69% compared to Baseline- and H-PIM, respectively. However, savings against Hetero.-PIM were limited to 3.72%, as both HH-PIM and Hetero.-PIM primarily utilize HP-SRAM and LP-SRAM in this scenario, resulting in minimal differences. Detailed energy savings for the other cases are reported in Table VI. HH-PIM achieved the highest energy savings over the baseline in ResNet-18, with up to 8.59% difference across cases. On average, HH-PIM achieved energy savings of up to 60.43%, 36.3%, and 48.58% compared to Baseline-PIM, Hetero.-PIM, and H-PIM, respectively.

V. Conclusion

In this study, we proposed the HH-PIM architecture designed to efficiently execute AI applications on edge devices. The architecture integrates HP MRAM-SRAM PIM modules and LP MRAM-SRAM PIM modules to achieve a balance between performance and energy efficiency. Additionally, we introduced a data placement optimization algorithm that dynamically allocates data based on computational demand, maximizing energy savings while maintaining application performance requirements. To validate the effectiveness of the proposed architecture and algorithm, we designed processors incorporating HH-PIM and established comparison baselines with processors equipped with Baseline-, Heterogeneous-, and Hybrid-PIM architectures. These designs were evaluated through FPGA prototyping and power simulations. By running various edge AI application benchmarks, the proposed HH-PIM demonstrated its superiority, achieving up to 60.43% average energy savings compared to conventional PIM architectures, while meeting application performance requirements.

References

[1] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "TinyML: Current progress, research challenges, and future roadmap," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1303–1306.

[2] Y. Dong, T. Jia, K. Du, Y. Jing, Q. Wang, P. Zhan, Y. Zhang, F. Yan, Y. Ma, Y. Liang *et al.*, "A model-specific end-to-end design methodology for resource-constrained TinyML hardware," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[3] Q. Zheng, S. Li, Y. Wang, Z. Li, Y. Chen, and H. H. Li, "Accelerating sparse attention with a reconfigurable non-volatile processing-in-memory architecture," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[4] X. Yang, K. Zhu, X. Tang, M. Wang, M. Zhan, N. Lu, J. P. Kulkarni, D. Z. Pan, Y. Liu, and N. Sun, "An in-memory-computing charge-domain ternary CNN classifier," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 5, pp. 1450–1461, 2023.

[5] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori *et al.*, "16.4 an 89TOPS/W and 16.3 TOPS/mm2 all-digital SRAM-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 252–254.

[6] J. Heo, J. Kim, S. Lim, W. Han, and J.-Y. Kim, "T-PIM: An energy-efficient processing-in-memory accelerator for end-to-end on-device training," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 3, pp. 600–613, 2022.

[7] K. Park, H. Jeong, S. Kim, J. Shin, M. Kim, and K. J. Lee, "A 701.7 TOPS/W Compute-in-Memory processor with time-domain computing for spiking neural network," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–11, 2024.

[8] K. Lee, S. Jeon, K. Lee, W. Lee, and M. Pedram, "Radar-PIM: Developing IoT processors utilizing Processing-in-Memory architecture for ultra-wideband radar-based respiration detection," *IEEE Internet of Things Journal*, 2024.

[9] J.-W. Su, X. Si, Y.-C. Chou, T.-W. Chang, W.-H. Huang, Y.-N. Tu, R. Liu, P.-J. Lu, T.-W. Liu, J.-H. Wang *et al.*, "15.2 A 28nm 64Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 240–242.

[10] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin *et al.*, "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," in *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 43–56.

[11] Y.-C. Chiu, W.-S. Khwa, C.-Y. Li, F.-L. Hsieh, Y.-A. Chien, G.-Y. Lin, P.-J. Chen, T.-H. Pan, D.-Q. You, F.-Y. Chen *et al.*, "A 22nm 8Mb STT-MRAM near-memory-computing macro with 8b-precision and 46.4-160.1 TOPS/W for edge-ai devices," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 496–498.

[12] H. Cai, Z. Bian, Y. Hou, Y. Zhou, Y. Guo, X. Tian, B. Liu, X. Si, Z. Wang, J. Yang *et al.*, "33.4 A 28nm 2Mb STT-MRAM computing-in-memory macro with a refined bit-cell and 22.4-41.5 TOPS/W for ai inference," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 500–502.

[13] C.-X. Xue, Y.-C. Chiu, T.-W. Liu, T.-Y. Huang, J.-S. Liu, T.-W. Chang, H.-Y. Kao, J.-H. Wang, S.-Y. Wei, C.-Y. Lee *et al.*, "A CMOS-integrated compute-in-memory macro based on resistive random-access memory for AI edge devices," *Nature Electronics*, vol. 4, no. 1, pp. 81–90, 2021.

[14] T. Yang, D. Li, Y. Han, Y. Zhao, F. Liu, X. Liang, Z. He, and L. Jiang, "PIMGCN: A ReRAM-based PIM design for graph convolutional network acceleration," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 583–588.

[15] Y. Zhu, Z. Zhu, G. Dai, F. Tu, H. Sun, K.-T. Cheng, H. Yang, and Y. Wang, "Pim-hls: An automatic hardware generation tool for heterogeneous processing-in-memory-based neural network accelerators," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[16] H. Kim, H. Ye, T. Mudge, R. Dreslinski, and N. Talati, "RecPIM: A PIM-enabled DRAM-RRAM hybrid memory system for recommendation models," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2023, pp. 1–6.

[17] F. Zhang, A. Sridharan, W. Tsai, Y. Chen, S. X. Wang, and D. Fan, "Efficient memory integration: MRAM-SRAM hybrid accelerator for sparse on-device learning," in *ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[18] W. Lee, Y. Wang, and M. Pedram, "VRCon: Dynamic reconfiguration of voltage regulators in a multicore platform," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.

[19] S. K. Panda, M. Lin, and T. Zhou, "Energy-efficient computation offloading with DVFS using deep reinforcement learning for time-critical iot applications in edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 8, pp. 6611–6621, 2022.

[20] W. Lee, Y. Wang, and M. Pedram, "Optimizing a reconfigurable power distribution network in a multicore platform," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1110–1123, 2015.

[21] H. Bouzidi, M. Odema, H. Ouarnoughi, M. A. Al Faruque, and S. Niar, "HADAS: Hardware-aware dynamic neural architecture search for edge performance scaling," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[22] K. Lee, K. Bong, C. Kim, J. Park, and H.-J. Yoo, "An energy-efficient parallel multi-core ADAS processor with robust visual attention and workload-prediction DVFS for real-time HD stereo stream," in *IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX)*, 2016, pp. 1–3.

[23] J. Park, E. Choi, K. Lee, J.-J. Lee, K. Han, and W. Lee, "Developing an ultra-low power RISC-V processor for anomaly detection," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–2.

[24] E. Choi, J. Park, K. Lee, J.-J. Lee, K. Han, and W. Lee, "Day–Night architecture: Development of an ultra-low power RISC-V processor for wearable anomaly detection," *Journal of Systems Architecture*, vol. 152, p. 103161, 2024.

[25] ARM, https://www.arm.com/technologies/big-little, Accessed 19 11. 2024.

[26] J. H. Kim, S.-H. Kang, S. Lee, H. Kim, Y. Ro, S. Lee, D. Wang, J. Choi, J. So, Y. Cho *et al.*, "Aquabolt-XL HBM2-PIM, LPDDR5-PIM with in-memory processing, and AXDIMM with acceleration buffer," *IEEE Micro*, vol. 42, no. 3, pp. 20–30, 2022.

[27] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[28] K. Han, S. Lee, K.-I. Oh, Y. Bae, H. Jang, J.-J. Lee, W. Lee, and M. Pedram, "Developing TEI-aware ultralow-power SoC platforms for IoT end nodes," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4642–4656, 2020.

[29] H. Jang, K. Han, S. Lee, J.-J. Lee, S.-Y. Lee, J.-H. Lee, and W. Lee, "Developing a multicore platform utilizing open RISC-V cores," *IEEE Access*, vol. 9, pp. 120 010–120 023, 2021.

[30] SiFIVE, https://github.com/chipsalliance/rocket-chip, Accessed 19 11. 2024.

[31] K. Han, S. Lee, J.-J. Lee, W. Lee, and M. Pedram, "TIP: A temperature effect inversion-aware ultra-low power system-on-chip platform," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[32] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[33] P.-H. Lee, C.-F. Lee, Y.-C. Shih, H.-J. Lin, Y.-A. Chang, C.-H. Lu, Y.-L. Chen, C.-P. Lo, C.-C. Chen, C.-H. Kuo *et al.*, "33.1 A 16nm 32Mb embedded STT-MRAM with a 6ns read-access time, a 1M-cycle write endurance, 20-year retention at 150° c and MTJ-OTP solutions for magnetic immunity," in *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 494–496.

[34] Genesys2, https://digilent.com/shop/genesys-2-kintex-7-fpga-development-board, Accessed 19 11. 2024.

[35] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] NCSU, https://eda.ncsu.edu/freepdk/freepdk45, Accessed 19 11. 2024.

INSIGHT: A Universal Neural Simulator Framework for Analog Circuits with Autoregressive Transformers

Souradip Poddar[1], Youngmin Oh[2], Yao Lai[1], Hanqing Zhu[1], Bosun Hwang[2], David Z. Pan[1]
[1]*ECE Department, The University of Texas at Austin, Austin, TX, USA,*
[2]*Samsung Advanced Institute of Technology, Republic of Korea*
souradippddr1@utexas.edu, youngmin0.oh@samsung.com, yao.lai@austin.utexas.edu,
hqzhu@utexas.edu, bosun.hwang@samsung.com, dpan@ece.utexas.edu

***Abstract*—The compute-intensive nature of SPICE simulations hinders effective analog design automation. This paper introduces INSIGHT, a data-efficient, adaptive, high-fidelity, technology-agnostic universal neural simulator framework that formulates analog performance prediction as an autoregressive sequence generation task to accurately predict performance across diverse circuits. INSIGHT achieves test R^2 scores $\geq$0.95, outperforming existing neural surrogates. Cross-technology transfer learning experiments show that INSIGHT can preserve model performance with $\sim$60% less training data. Low-Rank Adaptation (LoRA) integration further reduces memory footprint by $\sim$42% and training time by $\sim$25%, maintaining high performance. Our experiments show that INSIGHT-based RL sizing framework achieves 100-1000$\times$ lower simulation costs over existing sizing methods for identical benchmarks and target specifications.**

I. Introduction

Analog circuits exhibit significant variation across and within types, each characterized by distinct performance metrics, design parameters of varying dimensions, and technologies. Consequently, this has spurred numerous optimization efforts to automate the labor-intensive analog design process. Bayesian Optimization methods [1], while known for their sample efficiency, often face scalability challenges in high-dimensional optimization for complicated analog circuit optimization tasks. On the other hand, learning-based approaches, like [2–10], require numerous real-time simulator interactions (i.e., poor sample efficiency). Given the vast and complex design space, the intricate interplay of underlying circuit physics in deep submicron technologies, and the multifaceted performance trade-offs, repeated simulations within the optimization loops become inherent for effective design space exploration for optimal circuit performance.

This heavy reliance on on-the-fly, time-consuming SPICE simulations motivates the development of cheap circuit simulator surrogates to expedite the analog optimization tasks. Traditional equation-based analog performance prediction methods [11, 12] are time-consuming, unscalable, and highly inaccurate in modern processes. Black-box ML-based performance modeling approaches employing polynomial regression [13], neural networks [14], and graph representations with neural networks [15] yield rapid inference but suboptimal prediction accuracies for complex circuits while still requiring substantial, costly simulation-based training data. [16] employs semi-supervised contrastive regression with scaling rules for synthetic data augmentation for effective analog performance prediction. While aiming for data-efficiency, this approach yields suboptimal R^2 scores with complex circuits on advanced processes across a broader range of performance metrics. Developing a *data-efficient*, *high-fidelity* circuit simulator surrogate with strong generalization capabilities and rapid fine-tuning potential remains an open challenge.

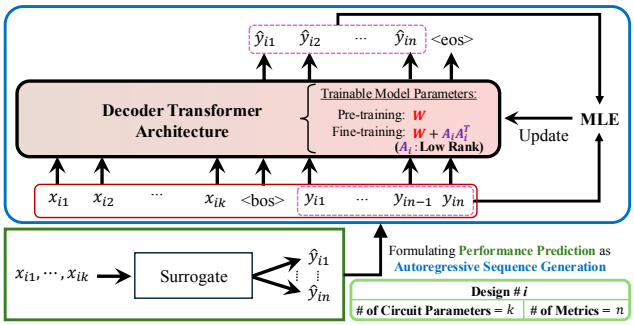


Fig. 1: INSIGHT Framework

Given the high interdependence of performance metrics inherent to analog circuits, for the first time, we propose to formulate the analog circuit performance prediction problem as *an autoregressive generation task*, i.e., predicting simulation-costly performance specification by leveraging provided design parameters and partial performance information (provided or previously-generated) in a sequential manner. The autoregressive problem setup enforces the model to automatically reason and capture the inherent interdependency in the challenging analog performance prediction task, easing the learning complexity and providing much better prediction quality than previous learning-based methods [10].

In this paper, we introduce INSIGHT, a novel, technology-agnostic, data-efficient, high-fidelity universal neural circuit simulator framework that learns to accurately predict analog performance on previously unseen parameters autoregressively, using decoder-based transformer architecture [17, 18]. Our framework's autoregressive learning facilitates accurate prediction of simulation-costly critical transient specifications leveraging performance data from less expensive simulations. INSIGHT consistently demonstrates high R^2 scores and minimal test losses across various circuits and technologies, at microsecond inference times. Furthermore, INSIGHT is highly scalable and can integrate into existing EDA optimization workflows to enhance sample efficiency. INSIGHT supports data-efficient knowledge transfer across technology nodes and

enables parameter-efficient fine-tuning on unseen nodes using Low-Rank Adaptation (LoRA) [19, 20], ensuring strong model performance and effective transfer learning.

The key contributions of this work include:

- **Novel autoregressive formulation**. We formulate analog circuit performance prediction as an autoregressive sequence modeling task, leading to our INSIGHT, a universal neural simulator framework designed around a decoder-only autoregressive Transformer architecture.
- **High data-efficiency with high prediction fidelity**: Our novel autoregressive formulation simplifies the learning complexity of the prediction task by inherently capturing the metric interdependencies and leveraging partial, less expensive metric information to boost prediction accuracy for the simulation-costly ones, thus enabling high-fidelity predictions with enhanced data efficiency.
- **A universal framework with circuit- and technology-agnostic capabilities**: The INSIGHT framework demonstrates strong generalization across unseen circuit types. Additionally, it supports efficient knowledge transfer to unseen technology nodes through parameter-efficient fine-tuning, ensuring high computational efficiency.
- We have rigorously validated INSIGHT across different circuits and technologies. Our cross-technology transfer learning experiments demonstrate INSIGHT's ability to maintain test R^2 scores $\geq$0.95 with $\sim$60% less training data than training from scratch. LoRA integration enhances computational efficiency, reducing memory footprint by $\sim$42% and training time by $\sim$25% while maintaining high model performance.
- To our knowledge, this marks the first **successful** application of a universal neural simulator framework leveraging autoregressive Transformers in analog design automation.

The source code[1] is available on GitHub.

II. Preliminaries

In this section, we briefly discuss some insights into analog circuit basics/metrics and Transformer architectures, discussing our rationale for choosing them.

A. Analog Circuit Basics and Metrics

In analog circuits, performance metrics and parameters are highly interdependent. By leveraging these interdependencies, we can build effective and data-efficient prediction models. For instance, in OPAMPs, fundamental metrics such as Quiescent Current (I_Q) and DC Gain, when combined with parameter information, can significantly enhance predictions for other key specifications, like unity-gain bandwidth (UGBW) and phase margin (PM). Similarly, integrating all static metrics provides crucial insights into the circuit's potential transient behavior. Thus, leveraging partial performance metric information can significantly enhance prediction accuracies for critical, simulation-costly specifications such as slew rate and settling time. Furthermore, given the transferable nature of circuit physics across technologies for a given topology, understanding performance interdependencies within one node provides valuable insights applicable to other nodes. These insights, combined with the adaptability and transferability of Transformer architectures, motivate the adoption of autoregressive Transformers to build a universal neural simulator framework that effectively predicts performance across technologies.

B. Transformer Architectures

Transformers constitute a powerful class of generative deep learning architectures that have revolutionized sequence modeling tasks in natural language processing and beyond [21–23]. These models stand out due to their self-attention mechanism, which effectively captures dependencies between tokens in a sequence regardless of distance. The self-attention mechanism [18] evaluates dependencies within a sequence by computing a weighted sum of value vectors. These weights highlight the most relevant token information for predicting subsequent tokens and are derived from the dot product of input vectors. Unlike traditional RNNs [24], Transformers process data in parallel, enhancing efficiency and scalability. While the original Transformer architecture [18] consists of stacked layers of encoders and decoders, decoder-only models like GPT [25] have recently demonstrated notable success in sequence data generation. Decoder-only Transformers employ solely the decoder component, composed of multi-head self-attention and position-wise feed-forward networks [26].

III. The INSIGHT Framework

In this section, we introduce our proposed universal neural simulator framework, INSIGHT, for analog performance prediction. Section III-A details INSIGHT architecture and the probabilistic rationale behind our approach. Section III-B discusses our performance metric sequencing strategy for data efficiency, while Section III-C explores INSIGHT's transfer learning capabilities.

A. INSIGHT Architecture

In INSIGHT, we are driven by a critical inquiry: Can we develop an *effective* and *efficient* surrogate model that utilizes circuit parameters and partial performance metrics information (if available) to predict remaining simulation-costly performance numbers accurately? This question is particularly pertinent as current analog front-end design automation heavily depends on time-intensive simulations like SPICE, incurring a strong demand to ease the reliance.

The problem can be formally represented as follows: For a given analog circuit topology and a B sequence of N design parameters $\{x_{i,1:N}\}_{i=1}^{B}$, where $x_{i,1:N} = (x_{i,1}, \cdots, x_{i,N})$, for example, width-to-length ratios (W/L), our goal is to predict the corresponding M performance specifications $\{y_{i,1:M}\}_{i=1}^{B}$ where $y_{i,1:M} = (y_{i,1}, \cdots, y_{i,M})$ accurately. In the context of maximum likelihood estimation (MLE), this predictive task is mathematically framed as maximizing the likelihood function $P(y_{i,1:M} \mid x_{i,1:N}; \theta)$, where θ represents the model parameters:

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} \Pi_{i=1}^{B} P(y_{i,1:M}|x_{i,1:N};\theta) \\ &= \arg\max_{\theta} \Pi_{i=1}^{B}\Pi_{j=1}^{M} P(y_{i,j}|x_{i,1:N}, y_{i,1:j-1};\theta)\end{aligned}$$

[1] https://github.com/UTDA-group/INSIGHT-FRAMEWORK

This objective is equivalent to minimizing the Negative Log Likelihood (NLL), expressed as:

$$\text{NLL} = -\sum_{i=1}^{B}\sum_{j=1}^{M} \log P(y_{i,j} \mid x_{1:N}, y_{i,1:j-1}; \theta) \quad (1)$$

Each term, $P(y_{i,j} \mid x_{1:N}, y_{i,1:j-1}; \theta)$, quantifies the likelihood of observing the token $y_{i,j}$, given the design parameters, the previously predicted performance specifications, and the model parameters. This formulation connects to the contextual approach in autoregressive sequence generation tasks involving decoder-based Transformers like GPT [25] and motivates us to use a decoder-style Transformer model for INSIGHT. Figure 1 presents a conceptual diagram illustrating INSIGHT's autoregressive sequential inferencing of performance metrics.

The formulation of circuit performance prediction as an autoregressive sequence generation task allows us to leverage the inherent strengths of transformer architectures. Unlike fully connected layers, which require fixed input and output dimensions, transformers' adaptive capabilities automatically accommodate varying lengths, which is crucial for circuits which have varying numbers of design parameters, performance metrics, and types. While recurrent neural networks (RNNs) can handle variable-length inputs, they suffer from inefficient training due to the vanishing gradient problem and limited parallel processing capabilities. Furthermore, the self-attention mechanism in transformers captures complex interdependencies among circuit parameters and performance metrics, potentially yielding more accurate predictions than standard deep neural networks or RNNs, especially for high-dimensional, heterogeneous circuit data. INSIGHT exploits these transformer strengths, facilitating the automatic construction of tailored high-fidelity performance prediction models within a unified framework that adapts to diverse parameter lengths and performance metric lengths. Note that INSIGHT's universality stems from its adaptable framework, which can consistently accommodate diverse circuit types and topologies across technologies and not a one-model-fits-all approach.

We implement INSIGHT using a custom decoder-only transformer architecture specifically tailored for analog circuit performance prediction with real-valued inputs and outputs. Unlike traditional language models, our approach skips tokenization and directly processes continuous values [21, 27]. This streamlined design results in a compact model with ∼27.5K network parameters for a topology with 30 design variables and 20 performance metrics (total sequence length of 50). Compared to large language models (LLMs) with billions of parameters, INSIGHT's modest size enables computationally and data-efficient training from scratch, facilitating rapid training and inference. Table I presents INSIGHT's architecture details.

Remark 1: While the relationship between performance metrics, relative to design parameters and among themselves, is not inherently time-series, this formulation enables self-attention and autoregressive features to leverage their interdependencies for effective and efficient predictions.

TABLE I: INSIGHT Architecture

INSIGHT Parameter	Value
Network Architecture	Decoder-Based Transformer
Dim. of model	16
# of Heads	4
# of Decoder Layers	6
dim_feedforward	64
Activation	ReLU [28]
Learning rate	0.001 with cosine annealing scheduler [29]

B. Performance Metric Sequencing in INSIGHT

In autoregressive sequence generation, accurate prediction of one metric significantly enhances the prediction accuracy of the subsequent, emphasizing the importance of metric sequence order for effective and data-efficient predictions. We employ a greedy strategy for metric sequencing in INSIGHT.

Algorithm 1 Metric Sequencing for Data-Efficient Prediction

Require: List of performance metrics $Y = [y_1, y_2, \ldots, y_M]$
Ensure: Ordered list of metrics $P_{ordered}$
  Sort Y in ascending order based on runtime and complexity
  $P_{ordered} \leftarrow []$ ▷ Initialize as an empty list
  while $|P_{ordered}| < |Y|$ **do**
    $y^* \leftarrow \arg\max_{y_i \in Y} \text{ExpectedInformationGain}(y_i | P_{ordered})$
    $P_{ordered}$.append(y^*)
    Y.remove(y^*)
  end while
  Sequence known dependent metrics after their dependencies in $P_{ordered}$
  return $P_{ordered}$

As outlined in Algorithm 1, metrics are initially sorted based on runtime complexity. Ties are resolved by iteratively selecting metrics that provide the highest expected information gain, given the previously predicted metrics, to prioritize data efficiency. If a specification is known to be directly derived from others, it is sequenced after its dependencies to optimize both data utilization and prediction accuracy. By deferring complex metrics that demand more data and exploiting performance interdependencies, this approach leverages simpler specifications to enhance the prediction accuracy of more complex ones, ensuring effective and data-efficient sequencing. Figure 2 illustrates the performance metric sequencing for the Folded Cascode Amplifier as an example, with arrows Left → Right indicating significant contributors to the autoregressive dependencies in the prediction model.

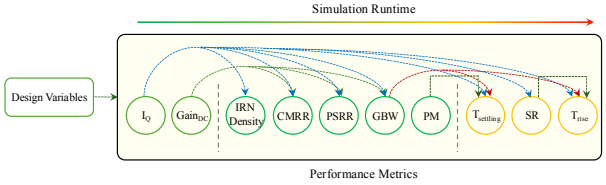


Fig. 2: Metric Sequencing in Folded Cascode Amplifier

Given a performance metric sequence order, INSIGHT utilizes self-attention to adaptively focus on the most relevant

segments of the parameter (x_1 to x_N) and generated performance output (y_1 to y_{i-1}) sequence, to accurately predict each output metric y_i. Positional encoding is used to preserve sequence order and maintain context and coherence.

Our autoregressive performance prediction formulation, integrated with the INSIGHT framework, which comprises an adaptive architecture combined with our strategy for optimizing performance sequence order, offers a data-efficient and effective analog circuit performance prediction approach.

C. Transfer Learning with INSIGHT

The fundamental circuit physics, including performance interdependencies, trade-offs, and design principles inherent to a given topology, remain largely consistent and often translate across different process technologies. INSIGHT leverages transfer learning to facilitate data-efficient knowledge transfer between processes while maintaining high prediction accuracy. This same capability further enables rapid fine-tuning to new real data within the same process and topology, allowing swift model updates—a crucial feature for effective surrogate modeling in model-based optimization frameworks.

To further enhance computational efficiency, INSIGHT incorporates Low-Rank Adaptation (LoRA) [19, 20], as detailed in Algorithm 2, to reduce the model's memory footprint and fine-tuning time by updating only a small set of rank-decomposition matrices instead of all model parameters, while maintaining high prediction performance, as shown in Fig. 3.

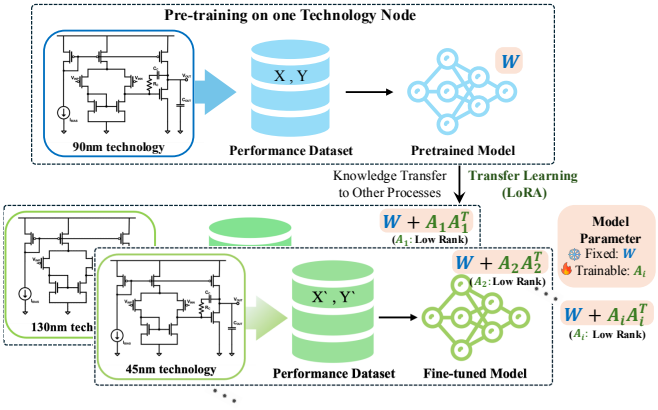


Fig. 3: LoRA-based Transfer Learning in INSIGHT

The overall wall-clock time for optimization in a model-based optimization framework can be approximated as:

$$T_{\text{optimization}} \approx t_{\text{overhead}} + K \cdot t_{\text{inference}} + N \cdot t_{\text{update}} + M \cdot t_{\text{policy-opt}} + R \cdot t_{\text{real-interaction}}, \quad (2)$$

where K, N, M, and R represent the number of surrogate model inferences, model updates, policy optimization steps, and real environment interactions during the optimization, respectively, with $t_{\text{inference}}$, t_{update}, $t_{\text{policy-opt}}$, and $t_{\text{real-interaction}}$ being their corresponding time durations, and t_{overhead} the time for data processing, one-time model pre-training, and other overhead tasks. INSIGHT aims to keep the critical components of Equation 2: N, R, t_{update}, and $t_{\text{inference}}$ small using its data-efficient high prediction accuracy, effective generalization, rapid fine-tuning capabilities, and fast ML-based inference, reducing the overall optimization time and enhancing the real-time sample efficiency.

Algorithm 2 LoRA-based Transfer Learning in INSIGHT

Require: Pre-trained $model$, $train_dataset_loader$, $device$, LoRA rank r, learning rate γ, epoch E, warm start step T_0
Ensure: Fine-tuned model with updated LoRA parameters
  function APPLY_LORA_TO_MODEL(network, LoRA_rank)
    for weight W in network **do**
      $W \leftarrow W.detach() + A_W A_W^T$ for a low rank learnable vector on A_W (LoRA parameter).
    end for
    return network
  end function
  $model$=APPLY_LORA_TO_MODEL($model$, r)
  $optimizer \leftarrow$ initialize Adam optimizer with LoRA parameters, learning rate γ
  $loss_fn \leftarrow$ MSE loss function
  $num_epochs \leftarrow E$
  $scheduler \leftarrow$ CosineAnnealingWarmRestarts with T_0
  for $epoch = 1$ to num_epochs **do**
    $total_loss \leftarrow 0$
    $model.train()$
    for all $batch \in train_dataset_loader$ **do**
      compute output and loss
      backpropagate and update LoRA parameters
      aggregate loss
    end for
    update learning rate scheduler
  end for

IV. Experiments

A. Benchmark Circuits and Technologies

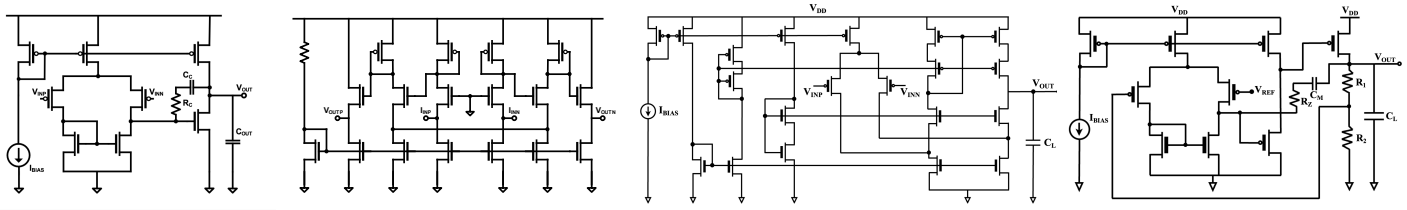
We trained and evaluated INSIGHT on a machine with eight NVIDIA P100 GPUs and an Intel Xeon E5-2698 V4 CPU. To demonstrate the effectiveness of INSIGHT, we compare its performance against two state-of-the-art neural surrogate approaches: Learn-by-Compare (LbC) [16] and the FC-Ensemble from CRONuS [10]. We use the following benchmark circuits:

I. 2-Stage Miller-Compensated OPAMP (NMOS Input Pair)
II. 2-Stage Miller-Compensated OPAMP (PMOS Input Pair)
III. 2-Stage TIA (Transimpedance Amplifier)
IV. 3-Stage TIA
V. Level Shifter

Additionally, we assess INSIGHT on the Folded Cascode Amplifier and Low-Dropout Regulator (LDO) using a commercial 45nm process with an extended set of performance metrics. Figure 4 provides detailed schematics for some of these circuits. Table II lists the performance metrics against which these circuits have been evaluated. We have validated our concept on 45nm, 90nm, and 130nm technologies.

B. INSIGHT Performance

1) Prediction Accuracy: An unseen test dataset of 1000 samples was held out for final evaluation. We created training sets of varying sizes; each split into an 80:20 train-validation



(a) (b) (c) (d)

Fig. 4: Schematics: (a) 2-Stage Miller-Compensated OPAMP (PMOS Input Pair), (b) 3-Stage TIA, (c) Folded Cascode Amplifier, (d) LDO

TABLE II: Performance Metrics for each circuit

Circuits	Performance Metrics									
Benchmark I Benchmark II Benchmark III Benchmark IV	I_Q (mA)	DC Gain (dB)	Input-Referred Noise Density (IRN Density) @ 1KHz ($V/\sqrt{Hz}$)		CMRR (dB)			UGBW (MHz)	Phase Margin (°)	
Benchmark V	DC Power (W)	Ratio	Avg. Delay (sec)					Delay Balance (sec)		
Folded Cascode Amplifier	I_Q (mA)	DC Gain (dB)	IRN Density @1KHz ($V/\sqrt{Hz}$)	CMRR (dB)	PSRR (dB)	GBW (MHz)	Phase Margin (°)	Settling Time (sec)	Slew Rate (V/sec)	Rise Time (sec)
LDO	I_Q (mA)	Phase Margin (°)		Efficiency	Dropout (mV)	Load Regulation (%)		PSRR (dB)	Line Regulation (%)	

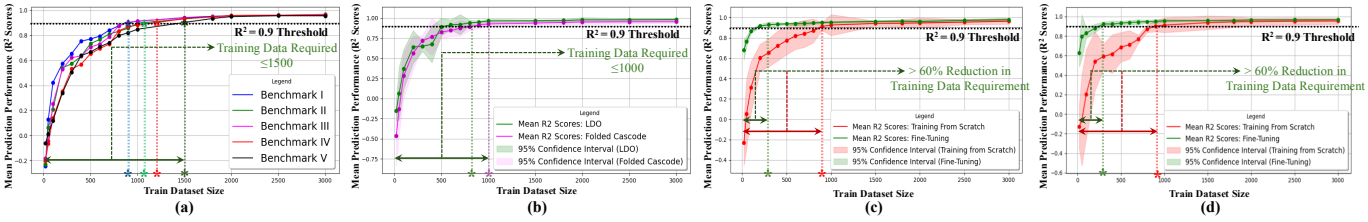


Fig. 5: INSIGHT Test Performance Plots: Demonstrating INSIGHT's high prediction accuracy and data efficiency

TABLE III: Test Performance Comparison Across Different Circuits: INSIGHT vs Baselines

	Test R^2 Scores for 2000:1000						
Neural Surrogate	**Benchmark I**	**Benchmark II**	**Benchmark III**	**Benchmark IV**	**Benchmark V**	**Folded Cascode**	**LDO**
FC Ensemble in CRONuS [10]	0.578316	0.493737	0.727980	0.627082	0.125414	0.317860	0.178161
LbC [16]	0.663117	0.540372	0.712673	0.695721	0.730163	0.521738	0.493361
INSIGHT	**0.957782**	**0.958691**	**0.957347**	**0.958614**	**0.950131**	**0.950587**	**0.976977**
	Test MSE Loss for 2000:1000						
Neural Surrogate	**Benchmark I**	**Benchmark II**	**Benchmark III**	**Benchmark IV**	**Benchmark V**	**Folded Cascode**	**LDO**
FC Ensemble in CRONuS [10]	0.02130	0.04518	0.04703	0.01640	0.04317	0.23961	0.17860
LbC [16]	0.01886	0.02964	0.03071	0.02377	0.02901	0.05928	0.06253
INSIGHT	0.00370	0.00793	0.00579	0.00427	0.00333	0.00360	0.00122

TABLE IV: Impact of Metric Sequencing on Data-Efficiency

Train Size	**Our Algorithm's Sequencing**	**Reverse Sequencing**
500	Mean R^2 = **0.824698**	Mean R^2 = 0.770055
1000	Mean R^2 = **0.926696**	Mean R^2 = 0.908772
2000	Mean R^2 = **0.950587**	Mean R^2 = 0.934486

TABLE V: Summary of Training Data Requirements

Benchmarks	**Training from Scratch**	**Transfer Learning**
Benchmark I	∼900	**∼300**
Benchmark II	∼1000	**∼200**
Benchmark III	∼900	**∼300**
Benchmark IV	∼1200	**∼300**
Benchmark V	∼1500	**∼400**

ratio. INSIGHT was trained on the training subset, validated for hyperparameter tuning and early stopping on the validation subset, and then evaluated on the fixed unseen test set. This process was repeated for 30 independent trials per training set size for statistical robustness, and the mean test performances were recorded. Figure 5(a) showcases INSIGHT's mean test performance across benchmarks I–V when trained

TABLE VI: Impact of LoRA on INSIGHT

Benchmarks	**# Trainable Parameters**		**Wall-clock Time Ratio** ($\frac{t_{\text{LoRA}}}{t_{\text{Full Fine-tuning}}} \times 100\%$)
	Full Fine-tuning (tune-all approach)	**LoRA**	
Benchmark I	27.249K	**15.729K**	**∼72.21%**
Benchmark II	27.265K	**15.745K**	**∼71.13%**
Benchmark III	27.217K	**15.697K**	**∼74.22%**
Benchmark IV	27.313K	**15.793K**	**∼73.41%**
Benchmark V	27.233K	**15.713K**	**∼74.49%**

TABLE VII: Performance Comparison of INSIGHT-based RL framework (with PPO) vs CRONuS vs AutoCkt (PPO)

	INSIGHT-based framework (with PPO)	CRONuS [10]	AutoCkt (PPO) [6]
Benchmarks	Real-time Simulations	Real-time Simulations	Real-time Simulations
I	**2 ± 0.18**	450	14,100
II	**2 ± 0.18**	330	9,300
III	**15 ± 5**	990	11,100
IV	**15 ± 5**	390	8,700
V	**15 ± 5**	420	10,400

from scratch for varying training set sizes as described in the above setup for 90nm technology, while Figure 5(b) shows INSIGHT's performance on Folded Cascode and LDO on a commercial 45nm process. INSIGHT achieves high prediction performance ($R^2 > 0.9$) with $\leq$1500 training samples across benchmarks, highlighting its data efficiency. Table III summarizes INSIGHT's mean test performance using 2000 samples for training and validation following the setup described earlier. As shown, INSIGHT consistently achieves high test R^2 scores and low MSE losses across all benchmark circuits, outperforming our baselines [16] and [10]. Table IV highlights the effectiveness of INSIGHT's metric sequencing strategy for the Folded Cascode, showcasing improved data efficiency in limited data scenarios over the reversed order.

2) Knowledge Transfer across Technologies: For each topology, similar to the prediction accuracy evaluation setup discussed in Section IV-B1, we reserved an unseen test set of 1000 samples. INSIGHT's test performance was then evaluated on each for different training set sizes, comparing training from scratch against fine-tuning over 30 independent trials per training set size. Mean R^2 scores and standard deviations were noted. Figure 5(c) illustrates prediction performance on Benchmark II from 90nm to 130nm (to larger node), while Figure 5(d) shows results for Benchmark III from 90nm to 45nm (to smaller node). Both figures display mean R^2 scores with the corresponding 95% confidence intervals, demonstrating INSIGHT's effectiveness across technologies. Table V summarizes the training data requirements for benchmark circuits I–V transferred from 90nm to 45nm, using a mean test R^2 score of 0.9 as the minimum acceptable performance threshold. Our experiments showed a significant $\sim$60% reduction in required training data for transfers between 90nm and 45nm, and 130nm and 90nm, while preserving high prediction fidelity. Implementing LoRA allowed us to maintain high model performance while significantly reducing computational requirements. As shown in Table VI, LoRA reduced memory footprint by $\sim$42% and training time by $\sim$25% for training size of 500 while maintaining model performance above the R^2 threshold of 0.9.

3) Inference Speed: When simultaneously processing a batch of 1,000 combinations of input parameters, INSIGHT's inference time across our benchmark circuits was measured to be $115.21\mu s$ (mean) $\pm 12.01\mu s$ (std dev), compared to several minutes for the case of commercial simulators. This result was derived from 100 independent trials per circuit on our setup.

C. Analog Sizing using INSIGHT

For these experiments, INSIGHT was pre-trained from scratch on a dataset of 1,000 randomly generated samples per topology, comprising Gaussian-sampled design parameters and their corresponding SPICE-simulated performance metrics. Note that INSIGHT does not make any a priori assumption on the underlying data distribution, and using a moderate to expert dataset would likely enhance the performance of the sizing framework. PPO-based RL agents [30] with default hyperparameters were employed, a method extensively used in EDA because of its robustness and effectiveness [6, 31], and the performance compared against two established analog circuit sizing methodologies: AutoCkt [6], a PPO-based online learning-based algorithm with default hyperparameter settings, and CRONuS [10], a state-of-the-art neural surrogate-based approach. Table VII presents the performance comparison based on the number of real-time simulations (steps) required for different circuits on 45nm technology, each row representing a test case corresponding to a fixed target specification randomly selected from the same set of designer-specified performance specifications used in [10]. The INSIGHT-based framework's performance is reported as mean $\pm$ standard deviation over 30 independent trials for robust statistical analysis. Our sizing experiments show that INSIGHT effectively reduces the critical components of Equation 2 (Section III-C) for this application, achieving $\sim$ 100-1000$\times$ improvement in computation costs over existing sizing methods on identical benchmarks and test cases, demonstrating INSIGHT's efficacy.

V. Conclusion

This paper introduces INSIGHT, a novel, technology-agnostic, and data-efficient universal neural simulator framework for analog performance prediction. By formulating the prediction task as autoregressive sequence generation, INSIGHT leverages the strengths of decoder Transformer architectures and their self-attention mechanisms to capture inherent parameter-performance interdependencies. This approach allows accurate prediction of simulation-costly critical specifications leveraging less expensive (predicted or previously available) performance metrics as conditioning information, achieving high-fidelity and data-efficient predictions across diverse circuit types and technologies. The Transformer-based architecture enables adaptive handling of variable-length design parameters and performance metrics, making INSIGHT a universal framework. INSIGHT supports compute- and data-efficient knowledge transfer across processes through parameter-efficient fine-tuning using Low-Rank Adaptation (LoRA) while maintaining high model performance. Furthermore, these fast fine-tuning capabilities make INSIGHT suitable for model-based optimization frameworks where model update time is critical. Our experiments demonstrate INSIGHT's effectiveness, consistently achieving test R^2 scores ≥ 0.95 across circuits and technologies.

To our best knowledge, this work represents the first successful application of a universal neural simulator framework using autoregressive Transformers for analog design automation. Future work will focus on developing more intelligent strategies to enhance data efficiency and extend this formulation to address more complex analog design challenges.

Acknowledgement

This work was supported in part by Samsung Advanced Institute of Technology (SAIT), NSF under grants CCF-1704758 and CCF-2112665, SRC under task 3160.007, and an equipment donation from Nvidia. We sincerely thank Prof. Joydeep Ghosh from the University of Texas at Austin for useful discussions and valuable feedback.

REFERENCES

[1] Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. Recent Advances in Bayesian Optimization. *ACM Computing Surveys*, 55(13s):1–36, 2023.

[2] Minjeong Choi, Youngchang Choi, Kyongsu Lee, and Seokhyeong Kang. Reinforcement Learning-based Analog Circuit Optimizer using gm/ID for Sizing. In *Proc. DAC*, pages 1–6. IEEE, 2023.

[3] Jinxin Zhang, Jiarui Bao, Zhangcheng Huang, Xuan Zeng, and Ye Lu. Automated Design of Complex Analog Circuits with Multiagent based Reinforcement Learning. In *Proc. DAC*, pages 1–6. IEEE, 2023.

[4] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind Sharma, Sachin Sapatnekar, Ramesh Harjani, and Jiang Hu. A Circuit Attention Network-Based Actor-Critic Learning Approach to Robust Analog Transistor Sizing. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 2021.

[5] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *Proc. DAC*, pages 1–6. IEEE, 2020.

[6] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs. In *Proc. DATE*, pages 490–495. IEEE, 2020.

[7] Ahmet F. Budak, Prateek Bhansali, Bo Liu, Nan Sun, David Z. Pan, and V. Kashyap Chandramouli. DNN-Opt: An RL inspired optimization for analog circuit sizing using deep neural networks. In *Proc. DAC*, pages 1219–1224, 2021.

[8] Ahmet F Budak, David Smart, Brian Swahn, and David Z Pan. APOSTLE: Asynchronously parallel optimization for sizing analog transistors using DNN learning. In *Proc. ASPDAC*, pages 70–75, 2023.

[9] Souradip Poddar, Ahmet Budak, Linran Zhao, Chen-Hao Hsu, Supriyo Maji, Keren Zhu, Yaoyao Jia, and David Z Pan. A data-driven analog circuit synthesizer with automatic topology selection and sizing. In *Proc. DATE*, pages 1–6. IEEE, 2024.

[10] Youngmin Oh, Doyun Kim, Yoon Hyeok Lee, and Bosun Hwang. CRONuS: Circuit Rapid Optimization with Neural Simulator. In *Proc. DATE*, pages 1–6, 2024.

[11] Behzad Razavi. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill, Inc., USA, 1 edition, 2000.

[12] Tony Chan Carusone, David Johns, and Kenneth William Martin. *Analog integrated circuit design*. John Wiley and Sons, 2nd ed edition, 2012.

[13] Walter Daems, Georges Gielen, and Willy Sansen. Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *IEEE TCAD*, 22(5):517–534, 2003.

[14] Alireza Pourkhalili and Sayed Mohammad Ali Zanjani. Design of a Two-Stage Operational Amplifier Using Artificial Neural Network. *Journal of Intelligent Procedures in Electrical Technology*, 2024.

[15] Zhenxin Zhao and Lihong Zhang. Efficient performance modeling for automated CMOS analog circuit synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(11):1824–1837, 2021.

[16] Zihu Wang, Karthik Somayaji NS, and Peng Li. Learn-by-Compare: Analog Performance Prediction using Contrastive Regression with Design Knowledge. In *Proc. DAC*, 2024.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Proc. NeurIPS*, 30, 2017.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Proc. NeurIPS*, 30, 2017.

[19] Uijeong Jang, Jason D Lee, and Ernest K Ryu. LoRA Training in the NTK Regime has No Spurious Local Minima. *arXiv preprint arXiv:2402.11867*, 2024.

[20] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*, 2021.

[21] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[23] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

[24] David E Rumelhart, James L McClelland, PDP Research Group, et al. *Parallel distributed processing, volume 1: Explorations in the microstructure of cognition: Foundations*. The MIT press, 1986.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving Language Understanding by Generative Pre-Training. 2018.

[26] Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[27] Romain Ilbert, Ambroise Odonnat, Vasilii Feofanov, Aladin Virmaux, Giuseppe Paolo, Themis Palpanas, and Ievgen Redko. Samformer: Unlocking the potential of transformers in time series forecasting with sharpness-aware minimization and channel-wise attention. In *Forty-first International Conference on Machine Learning*.

[28] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. ICML*, pages 807–814, 2010.

[29] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] Weidong Cao, Mouhacine Benosman, Xuan Zhang, and Rui Ma. Domain knowledge-infused deep learning for automated analog/radio-frequency circuit parameter optimization. In *Proc. DAC*, pages 1015–1020, 2022.

PUFiM: A Robust and Efficient FeFET-Based Security Solution Merging Physical Unclonable Function with Compute-in-Memory for Edge AI

Taixin Li[1], Thomas Kämpfe[2,3], Jianfeng Wang[1], Kai Ni[4], Vijaykrishnan Narayanan[5], Huazhong Yang[1], Xueqing Li[1]
[1]EE, BNRist/LFET, Tsinghua University, China; [2]Fraunhofer IPMS, Germany; [3]TU Braunschweig, Germany;
[4]University of Notre Dame, USA; [5]The Pennsylvania State University, USA; Email: xueqingli@tsinghua.edu.cn

***Abstract*—Compute-in-memory (CiM) has become a promising candidate for edge AI by reducing data movements through *in-situ* operations. However, this emerging computational paradigm also poses the vulnerability of model leakage as the weights are stored in plaintext for computing. While prior works have explored lightweight encryption methods, CiM is usually considered a separate module instead of a system component, leaving the origin of keys unclear and unprotected. Physical unclonable functions (PUFs) offer a potential origin of keys, but a comprehensive framework for securing key generation and delivery remains lacking. Besides, the complementary ciphertext storage incurs substantial costs and degrades the performance.**

This work proposes PUFiM, a robust and efficient security solution for edge computing based on ferroelectric FETs (FeFETs). For the first time, a strong PUF is synergized with CiM to enable authentication, key generation, and encrypted computations within a unified array for comprehensive protection. To achieve this synergization, a high-density hybrid storage and computation approach combining PUF and weight bits via multi-level cell (MLC) FeFETs is proposed. Besides, two PUF enhancement techniques and a novel mapping scheme are developed to improve security and efficiency further. Results show that PUFiM withstands PUF modeling attacks with up to 10M samples. Moreover, PUFiM reduces the inference accuracy by >60% under 95% key leakage and achieves >9.7× compute density and >1.2× energy efficiency improvement compared with the state-of-the-art SRAM/NVM secure CiMs.

***Keywords*—Hardware Security, Computing-in-Memory, Physical Unclonable Functions, Ferroelectric FET (FeFET)**

I. Introduction

Deep neural networks (DNNs) have shown superior performance in diverse application domains, from visual recognition to natural language processing [1]–[3]. With the increasing demand for privacy and real-time responses, DNN models are migrating from cloud to edge by deploying the trained models on edge devices for inference. However, in conventional von Neumann architectures, efficient DNN inference faces the bottleneck of substantial data movements between processing units and memories [4]. This memory wall challenge prompts the emerging computational paradigm of compute-in-memory (CiM), which aims to reduce data movements via *in-situ* operations [5], [6]. Towards the paradigm, various impressive implementations have been demonstrated, including CMOS CiMs [7], [8] and nonvolatile memory (NVM) CiMs [9]–[12].

However, in the unsupervised environment, CiMs suffer from the vulnerability of model leakage as the weights are stored in plaintext for computing. This data theft risk is exacerbated when NVMs are employed, as shown in Fig. 1(a). Therefore, weight encryption becomes crucial to protect the DNN models in NVM CiMs. Conventional approaches include software runtime encryption [13], [14] and hardware data encryption such as AES [15], [16]. Despite the wide usage of these methods, software encryption still holds an attack window during algorithm execution, while AES-based encryption is incompatible with in-memory computing. Besides, both approaches incur significant performance and energy overheads, making them less feasible for edge devices.

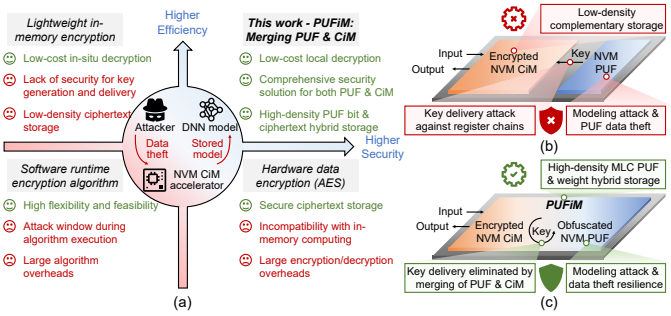


Fig. 1. Secure CiMs: (a) Comparison between encryption methods. (b) Challenges in existing designs. (c) Insights of the proposed PUFiM.

Beyond the conventional approaches, recent works have explored the use of *in-situ* XOR cipher for lightweight encryption [17]–[24]. Through the subtle combination of inputs and keys, the weight decryption can be performed simultaneously with computing, achieving ultra-low cost and secure storage. However, the security of this method is premised on the secure generation and delivery of the keys, which is underexplored in prior designs. While physical unclonable functions (PUFs), which generate unique and unpredictable responses to given challenges, can be employed for key generation [20]–[24], the attacks against PUFs [25]–[27] and key delivery via register chains [28] still pose threats to the system, as shown in Fig. 1(b). Furthermore, the commonly used complementary ciphertext storage degrades both density and efficiency.

To cope with this challenge, we highlight the necessity of viewing PUF and CiM as inseparable parts of the system and propose **PUFiM**, a robust and efficient ferroelectric FET (FeFET)-based security solution merging PUF with CiM. As shown in Fig. 1(c), the core insight is to synergize PUF with CiM in a unified array to enable local key generation, decryption and computation with comprehensive security,

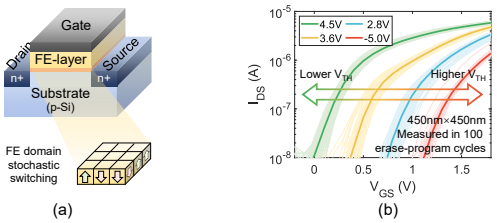


Fig. 2. FeFET: (a) Structure. (b) I_{DS}-V_{GS} curves with C2C variations [29].

where multi-level cell (MLC) FeFETs are exploited to support efficient PUF and weight hybrid storage and computation.

The highlighted contributions are summarized as follows:

- To the best of our knowledge, PUFiM is the **first** edge security solution that fuses PUF and CiM into one array for comprehensive and area-efficient protection.
- The startup value-based obfuscation and nonlinearity-enhancing post-processing are proposed to secure PUF.
- A significance-aware weight mapping scheme is proposed to improve CiM energy efficiency and security further.
- Extensive experiments are performed, and optimal configurations are explored. Results show that PUFiM reduces the inference accuracy by >60% under 95% key leakage, and achieves >9.7× compute density and >1.2× energy efficiency improvement compared with the state-of-the-art SRAM/NVM secure CiMs [18]–[21].

II. Background and Related Works

This section introduces the FeFET basics, illustrates the attack model and analyzes the bottlenecks of related works.

A. FeFET Basics

FeFET is a MOSFET with the insertion of a ferroelectric (FE) layer in the gate stack, as shown in Fig. 2(a) [29]–[31]. Reports have shown the fabrication of FeFET arrays at 28nm bulk [29] and 22nm FDSOI [31] technology node. Recent research has also reported the large-scale 3D integration of FeFETs [32] and the FeFETs with unlimited endurance [33].

The FE layer consists of multiple independent FE domains whose polarization states collectively affect the FeFET threshold voltage (V_{TH}). The multi-domain structure of the FE layer leads to two intriguing features [30] shown in Fig. 2(b): (i) By applying different write voltages (V_W) to the FeFET gate, the polarization state of the FE layer can be partially switched to achieve MLC storage; (ii) The polarization switching of a single FE domain is an abrupt and stochastic process due to the random polarization nucleation event, enabling FeFETs with significant cycle-to-cycle (C2C) variations.

Inspired by the two features, we leverage the FeFET MLC capability to achieve high-density hybrid storage and computation of the PUF and weight bits. Meanwhile, the FeFET C2C variation is exploited as the PUF entropy source by comparing the V_{TH} obtained from two write cycles of the same device.

B. Attack Model

The attacker is assumed to have access to an edge AI chip with an integrated secure CiM, which can be obtained from the market and used for inference as an authorized user. The communication channel between the manufacturer or the model vendor and the chip can be monitored. Additionally, the attacker can carry out physical attacks to probe the weights of the DNN model stored in the NVM cells and the keys stored in the register chain, as demonstrated in [28], [34]. The attacker also has complete knowledge of the hardware implementation. However, due to the advancements in technology nodes, directly probing the signals within the secure CiM array is practically infeasible [35]. Therefore, the internal signals, including those related to the computation and key operations, are assumed to be secure from such attacks.

C. Related Works

To thwart model leakage with tolerable overheads, several lightweight secure CiM designs have been explored. A common approach involves pre-storing the encrypted weights in complementary cells with dual word lines (WLs), which allows the correct WL to be selected based on the key for *in-situ* decryption. Representative works include XOR-CIM [20], Secure-RRAM [17], SRA [18] and NAND FeFET CiM [19]. However, these designs double the memory cells and incur substantial area overheads. [36] eliminates the additional cells using a novel device, though its reliability and scalability remain challenges. Moreover, the above designs consider CiM a separate module instead of a system component, neglecting key generation and delivery as the basis for CiM security.

For secure key generation, PUF is employed as security primitives, exploiting entropy sources such as manufacturing variations to produce unique challenge-response pairs (CRPs). However, the weak PUFs utilized in [20]–[22] provide insufficient CRP space for authentication, thus vulnerable to attacks on communication channels [26]. In contrast, [23], [24] implement strong PUFs based on in-memory computing but face challenges from modeling attacks [25] and NVM data theft [34]. Furthermore, the separate implementations of PUF and CiM remain prone to key delivery attacks [28] and limit the opportunity for system performance co-optimization. Therefore, PUFiM is motivated to synergize PUF with CiM in one array for comprehensive security and improved efficiency.

III. PUFiM Overview

Fig. 3(a) illustrates the complete workflow of PUFiM from PUF enrollment to edge inference. The novel merging of PUF and CiM, combined with other highlighted techniques, makes PUFiM behave as a black box for the attacker and ensures robust and efficient protection throughout the workflow:

(i) PUF enrollment with obfuscation. After the PUFiM chip is manufactured in a trusted environment, the PUF bits are generated by exploiting the FeFET C2C variations and written back into the array. Based on the random startup values from the peripherals, a subset of the PUF bits is selected and activated for CRP generation, providing obfuscation for the nonvolatile PUF bits. These startup values and CRPs are subsequently stored in the cloud as the device fingerprint.

(ii) Secure PUF authentication. Before communication, PUFiM sends challenges to the cloud and verifies between the

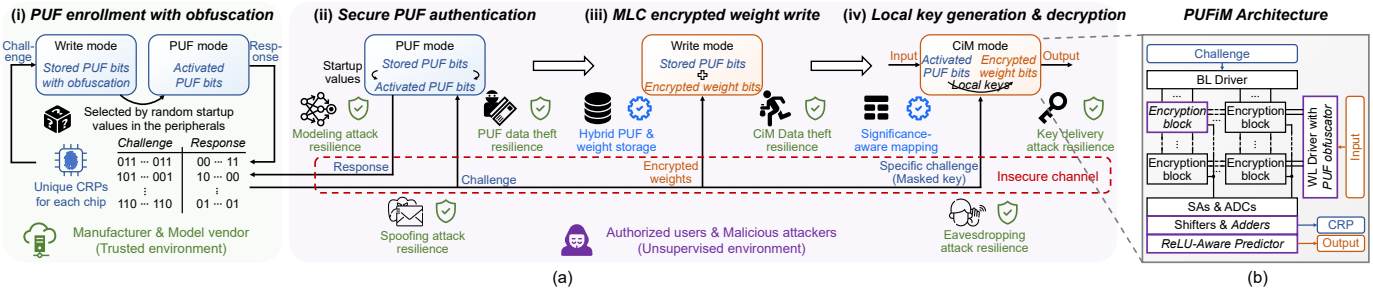


Fig. 3. Overview of PUFiM: (a) Complete workflow with security and efficiency enhancement approaches. (b) Overall architecture with blockwise structure.

received and self-generated responses for authentication. This prevents the *spoofing attack* against device identity. Besides, the proposed obfuscation and post-processing techniques provide resilience to the *PUF data theft* and *modeling attack*.

(iii) MLC encrypted weight write. Once authenticated, the model vendor encrypts the weights using the stored CRPs and offloads the encrypted weights to the edge to thwart *CiM data theft*. These weights are written to the MLC FeFETs based on the significance-aware mapping scheme and co-exist with the PUF bits, enabling high-density and secure hybrid storage.

(iv) Local key generation, decryption and computation. For edge inference, PUFiM acquires a specific challenge from the cloud and computes it with the activated PUF bits to generate local keys for each encryption block. These internal keys, distinct from the publicly accessible CRPs, are secure and can be retained until power-down. The local keys decrypt the weights in the block for computation. By merging PUF with CiM, the challenge (as a masked key) and the local keys protect PUFiM from *eavesdropping* and *key delivery attacks*.

Fig. 3(b) shows the PUFiM architecture. Challenges and DNN inputs are applied vertically and horizontally, respectively. During inference, multi-bit inputs are sequentially processed cycle by cycle, with the outputs accumulated using shifters & adders. Further summation of the outputs produces the final feature map. The encryption block with hybrid storage and computation is the core of PUFiM to enable the synergization of PUF and CiM. The startup value-based obfuscation and nonlinearity-enhancing post-processing are facilitated by the WL driver and the adders, while the significance-aware weight mapping improves efficiency via the ReLU-aware predictor.

IV. Design Details

This section details the four highlighted techniques of PUFiM, which synergizes PUF with CiM and significantly improves security, area efficiency and energy efficiency.

A. MLC Hybrid Storage and Computation

A key challenge in merging PUF and CiM lies in hybridly storing PUF and weight bits without affecting their respective computations. This becomes even more challenging with MLC, where the two bits stored in one device can cause interference, and the FeFET variations degrade the computing reliability. Therefore, PUFiM introduces a novel scheme to achieve both **high-density storage** and **reliable computation**.

Fig. 4(a) shows the proposed encoding and storage scheme. The stored bits are encoded into FeFET states with Gray code

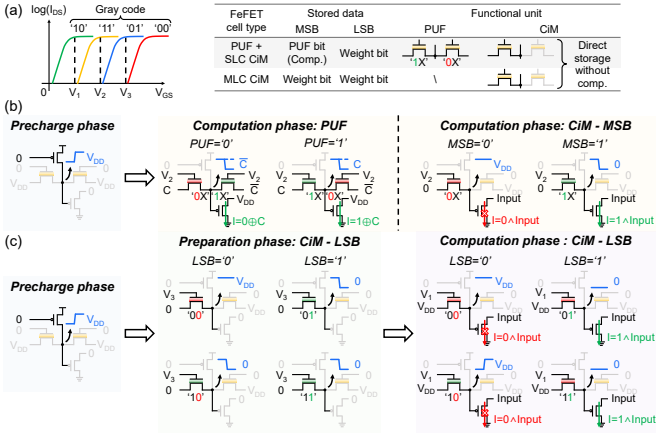


Fig. 4. MLC hybrid storage and computation: (a) Encoding and storage scheme. (b) MSB computation (PUF and CiM). (c) LSB computation (CiM).

and read out with $V_{1/2/3}$. For hybrid storage, FeFET cells are categorized into two types: (i) Some cells store weight bits in the LSB and complementary PUF bits in the MSB, with two such cells forming a PUF functional unit. (ii) Other cells store weight bits in both the MSB and the LSB. Note that all weight bits are stored directly and can be operated in one cell.

Fig. 4(b) shows the computation scheme for the MSB, which begins with a precharge phase. For PUF computation, V_2 is applied to the gates of two FeFETs with complementary PUF bits, and the challenge is input complementarily from the drains. This setup makes the PMOS on/off state consistent with the XOR result of the PUF and challenge bits. For CiM computation, the FeFETs storing '1' in the MSB discharge the internal node at V_2, activating the PMOS if the input is VDD.

Fig. 4(c) shows the computation scheme for the LSB. The preparation phase sets the FeFET gate voltage to V_3, discharging the internal nodes of all cells to GND except those storing '00'. Then V_1 is applied, and the FeFET drain is raised to VDD, which charges the '10' cells to switch off the PMOS. Consequently, only cells storing '1' in the LSB output currents.

The above method enables the hybrid storage of PUF and CiM while supporting their respective computations (XOR and AND). The FeFET variations are mitigated by converting the readout results into PMOS currents. The related evaluation of the PUFiM reliability is presented in Section V-B.

B. Local Key Generation, Decryption and Computation

Based on the proposed MLC hybrid storage and computation method, the encryption block circuit and operation mechanism are illustrated in Fig. 5 and detailed below:

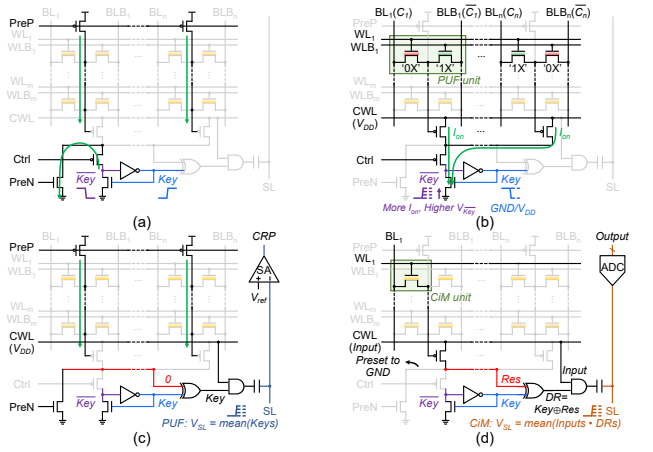


Fig. 5. Encryption block circuit and operation mechanism: (a) Precharge. (b) Local key generation. (c) PUF mode. (d) CiM mode.

(i) Precharge. The internal nodes in the FeFET array and the inverter output are precharged to VDD.

(ii) Local key generation. The challenge vector is input via BLs and BLBs to perform XOR computation with one row of PUF units. The resulting output currents are accumulated, affecting the voltage division at the inverter input. Once the voltage exceeds the inversion threshold, the inverter output (i.e., the key) switches to GND and shuts down the NMOS loop to lock the result. Otherwise, the output remains at VDD.

(iii) PUF-mode CRP generation. In PUF mode, the generated key is delivered to the capacitor connected to the SL. The capacitive coupling across encryption blocks averages the keys, and the obtained SL voltage is compared with a reference voltage by the sense amplifier (SA) to obtain the CRP. Note that this CRP requires post-processing for the final output.

(iv) CiM-mode decryption and computation. In CiM mode, one cell per block is activated to convert the weight bit into the voltage at Res, which is XORed with the generated local key for decryption. The decrypted result is subsequently ANDed with the input from the CWL for charge-domain computing.

The proposed encryption block generates and utilizes keys locally, effectively **preventing key delivery attacks**. The key generation requires only **one row of activated PUF bits** in each block without introducing heavy costs. The charge-domain computing offers energy-efficient and reliable computation, while the CMOS circuits can be shared by the FeFET array to reduce the area overheads. The block size configuration is analyzed and optimized in Section V-B.

C. PUF Obfuscation and Post-Processing

Despite the robust security of keys and weights in the PUFiM encryption block, the FeFET-based PUF remains vulnerable to probing and modeling attacks. **To thwart these threats**, obfuscation and post-processing techniques are proposed.

Fig. 6(a) illustrates the startup value-based obfuscation. Upon power-on, the bistable circuits output random startup values due to process variations, which are used to activate one of the two rows of PUF bits in a block. Therefore, the PUF bits stored in the nonvolatile FeFETs are obfuscated. However, some bistable circuits may be less robust and output unstable

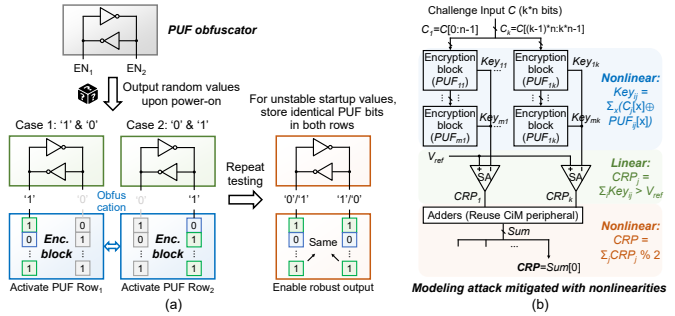


Fig. 6. PUF enhancement techniques. (a) Obfuscation. (b) Post-processing.

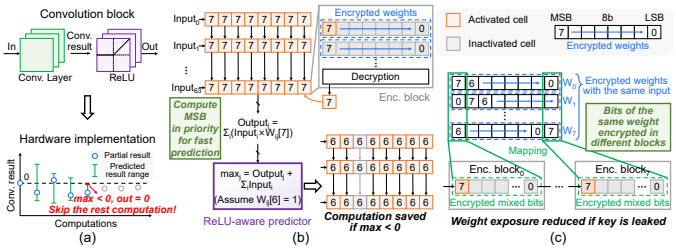


Fig. 7. Significance-aware weight mapping. (a) Insight of ReLU prediction. (b) Efficiency enhancement mapping. (c) Security enhancement mapping.

values. These circuits can be identified in advance, and the two PUF rows can be stored identically to enhance reliability at the cost of potential key leakage, as evaluated in Section V-D.

Fig. 6(b) shows the nonlinearity-enhancing post-processing with the complete CRP generation workflow. The XOR operations involved in key generation form the first nonlinear layer. Then m keys of the same column are aggregated in a linear approach. During post-processing, the adders designed for accumulating CiM outputs are **reused** to sum the results from k columns, with the LSB of the sum serving as the final CRP output. This additional nonlinearity increases the complexity of modeling the PUF without introducing additional area, and the improved resilience is shown in Section V-C.

D. Significance-Aware Weight Mapping

To further explore the potential of the PUFiM architecture based on encryption blocks, a significance-aware weight mapping scheme consisting of two sub-schemes is proposed.

Fig. 7(a) and (b) show the insight of the efficiency enhancement mapping. In convolution computation, the ReLU output would be 0 once the maximum possible value of the partial result is less than 0. As a result, the remaining computations can be skipped. Based on this observation, PUFiM maps all bits of one weight into the same encryption block and prioritizes the MSB computations in all blocks to accelerate prediction. If the predicted maximum output of any column is below 0, the corresponding blocks can be idled to save energy.

Fig. 7(c) shows the security enhancement mapping. The core idea is to shuffle the bits of the weights with the same input and map them into different encryption blocks without affecting the computations. This ensures that bits of the same weight are encrypted by different keys, significantly reducing the weight exposure caused by key leakage from any single block.

Both sub-schemes can be applied simultaneously to improve **energy efficiency and security** as shown in Section V-D.

TABLE I
PUFiM PARAMETERS

Unit	Spec	
FeFET array	Array size[1]	1024×1024 FeFETs 0.22fF/μm parasitic capacitance
	FeFET dimension[2]	Exp: 450nm×450nm Sim: 200nm×100nm
	Write voltage	-5.0V/2.2V/3.6V/4.5V, 500ns
	Read voltage	$V_1/V_2/V_3$ = 0.3V/0.95V/1.2V
	V_{TH} variation	130mV/85mV/176mV/110mV
	Capacitor	1fF MOM, σ = 0.8%, 2.74μm^2
ADCs & SAs	5-bit SAR ADC, VSA, 1 per column of blocks	

[1] The encryption block size is 8×10 based on the optimizations in Section V-B, and two rows per block store PUF bits in the MSBs.
[2] The FeFET layout in Fig. 9 is close to the fabricated ones.

V. Evaluation and Discussion

A. Experimental Setup

Hardware-level evaluation: To comprehensively capture FeFET characteristics, we utilize the experimental measurement results of 9×7 FeFET arrays fabricated at 28nm node to extract the C2C variation, MLC storage, and device non-idealities for accurate evaluation. The area, latency and energy consumption of FeFETs are simulated based on a widely used FeFET model [37] using the parameters from the fabricated FeFET arrays. 5-bit SAR ADCs and voltage SAs (VSAs) are designed, and digital peripherals are synthesized at 28nm node. The array size is set to 1024×1024 FeFETs with the parasitic capacitance considered. Table I lists the detailed information.

Software-level evaluation: Five powerful modeling attacks are performed, i.e., LR, MLP, SVM, XGBoost and CMA-ES, which have shown success against various PUF designs [25], [26], [38]. The inference accuracy is evaluated on VGG8 for CIFAR10 and ResNet18 for CIFAR100, covering different dataset scales and network complexity. We use PyTorch for quantization-aware training to obtain the model with 8-bit input and 8-bit weight. The inference process is modified to include the crypto operations and non-idealities from hardware.

Baselines: The modeling attack resilience is benchmarked with the arbiter PUF with and without 6-XOR structure [39] and the state-of-the-art FeFET PUF [23]. For secure CiM evaluation, four state-of-the-art designs based on SRAM [20], RRAM [18], STT-MRAM [21] and FeFET [19] are compared. The technology node of all baselines is scaled to 28nm using DeepScale [40] for accurate and apple-to-apple comparisons.

B. Circuit-level Evaluation and Optimization

Fig. 8 evaluates the reliability of the encryption block. As shown in Fig. 8(a), considering FeFET variations, the outputs of PUF units still exhibit tight distributions primarily induced by the CMOS variation, while the outputs of CiM units show an error rate of <2% due to the FeFET state overlap. This is attributed to the proposed MLC hybrid computation method, which converts FeFET readout results to CMOS currents. For key generation, the accumulation of output currents is sensed by the inverter-NMOS loop with <5% error rate, facilitating

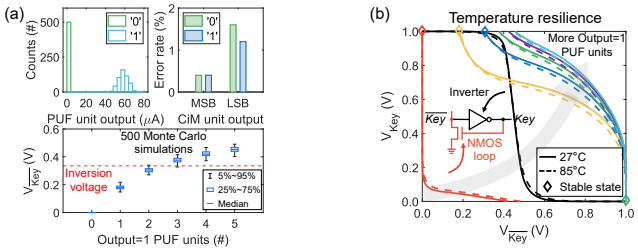


Fig. 8. Encryption block reliability: (a) Distributions and error rates of the PUF and CiM unit outputs. (b) Temperature resilience of key generation.

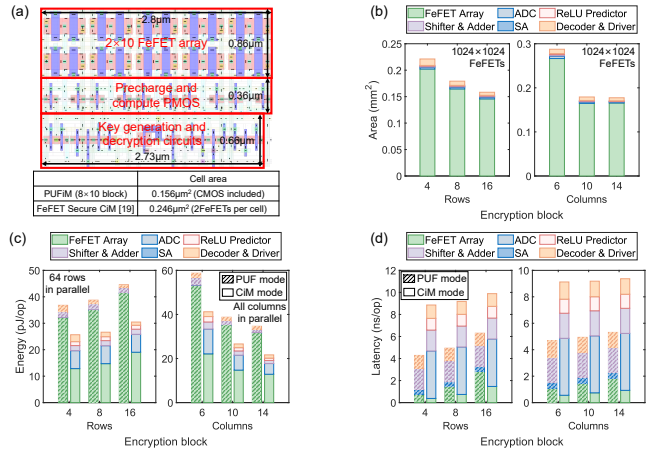


Fig. 9. Encryption block performance: (a) Layout of 2×10 block and cell area comparison with prior works [19]. (b) Area. (c) Energy. (d) Latency.

high randomness and uniqueness of the keys. Fig. 8(b) further shows the temperature resilience of the sensing circuits. Thanks to the voltage division between transistors in series, the intersection of the inverter-NMOS loop in-out curves, representing stable outputs, shifts negligibly as temperature varies from 27°C to 85°C, which indicates robust resilience.

Fig. 9(a) shows the layout of a 2×10 encryption block based on 28nm design rules. The width of the CMOS circuits perfectly matches the FeFETs, and the capacitor (2.74μm^2) can be placed on top. Consequently, PUFiM achieves 0.156μm^2 per cell with CMOS included, which is 1.6× lower than the complementary storage in [19] with the same device size.

Fig. 9(b-d) evaluate the PUFiM area, energy and latency in both PUF and CiM modes with different encryption block sizes. The FeFET array occupies most of the area as the blocks share the peripherals, and the resulting parasitic capacitance further contributes to a high energy proportion. Compared with the CiM mode, the PUF mode consumes more energy due to the power-hungry current summation required for key generation but shows less latency with the ADCs not involved.

For the impact of the encryption block size, increasing rows and columns enables the CMOS circuits to be shared by more FeFETs and reduces the area. Besides, increasing the rows within a block, while maintaining the same row parallelism, heavies the precharging burden and results in more energy and latency. While more columns have a similar effect on precharging, their impact on energy is covered by the reduction in column parallelism. With the above observations, we select **8×10** as the optimized encryption block size for PUFiM.

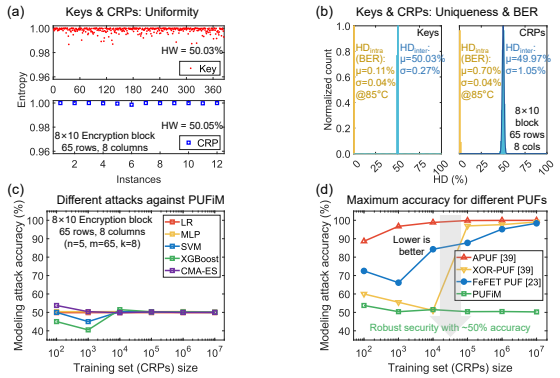


Fig. 10. PUF mode evaluation: (a) Uniformity. (b) Uniqueness and BER. (c) Modeling attack resilience of PUFiM. (d) Comparison with existing PUFs.

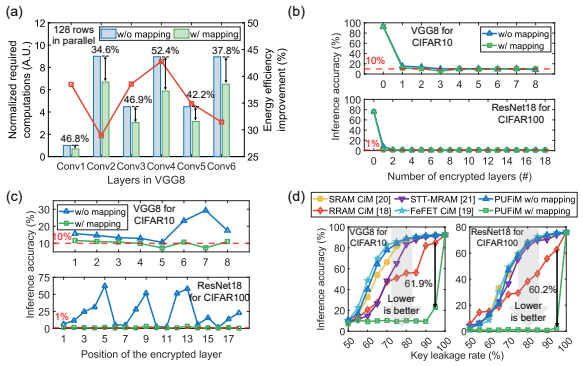


Fig. 11. CiM mode evaluation: (a) Energy efficiency improvement of the significance-aware weight mapping. (b) Encryption security performance. (c) Impact of encrypted layer position on security. (d) Key leakage resilience.

C. PUF Performance and Security Evaluation

Fig. 10(a) and (b) show the basic PUF metrics of PUFiM. Both the keys (within encryption blocks) and CRPs (after post-processing) show excellent ∼50% uniformity and uniqueness, and achieve an ultra-low <0.70% bit error rate (BER) at 85°C. This is attributed to the proposed reliable computation method and the use of FeFET C2C variations as the entropy source.

Fig. 10(c) shows the PUFiM modeling attack resilience across multiple attack models. Thanks to the nonlinearity provided by the proposed post-processing technique, PUFiM achieves a maximum accuracy of 50.26% (close to random guessing) even with a training set size of 10M, which is almost infeasible for the attacker to obtain. When applying these attacks to various PUFs, PUFiM shows significantly enhanced security compared with the vulnerable baselines.

D. CiM Performance and Security Evaluation

Fig. 11(a) shows the benefits of the proposed significance-aware mapping in VGG8. By accelerating the ReLU prediction convergence, the required computations are reduced with the related blocks idled, improving the PUFiM energy efficiency.

Fig. 11(b-d) evaluate the security performance of PUFiM inference. Encrypting a single layer drastically reduces accuracy to unusable levels, with the position of the layer affecting the accuracy drop. The proposed weight mapping scheme ensures consistently low accuracy levels close to random guessing. As shown in Fig. 11(d), thanks to the finer-grained encryption and the proposed mapping scheme, PUFiM achieves much higher security compared with the prior secure CiMs, lowering the accuracy by >60% under 95% key leakage on both models.

TABLE II
COMPARISON BETWEEN PUFIM AND SOTA SECURE CIMS

	[20]	[18]	[21]	[19]	**PUFiM**
Cell type	SRAM	RRAM	STT-MRAM	FeFET	**FeFET**
Technology	28nm	32nm	22nm	28nm	**28nm**
Bit/cell	1	1	1	2	**2**
PUF support	Weak	No	Weak	No	**Strong**[1]
Key delivery vulnerability	Yes	Yes	Yes	Yes	**No**[2]
Key leakage vulnerability[3]	74.88%	62.65%	73.88%	75.59%	**2.46%**
Storage density (Mb/mm^2)	0.05	6.85	0.73	0.61	**10.52** (>**1.5**×)
Compute density[4] ($TOPS/mm^2$)	0.004	0.046	0.044	0.011	**0.444** (>**9.7**×)
Energy efficiency[4] (TOPS/W)	2.86	0.33	12.24	8.08	**14.75** (>**1.2**×)

[1] PUFiM also achieves resilience against modeling attack and data theft on NVM PUFs.
[2] The keys are generated and used internally without the delivery via register chains.
[3] Inference accuracy of ResNet18 for CIFAR100 with 95% key leakage.
[4] Normalized to 8-bit input, 8-bit weight precision with 50% sparsity.

E. Macro-Level Comparison with SOTA Designs

Table II summarizes the macro-level comparison between PUFiM and the state-of-the-art secure CiMs [18]–[21]. **From a security perspective**, the merging of PUF and CiM eliminates the key delivery vulnerabilities, while the proposed PUF enhancement techniques and the significance-aware mapping further improve the resilience against PUF attacks and key leakage. This comprehensive security solution surpasses all the baselines in robustness for edge AI systems. **For area and energy efficiency**, thanks to the proposed MLC hybrid storage and computation, PUFiM achieves >1.5× storage density and >9.7× compute density compared with the baselines. Besides, the compact merging saves the peripherals, and the mapping scheme reduces the required computation, collectively achieving >1.2× energy efficiency improvement.

F. More Discussions

PUFiM is designed for edge inference, where infrequent weight updates mitigate FeFET endurance concerns. The reliability of MLC FeFETs can be further improved by write-and-verify [41]. Besides, resilience against chosen input attacks (i.e., collecting input-output pairs for retraining) and side-channel attacks can be improved through enhanced DNN model robustness [42] and masking techniques [43].

VI. CONCLUSION

This paper proposes PUFiM, a FeFET-based edge security solution with comprehensively robust security and significant efficiency improvement. The key highlight is the merging of PUF and CiM through MLC hybrid storage and computation, which enables highly secure local key generation and decryption. The obfuscation and post-processing techniques are proposed to enhance PUF security, and a significance-aware mapping scheme tailored for the PUFiM architecture is developed to improve CiM energy efficiency and security. Results show that PUFiM outperforms the state-of-the-art secure CiMs in security, area efficiency and energy efficiency.

ACKNOWLEDGMENT

This work is supported in part by NSFC (92264204, U21B2030, U24B6015) and in part by NSF (2008365).

REFERENCES

[1] Y. Kim, M. Imani, and T. S. Rosing, "Image Recognition Accelerator Design Using In-Memory Processing," *IEEE Micro*, vol. 39, no. 1, pp. 17–23, 2019.

[2] L. Yang *et al.*, "Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[3] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-Memory Computing in Emerging Memory Technologies for Machine Learning: An Overview," in *2020 DAC*, 2020, pp. 1–6.

[4] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," *IEEE Micro*, pp. 1–5, 2024.

[5] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, 2018.

[6] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[7] Y. Yuan *et al.*, "34.6 A 28nm 72.12TFLOPS/W Hybrid-Domain Outer-Product Based Floating-Point SRAM Computing-in-Memory Macro with Logarithm Bit-Width Residual ADC," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 576–578.

[8] S. Xie, C. Ni, A. Sayal, P. Jain, F. Hamzaoglu, and J. P. Kulkarni, "16.2 eDRAM-CIM: Compute-In-Memory Design with Reconfigurable Embedded-Dynamic-Memory Array Realizing Adaptive Data Converters and Charge-Domain Computing," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 248–250.

[9] T.-H. Wen *et al.*, "34.8 A 22nm 16Mb Floating-Point ReRAM Compute-in-Memory Macro with 31.2TFLOPS/W for AI Edge Devices," in *2024 ISSCC*, vol. 67, 2024, pp. 580–582.

[10] Y.-C. Chiu *et al.*, "A 22nm 4Mb STT-MRAM Data-Encrypted Near-Memory Computation Macro with a 192GB/s Read-and-Decryption Bandwidth and 25.1-55.1TOPS/W 8b MAC for AI Operations," in *2022 ISSCC*, vol. 65, 2022, pp. 178–180.

[11] Y. Xu *et al.*, "Embedding security into ferroelectric FET array via in situ memory operation," *Nature Communications*, vol. 14, p. 8287, 2023.

[12] T. Soliman *et al.*, "First demonstration of in-memory computing crossbar using multi-level Cell FeFET," *Nature Communications*, vol. 14, p. 6348, 2023.

[13] Y. Cai *et al.*, "Enabling Secure NVM-Based in-Memory Neural Network Computing by Sparse Fast Gradient Encryption," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1596–1610, 2020.

[14] N. Lin, X. Chen, H. Lu, and X. Li, "Chaotic Weights: A Novel Approach to Protect Intellectual Property of Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1327–1339, 2021.

[15] M. Xie, S. Li, A. O. Glova, J. Hu, and Y. Xie, "Securing Emerging Nonvolatile Main Memory With Fast and Energy-Efficient AES In-Memory Implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2443–2455, 2018.

[16] D. Reis *et al.*, "IMCRYPTO: An In-Memory Computing Fabric for AES Encryption and Decryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 5, pp. 553–565, 2022.

[17] W. Li, S. Huang, X. Sun, H. Jiang, and S. Yu, "Secure-RRAM: A 40nm 16kb Compute-in-Memory Macro with Reconfigurability, Sparsity Control, and Embedded Security," in *2021 IEEE Custom Integrated Circuits Conference (CICC)*, 2021, pp. 1–2.

[18] L. Zhao, Y. Zhang, and J. Yang, "SRA: a secure ReRAM-based DNN accelerator," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, p. 355–360.

[19] Z. Zhao *et al.*, "In-Situ Encrypted NAND FeFET Array for Secure Storage and Compute-in-Memory," in *2023 International Electron Devices Meeting (IEDM)*, 2023, pp. 1–4.

[20] S. Huang, H. Jiang, X. Peng, W. Li, and S. Yu, "Secure XOR-CIM Engine: Compute-In-Memory SRAM Architecture With Embedded XOR Encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 12, pp. 2027–2039, 2021.

[21] Y.-C. Chiu *et al.*, "A CMOS-integrated spintronic compute-in-memory macro for secure AI edge devices," *Nature Electronics*, vol. 6, pp. 534–543, 2023.

[22] L. Mankali, N. Rangarajan, S. Chatterjee, S. Kumar, Y. S. Chauhan, O. Sinanoglu, and H. Amrouch, "Leveraging Ferroelectric Stochasticity and In-Memory Computing for DNN IP Obfuscation," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 8, no. 2, pp. 102–110, 2022.

[23] H. Shao, B. Fu, J. Yang, W. Luo, C. Su, Z. Fu, K. Tang, and R. Huang, "IMCE: An In-Memory Computing and Encrypting Hardware Architecture for Robust Edge Security," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[24] Z. Chen *et al.*, "PUF-CIM: SRAM-Based Compute-In-Memory With Zero Bit-Error-Rate Physical Unclonable Function for Lightweight Secure Edge Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 8, pp. 1234–1247, 2023.

[25] U. Rührmair *et al.*, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, p. 237–249.

[26] T. Li *et al.*, "Demonstration of high-reconfigurability and low-power strong physical unclonable function empowered by FeFET cycle-to-cycle variation and charge-domain computing," *Nature Communications*, vol. 16, p. 189, 2025.

[27] B. Gao *et al.*, "Concealable physically unclonable function chip with a memristor array," *Science Advances*, vol. 8, no. 24, p. eabn7753, 2022.

[28] M. T. Rahman *et al.*, "The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes," in *2020 HOST*, 2020, pp. 262–272.

[29] M. Trentzsch *et al.*, "A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 11.5.1–11.5.4.

[30] H. Mulaosmanovic *et al.*, "Evidence of single domain switching in hafnium oxide based FeFETs: Enabler for multi-level FeFET memory cells," in *2015 IEDM*, 2015, pp. 26.8.1–26.8.3.

[31] S. Dünkel *et al.*, "A FeFET based super-low-power ultra-fast embedded NVM technology for 22nm FDSOI and beyond," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 19.7.1–19.7.4.

[32] S. Yoon, S.-I. Hong, D. Kim, G. Choi, Y. M. Kim, K. Min, S. Kim, M.-H. Na, and S. Cha, "QLC Programmable 3D Ferroelectric NAND Flash Memory by Memory Window Expansion using Cell Stack Engineering," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2023, pp. 1–2.

[33] S. G. Kirtania, K. A. Aabrar, A. I. Khan, S. Yu, and S. Datta, "Cold-FeFET as Embedded Non-Volatile Memory with Unlimited Cycling Endurance," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2023, pp. 1–2.

[34] S. Kannan, N. Karimi, O. Sinanoglu, and R. Karri, "Security Vulnerabilities of Emerging Nonvolatile Main Memories and Countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 2–15, 2015.

[35] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning Physically Unclonable Functions," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 1–6.

[36] J. Luo *et al.*, "Novel Ferroelectric Tunnel FinFET based Encryption-embedded Computing-in-Memory for Secure AI with High Area-and Energy-Efficiency," in *2022 IEDM*, 2022, pp. 36.5.1–36.5.4.

[37] S. Deng, G. Yin, W. Chakraborty, S. Dutta, S. Datta, X. Li, and K. Ni, "A Comprehensive Model for Ferroelectric FET Capturing the Key Behaviors: Scalability, Variation, Stochasticity, and Accumulation," in *2020 IEEE Symposium on VLSI Technology*, 2020, pp. 1–2.

[38] N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, "Neural-Network-Based Modeling Attacks on XOR Arbiter PUFs Revisited," Cryptology ePrint Archive, Paper 2021/555, 2021. [Online]. Available: https://eprint.iacr.org/2021/555

[39] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, p. 148–160.

[40] S. Sarangi and B. Baas, "DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era," in *2021 ISCAS*, 2021, pp. 1–5.

[41] Y. Zhou *et al.*, "A Compact Writing Scheme for the Reliability Challenges in 1T Multi-level FeFET Array: Variation, Endurance and Write Disturb," *IEEE Electron Device Letters*, pp. 1–1, 2024.

[42] D. Hendrycks, K. Lee, and M. Mazeika, "Using Pre-Training Can Improve Model Robustness and Uncertainty," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, 2019, pp. 2712–2721.

[43] T. Li *et al.*, "ProtFe: Low-Cost Secure Power Side-Channel Protection for General and Custom FeFET-Based Memories," *ACM Transactions on Design Automation of Electronic Systems*, 2023.

EnQode: Fast Amplitude Embedding for Quantum Machine Learning Using Classical Data

Jason Han*, Nicholas S. DiBrita*, Younghyun Cho†, Hengrui Luo*, Tirthak Patel*
Rice University, Santa Clara University†*

***Abstract*—Amplitude embedding (AE) is essential in quantum machine learning (QML) for encoding classical data onto quantum circuits. However, conventional AE methods suffer from deep, variable-length circuits that introduce high output error due to extensive gate usage and variable error rates across samples, resulting in noise-driven inconsistencies that degrade model accuracy. We introduce *EnQode*, a fast AE technique based on symbolic representation that addresses these limitations by clustering dataset samples and solving for cluster mean states through a low-depth, machine-specific ansatz. Optimized to reduce physical gates and SWAP operations, *EnQode* ensures all samples face consistent, low noise levels by standardizing circuit depth and composition. With over 94% fidelity in data mapping, *EnQode* enables robust, high-performance QML on noisy intermediate-scale quantum (NISQ) devices. Our open-source solution provides a scalable and efficient alternative for integrating classical data with quantum models.**

I. Introduction

As quantum computing advances toward broader applicability, one of its key challenges is interfacing classical data with quantum algorithms [40], [34]. Quantum machine learning (QML) has shown potential in fields ranging from material discovery to the physical sciences, with amplitude embedding (AE) being the critical mechanism for encoding classical data onto quantum states [11], [12], [25]. However, current AE approaches present significant limitations for practical implementation on noisy intermediate-scale quantum (NISQ) devices. Specifically, AE circuits often have a high depth and number of gates, leading to degraded model performance on error-prone architectures, where circuit depth and gate error directly impact output interpretability [28], [29].

Current AE implementations also struggle to maintain a uniform error profile across samples, as the circuit depth and gate composition inherently depend on the data values encoded [39], [32], [8]. The variable depth across samples results in unequal noise exposure – a problem that can obscure the true performance of quantum algorithms and limit their utility in practical applications. This variability not only exacerbates decoherence issues on quantum hardware but also introduces sample-specific noise, complicating the interpretability and fidelity of downstream QML tasks. *In such settings, the depth and structure of AE circuits need to be managed delicately to balance efficiency with noise resilience [7], [33], [20], underscoring the demand for a new data embedding approach.*

To address these issues, we introduce *EnQode*, a novel amplitude embedding framework designed to achieve low error and high efficiency through approximate embedding. Rather than attempting exact amplitude representation for each sample, *EnQode* leverages a cluster-based approach to approximate AE with high fidelity. Our method begins by grouping data samples using the k-means clustering algorithm, enabling us to identify representative states for each cluster that capture the main characteristics of the data distribution. For each cluster, we then train a low-depth, machine-optimized ansatz tailored to the quantum hardware of choice.

An important aspect of *EnQode* is its generalizability across different QML applications with symbolic representation of circuit parameters. Once trained, the cluster-based ansatz allows for the rapid generation of an AE circuit for new data samples by mapping them to their respective clusters and applying transfer learning. This enables *EnQode* to yield AE circuits with over 94% fidelity while maintaining a standardized depth across samples. By reducing the number of two-qubit gates and maintaining a consistent depth across all AE circuits, *EnQode* not only enhances fidelity but also ensures uniform error profiles across data samples.

Our evaluation shows that *EnQode* achieves significant improvements in efficiency and consistency over the Baseline approach across multiple metrics. *EnQode* reduces the circuit depth by 45% and the two-qubit gate count by 35%, with zero variability across samples due to its fixed ansatz design. In noisy IBM quantum hardware simulation settings, *EnQode* demonstrates a state fidelity improvement of 13% over the Baseline, highlighting its robustness to hardware noise. Furthermore, *EnQode* reduces the compilation time per sample by 36%, with an added offline overhead of only 1.4 seconds per dataset and class, making it both time-efficient and practical for real-time quantum machine learning applications.

II. Relevant Background

A. Qubits, Quantum Gates, and Hardware Noise

Qubits are the fundamental units of quantum computation, with each qubit in a superposition state $\alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2+|\beta|^2 = 1$. Quantum gates operate on qubits to modify their states, implementing unitary transformations in the form of matrix operations. A unitary operator on n qubits $U \in \mathrm{U}(2^n)$ satisfies the constraints $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of U.

In NISQ devices, gate noise, decoherence, and gate variability significantly impact circuit fidelity, particularly as circuit depth increases [31], [9], [16]. The circuit depth d for an amplitude embedding circuit directly correlates with error accumulation, leading to reduced fidelity in the final quantum state $|\psi_{out}\rangle$ due to decoherence effects. Gate error rates also

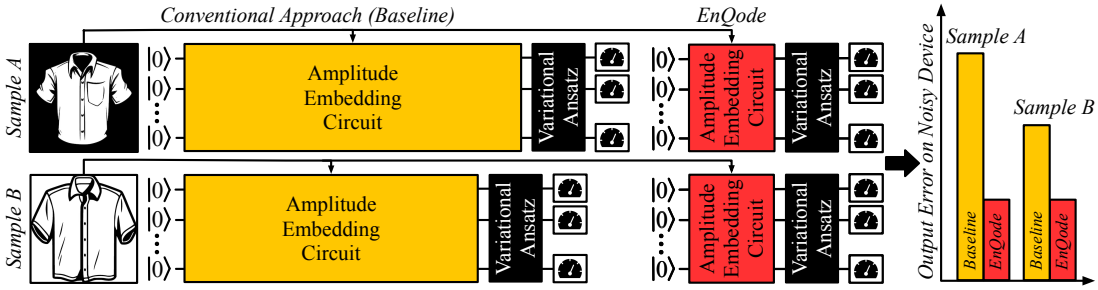


Fig. 1: *EnQode* provides short-depth and consistent circuits for amplitude embedding across different dataset samples.

vary across types of gates, with certain gates like SWAP being particularly error-prone – these gates do not add logical computation to the circuit but help connect distant qubits – which underscores the need for low-depth and reduced-SWAP implementations for reliable AE [3], [21], [37].

B. Variational Quantum Circuits and Optimization

Variational Quantum Circuits (VQCs) play a central role in QML, leveraging a combination of parameterized gates and classical optimization techniques to minimize an objective function [9], [18]. A typical VQC can be expressed as:

$$|\phi(\theta)\rangle = U(\theta)\,|0\rangle^{\otimes n}, \tag{1}$$

where $\theta = (\theta_1, \theta_2, \ldots, \theta_l)$ are trainable parameters and $U(\theta)$ is a unitary operator constructed from these parameters for an n-qubit state. The goal in variational optimization is to find parameters θ^* that minimize a given loss function $L(\theta)$, often achieved through gradient-based optimization:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta^{(t)}), \tag{2}$$

where η is the learning rate.

C. Quantum Data Encoding and Amplitude Embedding

AE is a technique that encodes classical data into the amplitudes of a quantum state, mapping a classical vector $\vec{f} = (f_1, f_2, \ldots, f_{2^n})$ to a quantum state $|\psi\rangle$ such that:

$$|\psi\rangle = \textstyle\sum_{i=1}^{2^n} f_i\,|i\rangle\,, \tag{3}$$

where $\sum_{i=1}^{2^n} |f_i|^2 = 1$ to satisfy the normalization constraint of quantum states. This process allows classical data to be represented as quantum amplitudes, making it possible to utilize quantum algorithms for ML tasks. In practice, AE requires unitary operators to construct the quantum state $|\psi\rangle$ from an initial state $|0\rangle$ [8], [34], [28].

D. Challenges with Amplitude Embedding on NISQ Devices

The reliance on precise amplitude encoding leads to challenges on NISQ devices, where variable-depth circuits introduce inconsistencies in noise exposure across samples. Given that an amplitude embedding circuit for a sample $\vec{f}$ may require $d(\vec{f})$ layers of gates, where $d(\vec{f})$ varies with the encoded values, noise variability becomes an inherent issue:

$$\left|\psi(\vec{f})\right\rangle = U(\vec{f})\,|0\rangle^{\otimes n}, \tag{4}$$

where $U(\vec{f})$ is the unitary operator tailored for encoding $\vec{f}$, and n is the number of qubits required for the embedding. As demonstrated in Fig. 1, traditional AE circuits lack uniform depth, leading to inconsistent fidelity across samples when used for a downstream variational QML task [11], [12], [25].

The effectiveness of VQCs is contingent on the data encoding, which impacts the circuit's ability to represent the problem space accurately. Inconsistent and noisy amplitude embeddings degrade the training process, as each sample faces a different noise profile depending on the circuit depth and composition of its AE circuit. This dependence on high-quality, low-noise encoding underscores the need for efficient AE techniques.

E. Need for EnQode: Low-Depth Approach to Consistent AE

EnQode is designed to overcome the limitations of traditional AE by clustering data samples and representing each cluster with a machine-optimized, low-depth ansatz. By clustering samples into k clusters, *EnQode* represents the mean state of each cluster $\vec{c}_i$ using a low-depth ansatz that is optimized offline. For each cluster, we solve for a unitary operator $U(\vec{c}_i)$ that approximates AE with high fidelity:

$$|\psi(\vec{c}_i)\rangle = U(\vec{c}_i)\,|0\rangle^{\otimes n}. \tag{5}$$

This approach can then be used to perform fast AE for any given dataset sample. Additionally, *EnQode* reduces and standardizes the depth across all AE circuits, ensuring that each sample experiences consistent noise exposure, as depicted in Fig. 1. By achieving high-fidelity, low-noise embeddings, *EnQode* provides a robust framework for scalable QML applications, mitigating the issues faced by traditional AE methods on NISQ devices. We describe *EnQode* next.

III. Key Concepts and Design

We begin by describing the ansatz that *EnQode* leverages to amplitude embed any given quantum states from a dataset.

A. EnQode's Ansatz for Hardware-Efficient Implementation

The *EnQode* ansatz is designed to leverage the IBM quantum computing hardware basis for optimal efficiency and fidelity (but can be designed for any other hardware basis). On IBM quantum devices, the native gate set consists of the ECR two-qubit gate and the $R_z(\theta)$, X, and SX one-qubit gates. Since R_z gates are implemented virtually in software through a frame-of-reference change, they introduce no additional error, making them ideal for parameterized rotations in our ansatz.

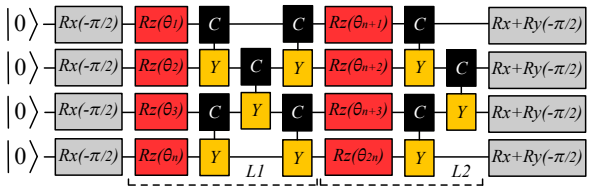


Fig. 2: *EnQode*'s ansatz design to reduce the circuit depth and the number of physical gates. CY gates are indicated with C for the control qubit and Y for the target qubit.

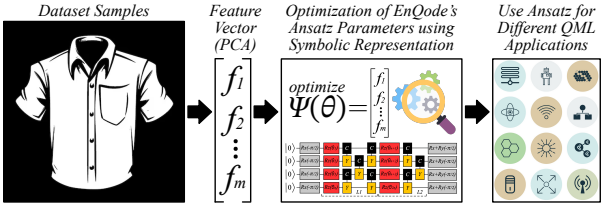


Fig. 3: *EnQode* employs symbolic representation for fast optimization of its amplitude embedding circuit ansatz.

Because amplitude-embedded states are real-valued with no complex phase, we first apply $R_x(\frac{-\pi}{2})$ gates to each qubit to rotate them from the $|0\rangle$ state in the z-x plane to the $|+i\rangle$ state in the x-y plane on the Bloch sphere (Fig. 2). This setup aligns the qubits in the plane and is suitable for purely R_z-based parameterization. Following this, we implement the ansatz shown in Fig. 2, which consists of parameterized R_z gates to maintain fidelity, as R_z gates introduce no noise.

For entanglement, we employ controlled-Y (CY) gates. Although controlled-X, Y, and Z gates all require a similar number of ECR and one-qubit gates, thus having similar noise cost, CY gates are preferred here due to their specific interaction properties that preserve the x-y plane alignment in the Bloch sphere post-rotation.

The CY gates are arranged in a compact layout that alternates from layer to layer, as depicted in Fig. 2, enabling interaction among all qubits within each layer of the ansatz while minimizing gate count. Through extensive experimentation, we identified this alternating configuration as the optimal sparse arrangement on IBM's heavy-hexagonal connectivity layout (linear section). It requires no additional SWAP gates, maximizing fidelity and minimizing error per layer.

We apply multiple layers of this structured ansatz, with each layer refining the AE while ensuring consistent qubit interactions. Finally, $R_x(-\frac{\pi}{2})$ and $R_y(-\frac{\pi}{2})$ gates are applied to return the qubits to the z-x plane, completing the embedding process with minimal depth and maximal fidelity. This hardware-efficient design directly reduces error-prone physical gates and provides a high-fidelity, low-noise AE circuit.

B. Symbolic Representation for Efficient Optimization

A key innovation in *EnQode* is its use of symbolic representation to achieve fast and accurate amplitude embedding. Unlike traditional approaches that rely on numerical approximations for each optimization step, *EnQode* represents the parameters of all R_z gates symbolically, allowing the amplitude embedded state vector $|\psi(\theta)\rangle$ to be fully expressed as an analytical function of these parameters. Given n qubits, let $\theta = (\theta_1, \theta_2, \ldots, \theta_l)$ denote the set of R_z gate parameters in the ansatz. The intermediate state vector $|\psi(\theta)\rangle$ that is optimized in the ansatz can be written as:

$$|\psi(\theta)\rangle = \frac{1}{\sqrt{2^n}} \begin{bmatrix} k_1 e^{i\frac{\sum_{j=1}^{l} p_{1j}\cdot\theta_j}{2}} \\ k_2 e^{i\frac{\sum_{j=1}^{l} p_{2j}\cdot\theta_j}{2}} \\ \vdots \\ k_{2^n} e^{i\frac{\sum_{j=1}^{l} p_{2^n j}\cdot\theta_j}{2}} \end{bmatrix}, \tag{6}$$

where $p_{rj} \in \{-1, 0, 1\}$ and $k_r \in \{-1, -i, i, 1\}$, for all $1 \leq r \leq 2^n, 1 \leq j \leq l$.

We make this symbolic representation to leverage the following insight: by converting the amplitudes into relative phases, we are able to have a compact expression with explicit, simple partial derivatives (as we simply take the derivative of an exponential function composed with a linear function), which we can leverage in gradient-based optimization. After optimizing the fidelity for our statevector in this compact representation, we perform a change of basis to convert the relative phases back to real-valued amplitudes used for amplitude embedding.

This approach allows for direct calculation of the impact of each R_z parameter on the state vector, providing an efficient and precise mapping from the circuit's symbolic parameters to the target amplitude values. This symbolic linkage between parameters and output state is unique in that it eliminates the need for repeated numerical recalculations, which are computationally expensive and prone to rounding errors in conventional approaches.

The optimization task in *EnQode* is to match each element $\psi_i(\theta)$ of the parameterized state vector to the target real amplitude value x_i, representing the cluster mean. Specifically, *EnQode* seeks the optimal parameters θ^* such that $\psi_i(\theta^*) \approx x_i$ for each i, where x_i are the target values from the data vector. The symbolic representation with simple derivatives enables *EnQode* to leverage an efficient gradient-based optimization algorithm. For this work, we use the Limited-memory BFGS algorithm because it empirically performed the best compared to other algorithms we tested [38] [23]. As a quasi-Newton method, it relies on accurate Jacobian computations, which are computed efficiently through our symbolic representation. We compute gradients and estimate the inverse Hessian using finite differences, which proved sufficient for effective optimization. However, this work can easily be extended by computing the Jacobian symbolically before running the optimization.

As visually represented in Fig. 3, this approach dramatically accelerates convergence, allowing *EnQode* to complete training faster than traditional iterative methods that require extensive numerical calculations and lack direct control over each parameter's effect on the final state. This unique design forms the foundation of *EnQode*'s ability to achieve rapid, high-fidelity amplitude embeddings.

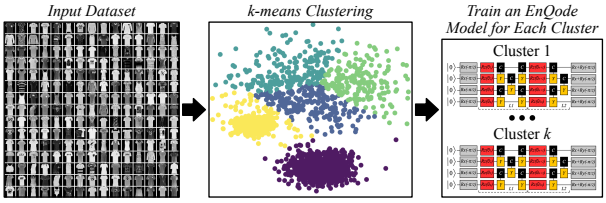


Fig. 4: *EnQode* divides a dataset into multiple clusters and trains a symbolic representation model for each cluster.

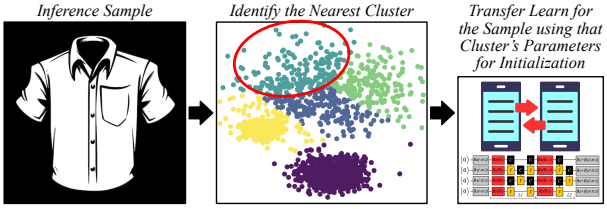


Fig. 5: An inference sample gets matched to one of the clusters, and its amplitude embedding circuit is quickly trained using the cluster's trained model to initialize the parameters.

C. EnQode's Use of Clustering for Offline Training

To enable efficient and noise-resilient AE, *EnQode* uses k-means clustering [22] to partition the dataset into k clusters. Each sample $\vec{x}$ in the dataset is assigned to one of these clusters based on its similarity to the cluster centroids, producing k clusters of similar data samples. For each cluster, we calculate the mean sample, $\vec{c}_i$, representing the central characteristics of that cluster. This mean sample serves is used to train the embedding circuit for each cluster.

Once the clusters are defined, *EnQode* performs "offline" training by applying the hardware-optimized ansatz discussed previously to the mean sample of each cluster. Using the symbolic representation optimization technique, the ansatz parameters for each cluster's mean sample $\vec{c}_i$ are optimized to maximize fidelity with minimal circuit depth. As this training approach leverages symbolic representations, it significantly reduces the computational overhead, completing the training for all clusters for a given class in a dataset in under 2 seconds. This rapid optimization allows each cluster's ansatz to be fine-tuned to capture the structure of its mean sample, effectively encoding the main features of all samples within the cluster.

As shown in Fig. 4, this clustering-based approach allows *EnQode* to achieve high-fidelity amplitude embedding efficiently across diverse datasets. By training on cluster means, *EnQode* reduces redundant computations and ensures that the embedding circuit depth and noise profile remain consistent, thereby providing a scalable solution for QML applications.

The trained cluster models are then stored and used to support "online" training and inference, as we describe next.

D. Transfer Learning for Fast Amplitude Embedding

With thousands or millions of samples in a dataset, it is critical that these states are able to be amplitude embedded into quantum circuits as quickly as possible (in "real-time") for fast training and inference. *EnQode* uses transfer learning from pre-trained cluster models to enable fast amplitude embedding for incoming samples. When a new sample $\vec{x}_{\text{new}}$ arrives, it is assigned to the nearest cluster based on its Euclidean distance to the cluster centroids. This quick assignment identifies the most relevant pre-trained model for the sample, allowing *EnQode* to initialize $\vec{x}_{\text{new}}$'s embedding circuit with the optimized parameters of the corresponding cluster mean. The parameters are then fine-tuned from that initialization using the same algorithm described in Section III-B.

As shown in Fig. 5, this initialization provides a close approximation for the embedding of $\vec{x}_{\text{new}}$, minimizing the need for further optimization and significantly accelerating the embedding process. By beginning with cluster-trained parameters, *EnQode* achieves high-fidelity AE with consistent circuit depth and noise profile across samples, all without retraining from scratch. This transfer learning approach ensures that *EnQode* can embed both training and inference samples rapidly, making it well-suited for large datasets and real-time quantum machine learning workflows.

E. Putting All of EnQode's Design Elements Together

EnQode integrates clustering, symbolic optimization, and transfer learning to create an efficient AE framework. By clustering data and training a hardware-optimized ansatz for each cluster mean using symbolic representation, *EnQode* achieves a rapid, low-overhead setup with optimized parameters. For each new sample, transfer learning initializes its embedding circuit using the pre-trained parameters of the nearest cluster, enabling consistent, fast embedding without extensive re-optimization.

The ansatz design maximizes hardware efficiency through virtual R_z gates and a sparse CY layout, minimizing circuit depth and noise. **Together, these elements provide *EnQode* with a scalable, high-fidelity solution suitable for real-time quantum machine learning on NISQ devices, maintaining uniform performance across samples.**

IV. Methodology

A. Experimental Setup and Framework

We run our compliation experiments and classical processing tasks on our local computing cluster. The cluster consists of nodes with the AMD EPYC 7702P 64-core processor with x86_64 architecture and a 2.0 GHz clock. We spawn virtual machines (VMs) on these nodes consisting of 8 cores, 32 GB memory, and 32 GB storage for each of our experiments, providing more than sufficient resources for each experiment. The VMs are resource-bounded and not overprovisioned, ensuring that each experiment has exclusive access to the hardware resources assigned to it without any interference, which helps us provide accurate and consistent timing analysis.

For implementing and evaluating *EnQode*, we wrote Python modules. We used `qiskit` [2] for representations of symbolic and parameterized circuits, as well as for density matrix fidelity calculations. We used `qiskit_aer` to perform ideal and noisy density matrix simulations given the characteristics of the `ibm_brisbane` quantum computer [5]. We used

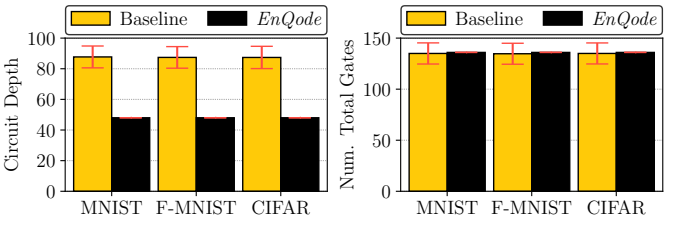


Fig. 6: *EnQode* achieves lower circuit depth and an almost identical total number of gates than the Baseline, with no variability due to fixed circuit – the Baseline displays high variability for both metrics.

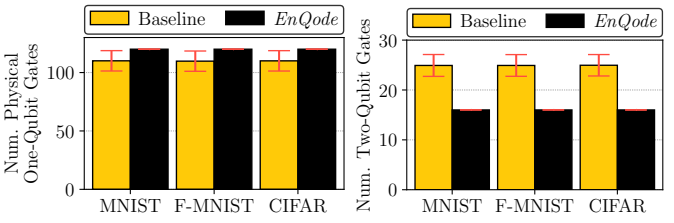


Fig. 7: *EnQode* also achieves similar one-qubit and lower two-qubit physical gates than the Baseline and has no variability due to a fixed circuit design – in contrast, the Baseline displays high variability for both the metrics.

`scipy` [38] to implement the L-BFGS optimization algorithm. For all experiments, we used 4 qubit circuits, with a 4 layer ansatz structure as depicted in Fig. 2. The number of clusters is chosen such that the *state fidelity* (defined below) between any datapoint and its nearest cluster is at least 0.95.

For implementing and evaluating the Baseline, we use the `qiskit` state preparation algorithm [2] [14]. For transpilation onto IBM hardware, we use the transpilation optimization level of 0 to eliminate the influence of confounding factors such as synthesis techniques for our analysis. All of our metrics are evaluated using circuits transpiled to `ibm_brisbane`.

B. Evaluation Datasets

To evaluate *EnQode*, we use three standard image datasets: MNIST [10], Fashion-MNIST (F-MNIST) [42], and CIFAR-10 (CIFAR) [17]. We randomly sampled 5 classes for each dataset, and for each class, we randomly sampled 500 images from that class. For each dataset, we apply Principal Component Analysis (PCA) [24] to reduce dimensionality, focusing on retaining the most significant features. The resulting features are then normalized to ensure compatibility with AE. This preprocessing step allows for efficient, high-fidelity embedding of each dataset in *EnQode*'s framework.

C. Assessment Metrics

We compare *EnQode*'s performance to the conventional exact amplitude embedding technique [30] [14] – referred to as Baseline. For all circuit-based metrics, we exclude R_z gate counts, as these can be implemented virtually. First, we assess the *circuit depth* and the *total number of gates* used for amplitude embedding with Baseline and *EnQode* – we examine the mean and standard deviation behaviors. These are the key metrics that affect circuit fidelity on NISQ devices. We also look at the *number of physical one-qubit and two-qubit gates*. Second, we evaluate the *compilation times* to generate the amplitude embedded circuits. We also investigate the *state fidelity* of how closely the amplitude embedded state matches the desired state. We use a common metric for mixed states defined as $F(\rho,\sigma) = \left(tr\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2$ [15], where ρ and σ are the two quantum states (one desired and one measured). We perform these assessments in ideal and noisy scenarios.

V. Evaluation and Analysis

A. EnQode Reduces the Circuit Depth and Two-Qubit Gate Count (and Their Variability) Over the Baseline Approach

As shown in Fig. 6, *EnQode* achieves a significant reduction in circuit depth (45%) on average compared to the Baseline, with the added advantage of complete consistency across data samples. The Baseline approach suffers from variability in circuit length due to the data-dependent nature of traditional AE, resulting in fluctuations in both circuit depth and the number of gates required. In contrast, *EnQode* standardizes the ansatz length and gate composition, ensuring a fixed circuit structure for each dataset.

Similarly, Fig. 7 illustrates that *EnQode* slightly increases one-qubit but considerably reduces two-qubit gate counts compared to the Baseline while completely eliminating gate count variability. By using a fixed arrangement of one-qubit R_z gates and a sparse CY entangling structure, *EnQode* ensures that each sample faces a uniform noise profile, essential for stable performance in QML applications. The Baseline approach shows high variability in both gate types, with some samples requiring up to 10% more gates than others.

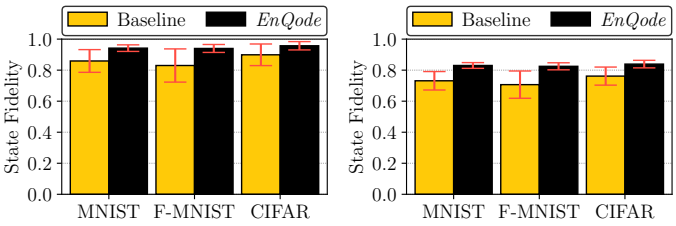
The reduction in circuit depth, two-qubit gate count, and variability provided by *EnQode* directly enhances reliability in noisy quantum computing environments. With a fixed circuit structure, *EnQode* minimizes exposure to noise, ensuring that each sample is treated uniformly. This consistency is particularly beneficial on NISQ devices, where circuit variability directly impacts fidelity. The stability and reduced complexity of *EnQode*'s circuits are expected to yield improved performance under noise, a hypothesis we test next.

B. EnQode's Generated Circuits Achieve High and Consistent State Fidelity in the face of Hardware Noise Effects

To assess *EnQode*'s effectiveness in representing a given data point state, we evaluate state fidelity across ideal simulations and noisy hardware simulation executions.

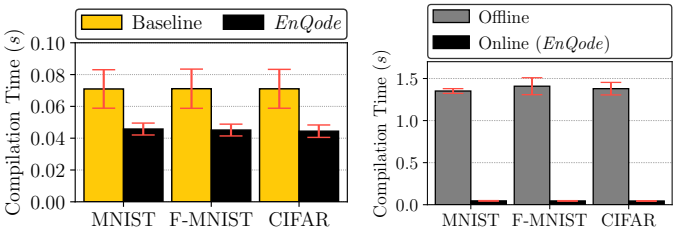
In ideal simulations (Fig. 8(a)), *EnQode* achieves a higher state fidelity of 9.6% than Baseline circuits when transpiled to real IBM hardware. This improvement highlights the high simulated fidelity that *EnQode* offers. Although Baseline circuits have near-perfect fidelity as they logically synthesize the state, when transpiling the circuit to the hardware basis set and connectivity, the ideal state fidelity was decreased.

In addition, under noisy simulation conditions (Fig. 8(b)), *EnQode* significantly outperforms the Baseline (by 13%). The Baseline circuits, which rely on deep circuits with variable



(a) Ideal Simulation (b) Noisy Simulation

Fig. 8: (a) *EnQode* achieves high state fidelity in ideal simulation. (b) In addition, it also achieves a higher fidelity than the Baseline in noisy NISQ simulation settings.



(a) Compilation Times (b) *EnQode*'s Time Overhead

Fig. 9: (a) *EnQode* achieves lower compilation times than the Baseline. (b) *EnQode*'s online compilation time is low. Its offline compilation time (for training the clusters) is also low.

lengths, suffer from high error accumulation in noisy environments, leading to substantial variability in state fidelity across samples. In contrast, *EnQode* maintains higher fidelity and reduced variability due to its consistent, low-depth circuit design. This stability in noisy settings underscores the advantage of *EnQode*'s fixed-length ansatz, which minimizes and standardizes exposure to noise-induced errors.

Overall, *EnQode*'s ability to maintain higher fidelity in noisy hardware environments demonstrates its practicality for quantum machine learning on NISQ devices. By ensuring lower circuit depth and consistent gate structure, *EnQode* provides a reliable amplitude embedding approach that is more robust to noise, a crucial attribute for achieving stable performance in real-world quantum applications.

C. EnQode Achieves Lower Compilation Time Compared To the Baseline and has a Low Offline Compilation Time

To evaluate *EnQode*'s efficiency in generating AE circuits, we compare its compilation times with the Baseline, breaking down the online and offline components. Fig. 9(a) shows that *EnQode* achieves an average speedup of 36% per sample compared to the Baseline due to *EnQode*'s symbolic representation and transfer learning from pre-trained cluster means. Since *EnQode*'s online compilation starts optimization using parameters from the closest cluster mean, it remains highly efficient for embedding new samples in real-time.

In Fig. 9(b), we provide a comparison of *EnQode*'s offline and online compilation times. The offline compilation step involves training the ansatz for each cluster mean, which is performed only once per dataset and class and thus introduces a minimal overhead of less than 2 seconds.

As a result, *EnQode* achieves rapid online compilation without sacrificing fidelity or adding significant preprocessing costs. The combination of low offline and online compilation times underscores *EnQode*'s practicality for real-time QML applications, where speed and efficiency are crucial.

VI. Related Work

The challenge of encoding classical data onto quantum circuits has spurred numerous methods for state preparation and feature embedding, but none directly address the unique requirements of AE with noise resilience and consistency across samples. Existing approaches primarily focus on general state preparation techniques [13], [43], [26], [4], [35], [41], which can lead to variable-depth circuits that suffer from inconsistencies in noise exposure and fidelity.

Quantum transfer learning [27] offers valuable insights into high-dimensional data adaptation but does not cater to AE's demand for low-noise, standardized circuit structures. Similarly, applications in drug discovery [19] and image classification [1] integrate quantum embedding methods. Yet, they rely on embedding mechanisms that often lack circuit depth control, making them unsuitable for AE, where consistent noise handling is critical for uniform sample treatment.

More recent efforts have tackled feature-specific embeddings [36] and variational techniques [6], optimizing for fidelity and efficiency in QML. However, these methods do not account for AE-specific needs like fixed circuit depth and low-noise environments for high-fidelity encoding. *EnQode* introduces the first AE-specific approach that achieves consistent, high-fidelity embeddings by clustering data samples and designing a low-depth, machine-optimized ansatz. By standardizing circuit depth and minimizing gate variability across samples, *EnQode* ensures uniform noise resilience, setting a new standard for practical AE in scalable QML.

VII. Conclusion

In this work, we introduced *EnQode*, an efficient AE framework designed to address the limitations of the traditional AE method on quantum devices. By employing a cluster-based approach with low-depth, symbolic representation ansatz, *EnQode* achieves consistent circuit depth and gate composition across samples, effectively mitigating noise variability and high error rates. Our solution delivers over 94% fidelity in the fast embedding of classical data, providing a robust foundation for QML. As an open-source tool, *EnQode* offers a practical pathway for more reliable data encoding, making it well-suited for diverse applications on quantum platforms.

Acknowledgement. This work was supported by Rice University, the DOE Quantum Testbed Finder Award DE-SC0024301, and the National Science Foundation NSF-DMS 2412403. It was also supported by the Ken Kennedy Institute and Rice Quantum Initiative, which is part of the Smalley-Curl Institute. *EnQode*'s code is open-sourced at `https://github.com/positivetechnologylab/EnQode`.

REFERENCES

[1] Mahabubul Alam, Satwik Kundu, Rasit Onur Topaloglu, and Swaroop Ghosh. Quantum-classical hybrid machine learning for image classification (special session). In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7. IEEE, 2021.

[2] Gadi Aleksandrowicz, Thomas Alexander, et al. Qiskit: An Open-source Framework for Quantum Computing, January 2019.

[3] Sashwat Anagolum, Narges Alavisamani, Poulami Das, Moinuddin Qureshi, and Yunong Shi. Élivágar: Efficient quantum circuit search for classification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 336–353, 2024.

[4] Israel F Araujo, Daniel K Park, Francesco Petruccione, and Adenilton J da Silva. A divide-and-conquer algorithm for quantum state preparation. *Scientific reports*, 11(1):6329, 2021.

[5] Davide Castelvecchi. IBM's Quantum Cloud Computer Goes Commercial. *Nature News*, 543(7644):159, 2017.

[6] Shouvanik Chakrabarti, Xuchen You, and Xiaodi Wu. Iccad special session paper: Quantum variational methods for quantum applications. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7. IEEE, 2021.

[7] Tianlong Chen, Zhenyu Zhang, Hanrui Wang, Jiaqi Gu, Zirui Li, David Z Pan, Frederic T Chong, Song Han, and Zhangyang Wang. Quantumsea: In-time sparse exploration for noise adaptive quantum circuits. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 51–62. IEEE, 2023.

[8] Manuel P Cuéllar, C Cano, Luis G Baca Ruíz, and Lorenzo Servadei. Time series quantum classifiers with amplitude embedding. *Quantum Machine Intelligence*, 5(2):45, 2023.

[9] Gennaro De Luca. A survey of nisq era hybrid quantum-classical machine learning research. *Journal of Artificial Intelligence and Technology*, 2(1):9–15, 2022.

[10] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[11] J Ismael Díaz-Ortiz, Axel Villanueva, and Francisco Delgado. Strongly entangling neural network: Quantum-classical hybrid model for quantum natural language processing. In *International Conference on Mathematical Modeling in Physical Sciences*, pages 503–514. Springer, 2023.

[12] José-Enrique García-Ramos, Álvaro Sáiz, José M Arias, Lucas Lamata, and Pedro Pérez-Fernández. Nuclear physics in the era of quantum computing and quantum machine learning. *Advanced Quantum Technologies*, page 2300219, 2024.

[13] Niels Gleinig and Torsten Hoefler. An efficient algorithm for sparse quantum state preparation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 433–438. IEEE, 2021.

[14] Raban Iten, Roger Colbeck, Ivan Kukuljan, et al. Quantum circuits for isometries. *Physical Review A*, 93(3):032318, 2016.

[15] Richard Jozsa. Fidelity for mixed quantum states. *Journal of modern optics*, 41(12):2315–2323, 1994.

[16] Bikram Khanal and Pablo Rivas. Evaluating the impact of noise on variational quantum circuits in nisq era devices. In *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*, pages 1658–1664. IEEE, 2023.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] Jinyang Li, Ang Li, and Weiwen Jiang. Quapprox: A framework for benchmarking the approximability of variational quantum circuit. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 13376–13380. IEEE, 2024.

[19] Junde Li, Mahabubul Alam, M Sha Congzhou, Jian Wang, Nikolay V Dokholyan, and Swaroop Ghosh. Drug discovery approaches using quantum machine learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1356–1359. IEEE, 2021.

[20] Tingting Li, Liqiang Lu, Ziming Zhao, Ziqi Tan, Siwei Tan, and Jianwei Yin. Qust: Optimizing quantum neural network against spatial and temporal noise biases. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[21] Zhiding Liang, Jinglei Cheng, Hang Ren, Hanrui Wang, et al. Napa: intermediate-level variational native-pulse ansatz for variational quantum algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[22] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[23] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[24] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993.

[25] Alexey Melnikov, Mohammad Kordzanganeh, Alexander Alodjants, and Ray-Kuang Lee. Quantum machine learning: from physics to software engineering. *Advances in Physics: X*, 8(1):2165452, 2023.

[26] Ar A Melnikov, Alena A Termanova, Sergey V Dolgov, Florian Neukart, and MR Perelshtein. Quantum state preparation using tensor networks. *Quantum Science and Technology*, 8(3):035027, 2023.

[27] Soronzonbold Otgonbaatar, Gottfried Schwarz, Mihai Datcu, and Dieter Kranzlmüller. Quantum transfer learning for real-world, small, and high-dimensional remotely sensed datasets. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2023.

[28] Tirthak Patel, Daniel Silver, and Devesh Tiwari. Optic: A practical quantum binary classifier for near-term quantum computers. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 334–339. IEEE, 2022.

[29] Agi Prasetiadi and Masahiro Mambo. Parity-based amplitude embedding for rational quantum dataset preprocessing. In *2024 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, pages 96–100. IEEE, 2024.

[30] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504, 2019.

[31] Priyabrata Senapati, Zhepeng Wang, Weiwen Jiang, Travis S Humble, Bo Fang, Shuai Xu, and Qiang Guan. Towards redefining the reproducibility in quantum computing: A data analysis approach on nisq devices. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 468–474. IEEE, 2023.

[32] Daniel Silver, Tirthak Patel, William Cutler, Aditya Ranjan, et al. Mosaiq: Quantum generative adversarial networks for image generation on nisq computers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7030–7039, 2023.

[33] Daniel Silver, Tirthak Patel, Aditya Ranjan, Harshitta Gandhi, William Cutler, and Devesh Tiwari. Sliq: quantum image similarity networks on noisy quantum computers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9846–9854, 2023.

[34] Daniel Silver, Tirthak Patel, and Devesh Tiwari. Quilt: Effective multi-class classification on quantum computers using an ensemble of diverse quantum classifiers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8324–8332, 2022.

[35] Ionel Stetcu, Alessandro Baroni, and Joseph Carlson. Projection algorithm for state preparation on quantum computers. *Physical Review C*, 108(3):L031306, 2023.

[36] Napat Thumwanit, Chayaphol Lortaraprasert, and Rudy Raymond. Trainable discrete feature embeddings for quantum machine learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1352–1355. IEEE, 2021.

[37] Teague Tomesh, Pranav Gokhale, Victory Omole, Gokul Subramanian Ravi, Kaitlin N Smith, Joshua Viszlai, Xin-Chuan Wu, Nikos Hardavellas, Margaret R Martonosi, and Frederic T Chong. Supermarq: A scalable quantum benchmark suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 587–603. IEEE, 2022.

[38] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[39] Hanrui Wang, Yongshan Ding, Jiaqi Gu, et al. Quantumnas: Noise-adaptive search for robust quantum circuits. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 692–708. IEEE, 2022.

[40] Hanrui Wang, Jiaqi Gu, Yongshan Ding, Zirui Li, Frederic T Chong, David Z Pan, and Song Han. Quantumnat: quantum noise-aware training with noise injection, quantization and normalization. In *Proceedings of the 59th ACM/IEEE design automation conference*, pages 1–6, 2022.

[41] Hanyu Wang, Daniel Bochen Tan, and Jason Cong. Quantum state preparation circuit optimization exploiting don't cares. *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2024.

[42] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[43] Julien Zylberman and Fabrice Debbasch. Efficient quantum state preparation with walsh series. *Physical Review A*, 109(4):042401, 2024.

Supporting Register-based Addressing Modes for in-DRAM PIM ISAs

Seok Young Kim[1], Byung Ho Choi[1], Seokwon Kang[2], Yongjun Park[2], and Seon Wook Kim[1]
[1]Department of Electrical Engineering, Korea University, Seoul, Korea
[2]Department of Computer Science, Yonsei University, Seoul, Korea
{tjrdud5136, pupopup, seon}@korea.ac.kr, kswon0202@gmail.com, yongjunpark@yonsei.ac.kr

***Abstract*—Processing-in-Memory architecture presents a promising solution to alleviate the data movement bottleneck that arises from transferring data between memory and compute units in traditional processor-centric systems, particularly for DNN applications. However, this architecture introduces two inherent overheads: PIM code offloading and data transferring between CPU and memory. To address these issues, we propose two register-based addressing modes, indexed and base-offset addressing, for DMA descriptor-based in-DRAM PIM ISAs. Our full-system performance evaluation demonstrates that the approach significantly reduces the overheads, resulting in up to 1.94x speedup compared to the baseline PIM, additionally only with 4.65% area and 8.61% power consumption.**

***Index Terms*—PIM ISA, Register-based Addressing Mode, Processing-in-DRAM, Look-up Table, Code Offloading.**

I. Introduction

Processing-in-Memory (PIM) is one of the attractive solutions to alleviate the data movement bottleneck caused by data transfer between memory and compute units in traditional computing systems. This bottleneck has become more severe over time due to the widening performance gap between processors and memory and the rapid growth in the size of transformer model parameters, which increase tenfold each year [1]. To address this challenge, memory manufacturers like SK Hynix and Samsung have developed PIM prototypes [2]–[4], and some companies like UPMEM [5] have brought commercial products to market. Although the rapid increase in model parameters provides a significant opportunity for PIM acceleration, current PIM architectures have two intrinsic performance overheads from the PIM code offloading and minimal PIM execution capability.

Fig. 1 shows the execution time breakdown using our PIM-testbed platform, which is used in Section IV. In this setup, we computed Generalized Matrix-Vector multiplication (GeMV), Generalized Matrix-Matrix multiplication (GeMM), and element-wise operations on the PIM, represented as PIM execution in the figure, and the remaining operations on the CPU. We further categorized the remaining, i.e., the non-PIM-capable operations, into two categories: 1) CPU execution (Unary: unary function execution) and 2) CPU execution (the Others). By offloading the PIM-capable operations comprising BLAS levels 1, 2, and 3 to PIM, we fully exploited DRAM's internal bandwidth and the PIM's computation parallelism, achieving an average speedup of 2.23x across four different transformer models over the CPU-only execution. However, the PIM code offloading overhead incurs an average of 9.65%, and the data transfer does an average of 35.97% of the total execution time. These two overheads represent a significant portion, making up 45.61% of the total execution time, thus preventing the PIM's performance scalability.

The following two structural characteristics in DNN's Computational Graph (CG) negatively impact PIM's performance.

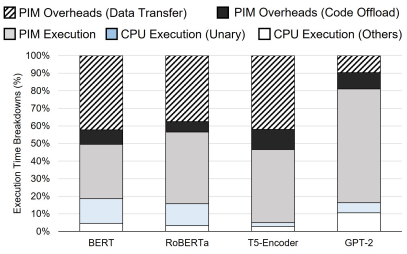


Fig. 1. The execution time breakdown of the transformer-based models in the PIM+CPU platform [6] with a sequence length of 16.

First, activation functions such as GeLU and SwishGLU [7], along with other unary operators like sqrt, reciprocal, and power, which PIM cannot execute, are often interspersed with operators that PIM can perform, like GeMV(M) and element-wise Add/Sub/Mul. Although the execution times of these functions are relatively short, their positioning between PIM-executable operators leads to significant data transfer overhead between the CPU and PIM. Second, the GeMM operations use nearly identical operation sizes ({p, q, r}) in specific model execution, and PIM codes represent the static physical addresses as operands. However, due to the dynamic allocation of input matrices (tensors), it is necessary to offload PIM codes each time, even though we repeatedly execute the same function with identical input sizes using PIM [2], [4], incurring the code offloading overhead.

In this paper, we introduce two register-based addressing modes, indexed and base-offset addressing for the DMA descriptor-based in-DRAM PIM ISAs called PISA [6], to maximize the operator capability and minimize the PIM code offloading overhead, which is critical to PIM performance. The original PISAs utilize statically generated absolute addressing, embedding the entire physical address within PIM instructions to denote operands [6]. In contrast, our register-based modes use PIM register values to reconstruct PISAs at runtime. While this technique is widely used in CPU execution, it is less common in PIM execution. Each addressing mode offers substantial performance improvements, as detailed below.

An indexed addressing mode: The current DNN models include various unary operators, some highly computationally intensive. We can execute the unary operators using Look Up Tables (LUTs), which is very well-suited to the execution characteristics of DRAM-based PIM. DRAM's large capacity allows us to use an unlimited number of large-sized LUTs to provide high accuracy and perform many LUT-based executions simultaneously. More importantly, while executing memory-intensive applications like DNN models, it is essential to reduce memory requests as much as possible. Therefore,

instead of exhaustive accessing the entire LUT ($2^{16} \times 2$ bytes for bfloat16 [8]), we build an address to access the LUT by concatenating the unary function's LUT start address and the computed PIM results (R_{vACC}), thus providing the tolerant performance during the multi-workload execution while accelerating the unary execution using PIM over CPU. Also, the PIM support of the unary operators, i.e., increasing the PIM's operator capability, provides the PIM operator fusion opportunity to a compiler while minimizing the data transfer between the CPU and PIM.

A base-offset addressing mode: PIM repeatedly executes the same operators with the same input sizes in the recent models [9]–[12]. Statically generated PIM codes use absolute addressing for operands, resulting in overhead from reloading the PIM code each time when the function is executed with different operands. To address this, we introduce argument registers, R_{args}s, in the PIM Interface Unit (PIU) to store the start address of operands. Functions are programmed using these argument registers and operand offsets. Like indexed addressing, the PIU combines the argument registers and the offsets at runtime to generate operand addresses. This approach eliminates the need to reload the PIM codes repeatedly, significantly reducing the code offloading overhead.

We implemented a system-level evaluation platform based on an ARM SoC using the PIM-emulated FPGA to validate our proposal and conducted a complete system stack evaluation using four transformer-based models, BERT [9], RoBERTa [10], T5-Encoder [11], and GPT-2 [12]. As a baseline, i.e., without supporting our proposal, PIM [6] achieved a speedup of 1.76x to 3.59x compared to the CPU. Our base-offset addressing mode almost eliminated the code offloading overhead, and indexed addressing mode for supporting LUT-based unary operators in PIM significantly reduced data transfer overhead and accelerated their execution simultaneously. Also, the LUT-based PIM execution allowed us to fuse PIM operators aggressively, thus eliminating memory spill between them. As a result, we could achieve 1.26x to 1.94x speedups only with 4.65% of the total area and 8.61% of the overall power consumption overhead compared to the baseline PIM design.

The remainder of this paper consists of the following: In Section II, we explain our baseline PIM architecture and the characteristics of recent transformer-based models for motivating our work. In Section III, we describe the proposed register-based addressing PIM architecture, and in Section IV, we evaluate our PIM design's performance. Finally, Section V concludes the paper.

II. Background

A. Baseline PIM Architecture: Silent-PIM and PISA-DMA

Many state-of-the-art in-DRAM PIMs [2], [5] target accelerator-based rather than memory-centric designs, thus not servicing the memory requests from other processes during the PIM computation and possibly necessitating modifications to the overall architecture layer. To overcome the disadvantages, Silent-PIM [13] has been proposed, which aimed for a memory-centric design, i.e., using a per-bank approach. It preserves the standard JEDEC DRAM interface, thus guaranteeing the simultaneous execution of both PIM and non-PIM processes and requiring little system modification, which is why we selected it as the baseline platform for our work.

PIM workloads sequentially read large amounts of data from memory to perform the same repetitive computations, such as MAC. This memory access characteristic is the same as

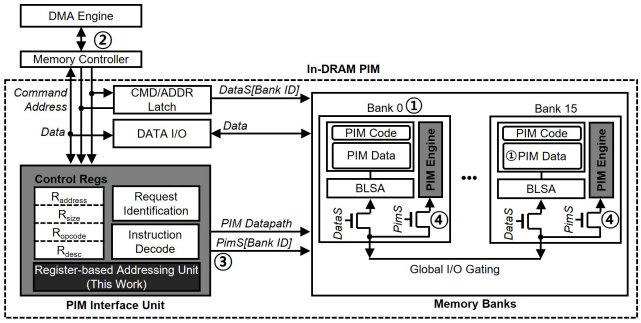


Fig. 2. Overview of the baseline architecture, including the PIM interface unit and the PIM engine. The gray blocks depict the extra components integrated into DRAM for PIM operations, with some darker ones, including the white text, representing the outcomes of this study.

DMA, so Silent-PIM utilizes DMA as an operand offloading engine by representing the operand address and its size and utilizing the unused fields to represent PIM opcodes in the DMA descriptor. Therefore, the PIM ISA was developed using the DMA descriptor, called PISA-DMA, and the PIM code was presented as the sequence of the DMA descriptors. Intuitively, completing the DMA transaction is the same as committing an instruction.

Silent-PIM's execution sequences using PISA-DMA are shown in Fig. 2 as follows: ① CPU stores the PIM code consisting of PISA-DMA descriptors in PIM memory and its start address and size in the descriptor register (R_{desc}) of PIU. Each PISA-DMA descriptor has the operand information such as the source/destination operand start addresses and the operand size, the PIM opcode, and other unused fields. ② CPU activates the DMA engine to start the DMA transaction. ③ The DMA engine fetches the descriptor from the PIM memory. The PIU stores the fetched descriptor in a descriptor latch forwarded to the DMA engine. At the same time, PIU determines whether it is the PISA-DMA descriptor by matching with the information in R_{desc}. If it is, the PIU stores the operand information in the operand registers ($R_{address}$, R_{size}) in the request identification unit of the PIU to distinguish the PIM memory requests from the non-PIM memory ones and the opcode in an opcode register (R_{opcode}) to configure the PIM datapath. ④ During the DMA engine transfers memory requests, the PIU recognizes the PIM operand by address matching using the information in the operand registers and turns on the pimS switch to provide the operand to the PIM engine for computation. ⑤ The pre-configured datapath computes with incoming operands and stores the results in the accumulator registers, R_{vACC}.

B. Characteristics of Recent Transformer Models

As the datasets used to train transformer-based models evolve, their models are increasing, and operators are becoming more complex due to the evolution of datasets to include images, text, code, and human language [14]–[16].

The PIM architecture enhances the efficiency of DNN models by leveraging MACs for GeMV(M) and element-wise operations. However, its design constraints limit its ability to support more intricate functions such as trigonometric, hyperbolic, and exponentiation. Consequently, these complex operations necessitate offloading to a CPU, resulting in data transfers to and from PIM memory and even preventing the compiler from fusing consecutive PIM operators. Such data

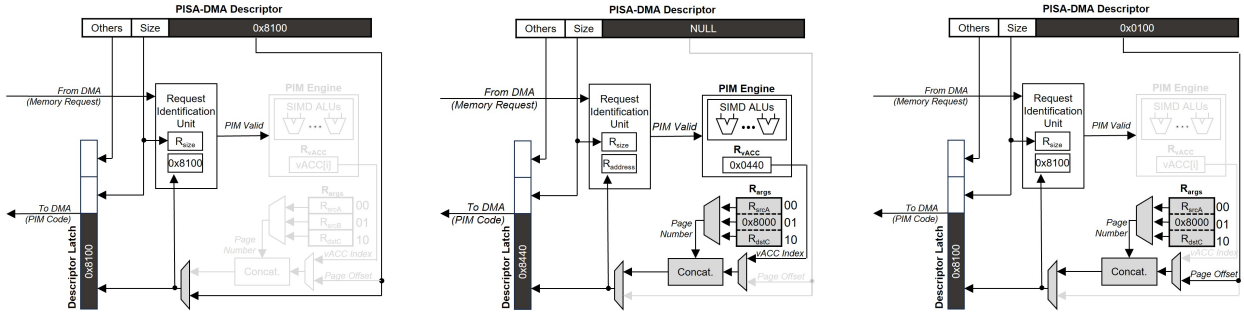


Fig. 3. The overall information flow of the proposed register-based addressing PIM, modifying the source field of the PISA-DMA descriptor at runtime (colored in black) and adding the circuit (colored in gray). (a) Absolute addressing. (b) Indexed addressing. (c) Base-offset addressing.

TABLE I
THE PIM OPERATOR SEQUENCE LENGTH AND THE RATIOS OF PIM-CAPABLE NODES.

Model		BERT		RoBERTa		T5-Encoder		GPT-2	
		Set (A)	Set (B)	Set (A)	Set (B)	Set (A)	Set (B)	Set (A)	Set (B)
OSL	2∼8	27	33	76	100	50	40	48	78
	Over 9	0	4	0	12	0	12	0	0
Ratio of PIM Nodes (%)		28.1	48.8	40.8	70.3	19.3	43.8	7.2	17.4

transfers are particularly detrimental to PIM's performance since we allocate the PIM memory uncacheable.

To gain deeper insights into and quantify the limitations of the PIM architecture, we introduce the concept of PIM Operator Sequence Length (OSL), reflecting the number of consecutive PIM operators. The longer the PIM OSL, the less the data movement between CPU and PIM, and the higher the opportunity a compiler fuses. We investigated the PIM OSL using four representative transformer-based models and classified their ONNX operators into two sets. Set(A) involves the operators intrinsically performed by PIM, such as BLAS operations (GeMV, GeMM, Add, Sub, Mul), and Set(B) does Set(A) *plus unary operations*, such as Reciprocal, Sqrt, Log, Abs, Neg, and so on. Table I shows that by supporting the unary operators in PIM, we can significantly increase the ratio of PIM-capable nodes from a minimum of 10.2% (17.4% - 7.2% in GPT-2) to a maximum of 29.5% (70.3% - 40.8% in RoBERTa) and the PIM OSL, which implies substantial performance improvement by eliminating the data transfer, accelerating the unary execution, and applying the operator fusion.

Applying LUT operations in DRAM-based PIM architectures faces a few drawbacks. AIM [2] utilizes one specific DRAM row as LUT; therefore, the DRAM row size constrains the LUT size, and the constraint limits the accuracy of LUT operations. To compensate for this accuracy drop, it adopts a Piecewise Linear Approximation (PLA) technique. Also, since it uses all-bank execution, it stores identical LUT contents (pre-computed results) across all banks. HBM-PIM [4] does not support any specific LUT operation, and instead, it uses the RISC-type core in DRAM to compute the unary operations. Therefore, the computation takes a considerable time.

Given that many PIM workloads, especially GeMV in transformer models, repeatedly execute operations with identical operand sizes, it is possible to eliminate the need for code offloading by simply changing the starting address of the tensor rather than re-generating and re-loading the entire PIM code. For example, in the case of GPT-2 [12], only 4 different sizes of GeMV operations ({p,768,768}, {p,768,2304}, {p,768,3072}, and {p,3072,768}) are used, and each is called 12 times. In more extreme cases, for RoBERTa [10] and T5-encoder [11], the GeMV operation with {p, 768, 768} requires offloading the PIM code 48 times. Thus, there is an excellent opportunity to reduce the code offloading overhead by applying a register-based addressing mode, such as using an argument register to use the base address of tensors.

III. DESIGN OF REGISTER-BASED ADDRESSING MODES

A. Operational Concept

Fig. 3 illustrates the information flow of the register-based addressing PIM system.

Our proposal reconstructs the DMA's source or destination operand address, called an effective address (EA), by introducing three argument registers, R_{args}, and by using the computation results in R_{vACC} in the PIM datapath. In addition to the absolute addressing in the baseline PIM, we proposed two more addressing modes; thus, our register-based addressing unit supports three addressing modes: ① Absolute addressing: EA ← $R_{address}$ (unchanged) (Fig. 3(a)), ② Indexed addressing: EA ← R_{args}[i] | R_{vACC} (Fig. 3(b)), and ③ Base-offset addressing, EA ← R_{args}[i] | [operand offset] (Fig. 3(c)).

Fig. 3(a) depicts an example of the operation in the absolute addressing mode, as in the baseline PIM [13]. In this mode, we load the PISA-DMA descriptor (e.g., 0x8100) into the descriptor latch without any change and store the operand's start address and size into the $R_{address}$ and R_{size} registers. Once the DMA engine fetches the operands from 0x8100 to 0x8100+size, the PIU recognizes the PIM memory requests and sends a valid signal to the PIM engine for computation.

In indexed addressing mode (Fig. 3(b)), we reconstruct the PIM source operand address by concatenating one of the argument register values (R_{args}[i]) to indicate the start address of a table and R_{vACC}. It is suitable for LUT operations. For example, to compute $sin(x)$, we store the starting address of the LUT for a sine function (0x8000) in R_{args}[01] and the PIM computation result x (0x0440) in R_{vACC}.

The base-offset addressing mode (Fig. 3(c)) is the reverse of the index addressing mode. The source operand address is determined by concatenating the argument register's value and a partial value of the operand address in the descriptor, i.e., the value of R_{args}[01] (0x8000) with the value (0x0100) in the

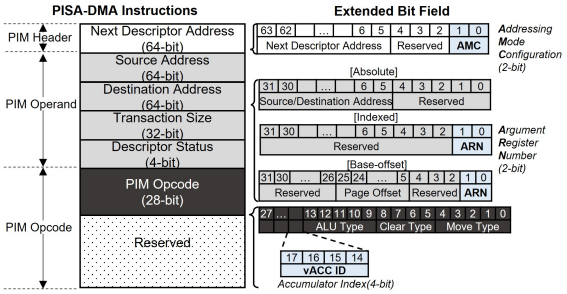


Fig. 4. PISA-DMA extension for supporting register-based addressing mode. The blue-colored component represents the modified fields in this study.

descriptor. The same function with the same operand size but a different operand can perform the computation without newly offloading PIM code into memory but only by changing the argument register values, like the CPU's linkage convention, and using the partial value as the operand offset in code.

B. PISA-DMA and Hardware Extension

1) Addressing Modes: We categorized the PISA-DMA descriptor according to its role into a header, an operand, and an opcode (Fig. 4). The 64-bit header stores the next descriptor's address; in the header, we use 2 bits out of the unused 5 Least Significant Bits (LSBs) to store Addressing Mode Configuration (AMC) information. The AMC value of 2'b00 indicates absolute addressing, 2'b01 indicates indexed addressing, and 2'b10 indicates base-offset addressing. The register-based addressing unit decodes the AMC bits to determine how to construct an effective address.

2) Argument Registers: For reconstructing an operand's effective address at runtime, i.e., to support the indexed and base-offset addressing modes, we introduce three 11-bit argument registers, R_{args}s, since 1) our PIM computation library uses at most three operands (two sources of srcA and srcB and one destination of dstC), and 2) we allocate PIM operands as 2M-byte huge pages using the PIM MemPool [6], resulting in an 11-bit page number and a 21-bit page offset. We distinguish R_{args}s by a 2-bit Argument Register Number (ARN), and 2'b00, 2'b01, and 2'b10 correspond to srcA, srcB, and dstC, respectively, in the 2-bit source/destination address field (bit 0 to 1) in the descriptor.

Given our 4G-byte memory system, we use only 32 bits out of the 64-bit source/destination address field in the descriptor, ignoring the LSB 5 bits due to the 32-byte DRAM access granularity. The absolute addressing mode uses the 27-bit source/destination address (bit 5 to 31) without any change to represent the read and write operand's start address, i.e., its effective address; thus, the mode does not use any R_{args}s.

3) Indexed Addressing Mode: We implemented an indexed addressing mode for computing unary operators in PIM, specifically in the form of $y = f(x)$, where x is the computation result from an immediate predecessor node, and $f(\cdot)$ represents a lookup table of pre-computed values. We store a start address of f in one of R_{args}s using ARN, take x from one of 16 R_{vACC}s within a bank, and concatenate them to make an effective address, i.e., a PISA-DMA descriptor's source address. We extend one more PIM opcode field, a 4-bit R_{vACC} index (bit 14 to 17), to access x, and thus, compute 256 distinct input x values, i.e., take x 256 times (16 elements per bank across 16 banks) instead of accessing an entire size of LUT.

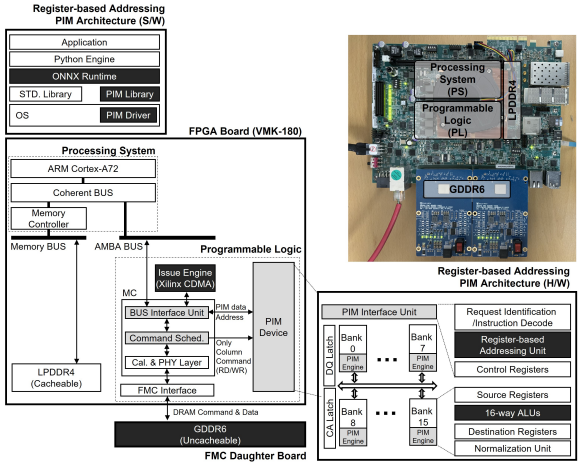


Fig. 5. The register-based addressing PIM architecture evaluation platform.

4) Base-Offset Addressing Mode: By adopting a base-offset addressing mode, we can pre-store descriptors, i.e., PIM codes in memory with a constant huge page offset (bit 5 to 25 source/destination address field in the descriptor) in the at compile time and update a huge page number at runtime using R_{args}s. This runtime address binding allows us to reuse PIM code and eliminate code offloading overhead. General PIM computations use three PIM operands (A, B, C), each associated with different R_{args}s.

IV. Performance Analysis

A. Experimental Methodology

Fig. 5 shows our PIM-based emulation platform and its specification for system-level performance evaluation. We evaluated the area and power overheads using a $65nm$ Samsung CMOS PDK, with a 200MHz cycle time constraint for the PIM device, due to the lack of a publicly available DRAM cell library. For FPGA implementation, we used the Xilinx Vivado [19] to synthesize the PIM device hardware and the PetaLinux to deploy Linux-based full-system software, including the PIM driver. We used four pre-trained transformer models for the experiment: BERT (29M parameters: 4 layers, 512 hidden dim), RoBERTa (125M: 12 layers, 768 hidden dim), T5-Encoder (220M: 12 layers, 768 hidden dim), and GPT-2 (117M: 12 layers, 768 hidden dim).

B. Transformer-based Models

Fig. 6 shows the speedups of the proposed register-based addressing PIMs over a baseline PIM architecture, PISA-DMA [6], and their time breakdowns with sequence lengths of 1 and 16 while running four transformer-based models. Our baseline PIM architecture (Ⓐ) achieves a speedup of 1.76x to 3.59x over a CPU. However, its PIM code offloading overhead results in 14.63% and 9.65%, and data transfer overhead does 16.81% and 35.97% of the total execution time on average at sequence lengths of 1 and 16, as discussed in Section I. As a sequence length grows, the amount of data transfer between CPU and PIM memory increases, thus dominating the overall overhead.

Our proposed base-offset addressing mode (Ⓑ) almost eliminated the code offloading overhead in all the models and sequence lengths, thus achieving the speedups of 1.11x, 1.23x, 1.14x, and 1.21x over the baseline (Ⓐ) with a sequence length

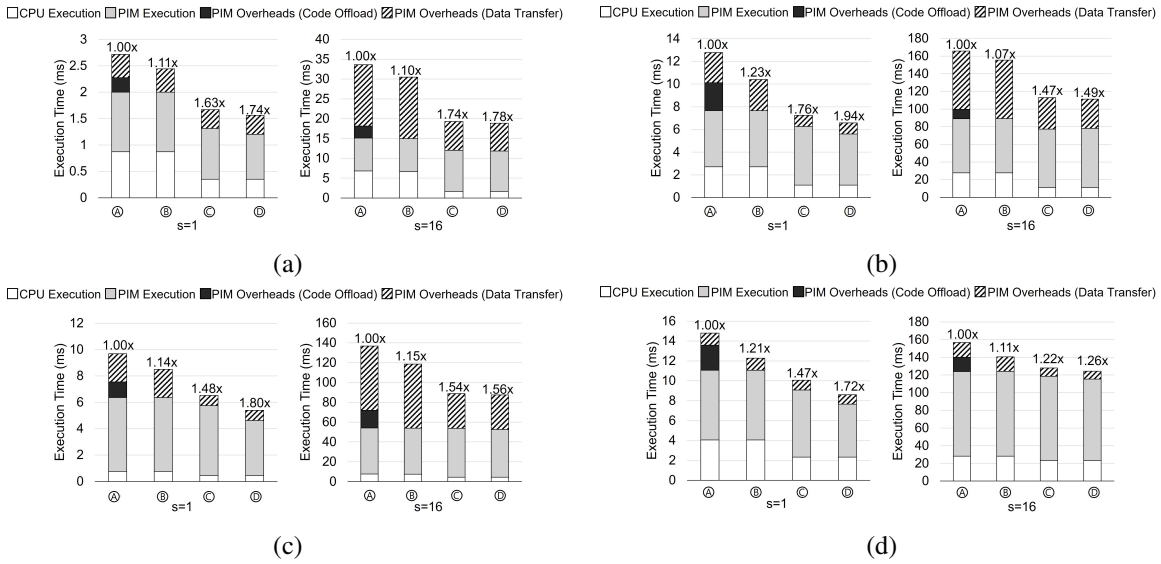


Fig. 6. The speedups of the proposed register-based addressing PIMs over a baseline PIM architecture [6] and their time breakdowns with two sequence lengths (s) of 1 and 16. (a) BERT. (b) RoBERTa. (c) T5-Encoder. (d) GPT-2. Ⓐ: PISA-DMA (baseline) [6], Ⓑ: Baseline+Base-offset addressing, Ⓒ: Baseline+(Base-offset+Indexed) addressing, Ⓓ: Baseline+(Base-offset+Indexed) addressing+Operator fusion).

of 1 in BERT, RoBERTa, T5-Encoder, and GPT-2, respectively, and an average of 1.17x. The larger the size of the PIM-capable operator, the higher the code offloading overhead; thus, the higher the performance gain from the base-offset addressing mode, as shown in RoBERTa and GPT-2, which use larger weight matrices. As the sequence length increases to 16, the data transfer dominates the total overhead; thus, the performance gain from the base-offset addressing decreases slightly, achieving speedups of 1.10x, 1.07x, 1.15x, and 1.11x.

Our indexed addressing mode for supporting LUT-based unary operators in PIM (Ⓒ) to expand PIM's computational capabilities reduced data transfer overhead and accelerated their execution simultaneously. We could apply LUT-based PIM execution to 40, 112, 91, and 148 nodes, which allowed us to obtain more speedups of 0.52x, 0.53x, 0.34x, and 0.26x with a sequence length of 1 and 0.64x, 0.30x, 0.39x, and 0.11x with a sequence length of 16 in BERT, RoBERTa, T5-Encoder, and GPT-2, respectively than Ⓑ. By using the mode, we could reduce the data transfer overhead of Ⓑ by 3.52% to 43.36%.

The higher PIM capability from the indexed addressing mode provides more opportunities to apply operator fusion to reduce spills, appearing as (Ⓓ) in the figure. Compared to Ⓒ, we achieved 0.11x, 0.18x, 0.32x, and 0.25x higher speedups in BERT, RoBERTa, T5-Encoder, and GPT-2, respectively, and 0.22x on average with a sequence length of 1, and slightly less performance gain with a sequence length of 16 due to intrinsic larger data transfer overhead than a sequence length of 1. We applied operator fusion for successive PIM-capable operators, where each operator has only one successor. Compared to the PIM OSL analysis in Table I, the operator fusion's performance impact is relatively limited. We observed that in many cases, the PIM-capable operator has more than one successor, resulting in data flow divergence that requires storing outgoing edge values (i.e., spills) for other successors anyway, thus diminishing the advantage of the operator fusion. The proportion of memory spills due to operator fusion within the total PIM memory requests required to execute the graph decreases as the sequence length increases.

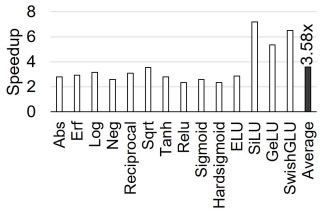


Fig. 7. Speedup of the proposed indexed-addressing based LUT execution in various unary functions over CPU.

C. Unary Functions

Fig. 7 shows the speedup achieved by applying the indexed addressing mode-based LUT technique to execute various unary functions on the PIM, compared to the CPU. Unlike the CPU, which exhibits varying execution times depending on the computational complexity of each unary function, the PIM achieves constant execution times due to using the LUT technique and an average speedup of 3.58x. For SiLU, GeLU, and SwishGLU, we observed up to a 7.19x speedup compared to the activation functions used in state-of-the-art large language models such as Bloom [21] and LLaMA [22]. The smaller the LUT size, the lower the accuracy. However, it's important to note that the LUT size does not impact the execution time, as we access the LUT using indexed addressing mode.

The proposed LUT approach accesses only one element in LUT corresponding to the input addresses, i.e., only 256 times (16 elements in R_{vACC} per bank $times$ 16 banks) per unit computation, resulting in lower memory bandwidth usage than accessing all LUT elements. This effectively minimizes memory interference between PIM applications and non-PIM applications. The higher the memory interference in a multi-workload scenario, the more significant the increase in execution time overheads. We selected the SPEC benchmarks as non-PIM applications to validate this impact alongside a single

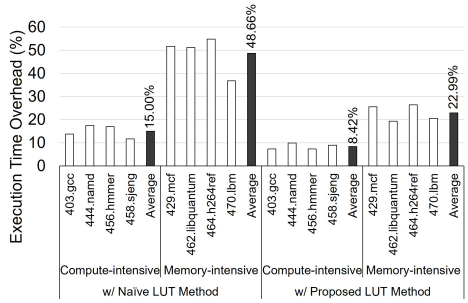


Fig. 8. Execution time overheads of the proposed indexed addressing-based and naïve LUTs when running multi-workload.

unary node's PIM execution. We excluded the benchmarks that caused out-of-memory errors in our experimental setup. We grouped the benchmarks into two categories: memory-bound applications (mcf, libquantum, h264ref, and lbm), which generate a high number of LLC misses, and compute-bound applications (gcc, namd, hmmer, and sjeng).

Fig. 8 shows the execution time overheads of each SPEC benchmark with executing the proposed and naïve LUT operation methods, compared to single workload execution. The naïve method accesses and broadcasts all LUT elements (i.e., $2^{16} \times 2$ bytes) to R_{vACC}s without the indexed addressing mode and matches the value in R_{vACC} with the LUT element address for the unary computation. The naïve LUT approach incurred an average overhead of 15.00% for the compute-intensive and 48.66% for the memory-intensive workloads. However, the proposed LUT technique, which consumes less memory bandwidth, resulted in lower overheads, i.e., 8.42% and 22.99%, respectively. The naïve LUT approach, even with a $2^{11} \times 2$-byte smaller LUT for higher throughput but less accuracy, still incurred much higher overhead of 9.72% and 29.35%, respectively. As a result, the proposed LUT method is more tolerant of the memory-intensive workload environment since it generates a lower memory interference overhead.

D. Area & Power Overheads

Compared to the baseline architecture [13], we only modified the PIU, introducing the argument registers, integrating a concatenation circuit to recompute an effective address, and adding circuits to support indexed addressing for LUT operations. The modified components contributed an overhead of 4.65% of the total area and 8.61% of the overall power consumption, highlighting the simple design.

V. Conclusion

This paper introduced two register-based addressing modes to an in-DRAM PIM-DMA architecture, enabling PIM register values to reconstruct PIM ISAs at runtime. Our indexed addressing maximized the PIM computational capabilities, thus reducing data transfer, accelerating the PIM unary operations, and enabling more operator fusion. Also, the base-offset addressing mode eliminated the code offloading overhead. Our system-level evaluation showed that we could speed up 1.26x to 1.94x compared to the baseline PIM, with a small area and power consumption overhead, while tolerating the memory-intensive workload environment.

In the future, we would support sparse data processing using the PIM-based indexed addressing mode, which is always challenging for all communities beyond the PIM research community.

Acknowledgment

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (Ministry of Science and ICT, MSIT) (No. 2025-0-00441, Memory-Centric Architecture Using the Reconfigurable PIM Devices). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Seon Wook Kim is the corresponding author.

References

[1] Z. Wayne Xin et al., "A Survey of Large Language Models," March 2023, arXiv preprint arXiv:2303.18223.

[2] Y. Kwon et al., "System Architecture and Software Stack for GDDR6-AiM," in 2022 IEEE Hot Chips, Cupertino, CA, USA, August 2022, pp. 1-25, DOI: 10.1109/HCS55958.2022.9895629.

[3] D. U. Lee et al., "22.3 A 128Gb 8-High 512GB/s HBM2E DRAM with a Pseudo Quarter Bank Structure, Power Dispersion and an Instruction-Based At-Speed PMBIST," in 2020 IEEE ISSCC, San Francisco, CA, USA, February 2020, pp. 334-336, DOI: 10.1109/ISSCC19947.2020.9062977.

[4] S. Lee et al., "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product," in 2021 ACM/IEEE ISCA, Valencia, Spain, June 2021, pp. 43-56, DOI: 10.1109/ISCA52012.2021.00013.

[5] F, Devaux,, "The True Processing in Memory Accelerator," in 2019 IEEE Hot Chips, Cupertino, CA, USA, August 2019, pp. 1-24, DOI: 10.1109/HOTCHIPS.2019.8875680.

[6] W. J. Lee, C. H. Kim, Y. Paik, and S. W. Kim, "Processing-in-memory Instruction Set Architecture using DMA," in IEEE Access, vol. 11, pp. 8622–8632, January 2023, DOI: 10.1109/ACCESS.2023.3238812.

[7] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," October 2017, arXiv preprint arXiv:1710.05941.

[8] Intel, "Bfloat16 – Hardware Numerics Definition," [Online]. Available: https: //software.intel.com/sites/default/files/managed/40/8b/bf16-hardwarenumerics-definition-white-paper.pdf.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Nov. 2018, arXiv preprint arXiv:1810.04805.

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," July 2019, arXiv preprint arXiv:1907.11692.

[11] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," Journal of Machine Learning Research, vol. 21, no. 140, pp. 1–67, October 2020.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," in OpenAI Blog, vol. 1, 2019.

[13] C. H. Kim, W. J. Lee, Y. Paik, K. Kwon, S. Y. Kim, I. Park, and S. W. Kim, "Silent-PIM: Realizing the Processing-in-Memory Computing With Standard Memory Requests," in IEEE TPDS, vol. 33, no. 2, pp. 251–262, February 2022, DOI: 10.1109/TPDS.2021.3065365.

[14] L. Chenyang et al. "Macaw-LLM: Multi-Modal Language Modeling with Image, Audio, Video, and Text Integration," June 2023, arXiv preprint arXiv:2306.09093.

[15] N. Rotstein, D. Bensaid, S. Brody, R. Ganz, and R. Kimmel " FuseCap: Leveraging Large Language Models for Enriched Fused Image Captions, " in WACV, pp. 5689-5700, February 2024.

[16] G. Yunhao et al. "Eyes Closed, Safety On: Protecting Multimodal LLMs via Image-to-Text Transformation," March 2024, arXiv preprint arXiv:2403.09572.

[17] W. Han, H. Cho, D. Kim, and J.-Y. Kim, "SAL-PIM: A Subarray-level Processing-in-Memory Architecture with LUT-based Linear Interpolation for Transformer-based Text Generation," January 2024, arXiv preprint arXiv:2401.17005.

[18] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in MICRO, New York, NY, USA, pp. 100–113, October 2019, DOI:10.1145/3352460.3358260.

[19] Xilinx Inc., "Vivado Design Suite User Guide," [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html.

[20] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," in ICLR, New Orleans, LOU, USA, pp. 353-355, February 2019, DOI:10.18653/v1/W18-5446.

[21] T. Le Scao et al., "BLOOM: A 176B-Parameter Open-Access Multilingual Language Model," June 2023, arXiv preprint arXiv:2211.05100.

[22] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," February 2023, arXiv preprint arXiv:2302.13971.

FedEDA: Federated Learning Framework for Privacy-Preserving Machine Learning in EDA

Joonseok Kim*, Donggyu Kim*, Seonghyeon Park, and Seokhyeong Kang
Pohang University of Science and Technology, Pohang, Republic of Korea
{adrs77, gyu511, seonghyeon98, shkang}@postech.ac.kr

***Abstract*—In advanced nodes, the optimization of Power, Performance, and Area (PPA) is becoming increasingly complex, requiring significant resources and time for circuit design and optimization using Electronic Design Automation (EDA). As a key approach to overcome these challenges, Machine Learning (ML) techniques have been widely studied in the field of EDA. However, security concerns around Intellectual Property (IP) limit access to real-world circuit data, making it difficult to gather sufficient data for training ML models. This lack of available circuit benchmarks restricts progress in ML research. In this study, we propose *FedEDA*, which, to the best of our knowledge, is the first Federated Learning (FL) aggregation algorithm specifically designed for EDA. *FedEDA* addresses concerns about IP security by exchanging model weights among FL participants instead of sharing raw data. Furthermore, *FedEDA* leverages Rent's Rule and circuit size to capture the hierarchical structure of circuits, mitigating issues related to data imbalance among participants and improving the quality of weight aggregation on EDA data. We demonstrate the applicability of *FedEDA* across various EDA tasks, including routability, parasitic RC, and wirelength prediction. *FedEDA* outperforms existing FL algorithms in EDA tasks, demonstrating superior performance.**

I. Introduction

Recent technological advancements have significantly increased transistor integration density [1]. As chip designs become more complex, the costs and efforts in Integrated Circuits (IC) development also rise, with resource shortages for optimizing servers and license scheduling presenting significant bottlenecks in this process [2]. Machine Learning (ML) has emerged as an effective solution to these problems, significantly reducing the time required for IC design [3]–[5].

In ML, model performance typically improves with larger and more reliable datasets [6]–[8]. However, unlike famous application fields such as computer vision and natural language processing, which have large accessible datasets [9]–[11], ML in Electronic Design Automation (EDA) suffers from data scarcity. This is due to a lack of data sharing in the EDA community to protect one's intellectual property (IP) from illegal duplication [12]–[16]. Centralized data collection for ML training can lead to IP infringement, as malicious users could access the database and extract patented circuit information [17]–[19]. This security issue limits data diversity in ML for EDA (ML-EDA).

To address data privacy concerns, Federated Learning (FL) [20] was developed, focusing on data localization and remote model training. FL is a distributed computing approach where independently trained models are aggregated without direct access to private data. In this setup, clients send their locally trained model weights to a central server, which aggregates these weights to update the global ML model. This method ensures secure ML model training without risking data leakage, as private data remains strictly within each client's local environment. However, since local clients hold data that differs in both quantity and label distribution, the issue of data imbalance arises, leading to significant performance degradation of the global model when aggregating the locally trained model weights.

P. Jingyu *et al.* introduced FLNet [21], which demonstrates the application of FL in the routability prediction task. FLNet alleviates the effects of data imbalance by reducing the number of model parameters, thus mitigating performance degradation in FL aggregation. However, it is a **model-based** approach that addresses data imbalance issues simply by modifying the model. Additionally, it does not leverage the unique characteristics of EDA data. This approach cannot integrate state-of-the-art models designed to improve prediction accuracy in the EDA domain, limiting the broad application of FL in the field.

In this work, we propose a novel model aggregation algorithm called *FedEDA*, specifically designed for EDA tasks. *FedEDA* incorporates the unique complexity and hierarchical characteristics of circuit data into the algorithm. This approach addresses data imbalance issues in distributed ML-EDA environments in the weight aggregation process, which is independent of the conventional model-based approach [21]. To demonstrate the applicability of *FedEDA*, we evaluate its performance across various prediction tasks and widely used ML models in the EDA domain. The contributions of our research can be summarized as follows:

- We present *FedEDA*, which to the best of our knowledge, is the first FL **aggregation algorithm** designed specifically for EDA. *FedEDA* addresses the challenges of data scarcity and imbalance in ML-EDA by considering the unique characteristics of EDA data in its weight aggregation process while ensuring data privacy and security.
- We recognize that circuit heterogeneity is the key factor leading to performance degradation. To capture circuit complexity, we consider Rent's Rule [22] along with circuit size. *FedEDA* systematically incorporates these characteristics into its loss function.
- *FedEDA*'s applicability is demonstrated across a wide range of EDA prediction tasks, including routability, parasitic RC, and wirelength prediction, using various models.

*These authors equally contributed to this work. This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2023-00222085, Development of memory module and memory compiler for non-volatile PIM) and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2019R1A5A1027055).

It outperforms conventional model merging algorithms like *FedAvg* [20], *FedProx* [23] and achieves up to an 8.74% improvement in prediction accuracy compared to FLNet in routability prediction.

II. Preliminaries

A. Merging Independently Trained Models

Model merging is an ML technique that combines the experiences of independent training runs without additional training or fine-tuning. The primary concept behind this strategy is to integrate various models, each trained on local data, to enhance performance by utilizing the collective expertise of multiple models. A key benefit of model merging is its contribution to data privacy, as it allows independently trained models to be combined without the need for centralized data storage. A notable example of this approach is FL [20]. A well-known algorithm that exemplifies this approach is *FedAvg* [20], an effective FL algorithm that aggregates locally trained models at each iteration. This process can be summarized as follows:

$$w_{t,j}^k \leftarrow w_{t,j-1}^k - \eta \sum_{b=1}^{B} \nabla_w F(y_b, f_{w_{t,j-1}^k}(x_b)), \tag{1}$$

where B is batch size, t is global round, j is local round, k is client, and w is model weight. At every global round t, for each client $k \in \{1, ..., K\}$, the local checkpoints $w_{t-1,0}^k$ are initialized to the aggregated result w_t. Every client updates their local checkpoints for each local step $j \in \{1, ..., E\}$ by mini-batch gradient descent for a learning rate of η. Locally trained weights w_t^k are transmitted to the host, which is then merged to form the global model, as follows:

$$w_{t+1} = \frac{1}{K} \sum_{k=1}^{K} w_{t,E}^k. \tag{2}$$

Global training rounds are repeated by a specified global epoch T defined by the host. Upon convergence, the globally trained weights are broadcast to each client for local inference.

B. Related work

FedAvg is one of the simplest and most effective aggregation algorithms in FL [24] when data is Independent and Identically Distributed (IID) among clients. However, in most real-world scenarios, data tends to be heterogeneously distributed, or non-IID and training models on such data using *FedAvg* can lead to biased or sub-optimal models [25]. As a result, the local optima for each client diverges from the global optimum, a phenomenon known as *drift* in local updates. Naive aggregation of models with *FedAvg* does not always ensure that the averaged model closely approximates the global optimum which leads to performance degradation.

Alternative aggregation algorithms such as *FedProx* [23], *FedNova* [26] and *SCAFFOLD* [27] have been studied to address the challenge posed by non-IID data. *FedProx* [23] incorporates an L2 regularization term into the loss function, directly limiting the drift between the local optima and global optimum. *FedNova* [26] assumes the varying number of local steps (τ_k) among participants, thus normalizing the local updates based on the number of preceding local updates before updating the global model. The *SCAFFOLD* [27] algorithm estimates local training divergence by subtracting the global model update direction from the local models, integrating this drift to refine local updates. However, these algorithms were developed primarily for conventional ML data, such as images and text.

P. Jingyu *et al.* designed a customized ML model called FLNet [21] to apply FL to routability prediction, an ML application in EDA. FLNet addresses the data heterogeneity issue in FL by removing batch normalization operators, proposing a model-based solution. However, this **model-based** approach cannot be applied to various ML-EDA models developed for performance enhancement, thus limiting the general application of FL in the field of EDA.

C. Rent's Rule

Rent's Rule [22], as one method to represent the characteristics of circuits, describes a power-law relationship between the number of input/output (I/O) terminals, denoted as T, and the number of gates, N, in a logic block. This relationship is expressed by the following equation:

$$T = kN^p, \tag{3}$$

where k and p are Rent's constant and exponent, respectively. These parameters are derived empirically and p is a measure of the interconnection complexity of the circuit. Its value ranges between 0 and 1; higher values mean higher complexity. Understanding the relationship between the number of gates and terminals allows designers to manage circuit complexity and connectivity more effectively, thereby enhancing overall design efficiency and performance [28].

In the context of FL for ML-EDA, leveraging Rent's Rule enables a systematic way to capture circuit complexity and integrate it into model aggregation algorithms like *FedEDA*. This helps address challenges arising from data heterogeneity, as circuit complexity significantly influences the distribution and behavior of EDA datasets.

III. Proposed Method

A. FedEDA for EDA based Federated Learning

We propose *FedEDA*, a robust model merging algorithm that accounts for the unique feature distribution of EDA data in FL. The key parameters used in *FedEDA* are described in Table I. Circuit designs typically exhibit a hierarchical structure [29], [30], where multiple small circuit modules are interconnected to form the top module, as depicted in Fig. 1 (a). Larger designs include more submodules and connections, expanding the diversity of new characteristics in the EDA feature space. For example, in Fig. 1 (b), circuit A offers a limited feature space, whereas a larger circuit C provides a significantly broader one. Since both A and C are hierarchically structured, the majority of the feature space introduced by A can also be represented by C, as illustrated by the red region in Fig. 1 (b). If local models are independently trained on A, B, and

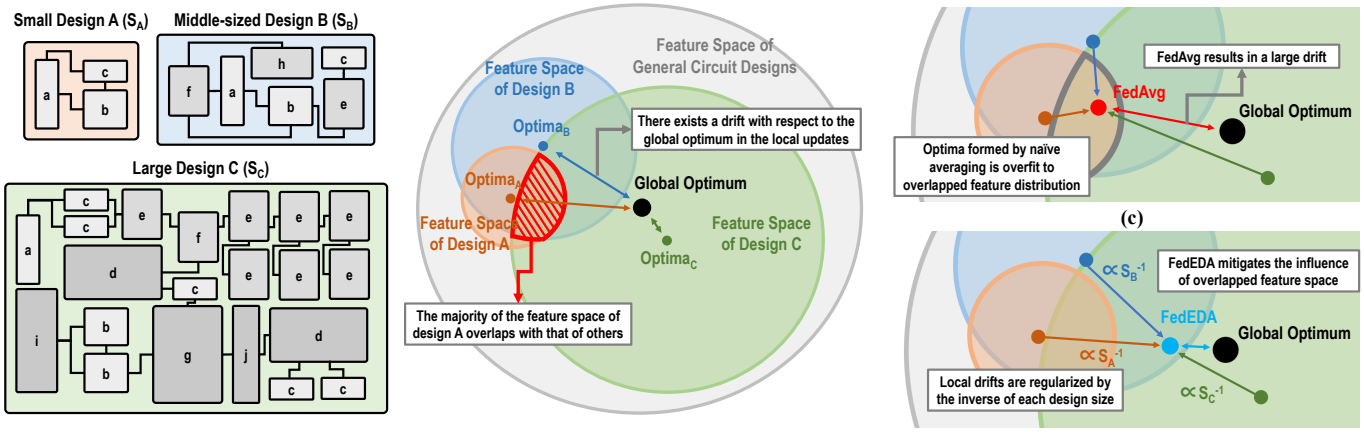


Fig. 1. Motivation for *FedEDA*. (a) shows the hierarchical characteristics of circuit designs. Larger circuits tend to have more diverse modules and number of connections. For each design i, S_i represents the circuit design size. (b) represents the feature space for designs A, B, and C. The red area on the middle left represents features shared among these three designs. (c) The averaged updates generated by *FedAvg* [20] has a considerable drift with respect to the global optima. (d) *FedEDA* alleviates this drift by taking the hierarchical information in EDA data, into account for each local update.

TABLE I
DESCRIPTION OF KEY PARAMETERS USED IN *FedEDA*

Notation	Description
p	Rent's exponent
σ	Standard deviation of p in submodule
μ	Mean of p in submodule
CM	Circuit metadata (p, σ and circuit size)
α	Inverse min-max normalization value
K	Set of clients ($k \in K$)
D^k	Local database of client k
T	Number of global iterations for FL
γ	Relative weight of importance of α
F	Original local loss function [20]
f	ML model
B	Batch size
(x_b, y_b)	Data sample from batch B

C, and averaged *FedAvg* [20], the resulting global model will be biased towards the feature space of A, since it contains the majority of the shared features, as shown in Fig. 1 (c). This significant *drift* between the averaged updates and the global optimum can lead to performance degradation when evaluated on other circuit designs. Therefore, we identify circuit size as a major factor of data heterogeneity for EDA data and incorporate this characteristic in formulating *FedEDA*.

However, circuit size alone is insufficient to capture characteristics of circuit data distribution since larger designs often consist of repetitive structures of identical submodules. We address this issue using Rent's Rule [22] by calculating Rent's parameter p for each submodule of the netlist and then computing the standard deviation σ as follows:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - \mu)^2}. \tag{4}$$

The value of σ represents the diversity of submodules, while p indicates the complexity of the circuit. A small σ implies that the circuit contains submodules with similar characteristics, resulting in a repetitive data distribution. Similarly, a small p indicates a simple circuit with a less diverse data distribution. From a distributed environment perspective, circuits with small σ and p contribute to a biased feature space in the overall circuit dataset. Therefore, in *FedEDA*, these factors are integrated into the local loss function, applying a penalty to circuits with smaller sizes (number of cells), low σ, and low p. The factors σ, p, and circuit size are collectively referred to as circuit metadata (*CM*) and represent the circuit complexity and hierarchical characteristics. CM is unique to each circuit dataset and rescaled using min-max normalization to stabilize the drift-based regularization term. Clients transmit their *CM* to the host and the host calculates the maximum and minimum *CM* values to get CM_{max} and CM_{min}, which are returned to every client. Clients calculate the inverse min-max normalization values α_{size}, α_σ, and α_p for each circuit data using the following equation:

$$\alpha = \frac{\mathrm{CM_{max}} - \mathrm{CM_{min}}}{\mathrm{CM} - \mathrm{CM_{min}}}. \tag{5}$$

As a weighted sum, α_{size}, α_σ, and α_p are combined with the L2 regularization term $\|w_t - w^k_{t,j-1}\|^2$, which represents the *drift* of the converged model and global optima [23]. The regularization term is integrated into the modified local loss function L^k_B as follows:

$$L^k_B = \sum_{b=1}^{B} F(y_b, f_{w^k_{t,j-1}}(x_b)) + (\alpha_{size}\gamma_{size} + \alpha_\sigma\gamma_\sigma + \alpha_p\gamma_p)\|w_t - w^k_{t,j-1}\|^2. \tag{6}$$

This approach ensures that if the local weight $w^k_{t,j-1}$ deviates significantly from the global weight w_t, the loss function increases, thus reducing the *drift* of each local update. As shown in Fig. 1 (d), the regularization of local updates per design is scaled with the inverse of its α values. Consequently, in circuit datasets where most of the features overlap with one another, the influence of local weight updates is reduced, so that the model merged by the *FedEDA* algorithm has a smaller *drift* from the global optimum.

Algorithm 1: $FedEDA$ (Proposed Method)

1 **Input:** $w^k_{t-1,0}, D^k, \alpha_{size}, \alpha_p, \alpha_\sigma, \gamma_{size}, \gamma_p, \gamma_\sigma, f$
2 **Phase 1: Initialization**
3 **Client executes:**
4 **for** *client* $k \in K$ **do**
5 return CM^k to host
6 **end**
7 **Server executes:**
8 $CM_{\max} \leftarrow \max\{CM^1, \ldots, CM^k\}$
9 $CM_{\min} \leftarrow \min\{CM^1, \ldots, CM^k\}$
10 Distribute w_0, ML model, CM_{max}, CM_{min} to $\forall k \in K$
11 **Client executes:**
12 **for** *client* $k \in K$ **do**
13 Extract D^k from Layouts
14 Initialize $w^k_{t-1,0} \leftarrow w_t$
15 **end**
16 **Phase 2: Federated learning**
17 **for** $t \leftarrow 1$ **to** T **do**
18 **Client executes:**
19 **for** *client* $k \in K$ **do**
20 **for** $j \leftarrow 0$ **to** E **do**
21 $w^k_{t,j+1} \leftarrow w^k_{t,j} - \eta \sum_{b=1}^{B} \nabla_w L^k_B$
22 **end**
23 return $w^k_{t,E}$ to host
24 **end**
25 **Server executes:**
26 $w_{t+1} \leftarrow \frac{1}{k} \sum_{k=1}^{K} w^k_{t,E}$
27 Distribute w_{t+1} to $\forall k \in K$
28 **end**
29 **Phase 3: Application**
30 **Server executes:**
31 Distribute w_{T+1} $\forall k \in K$
32 **Client executes:**
33 Result $\leftarrow$ ML model($dataset, w_{T+1}$)

B. Federated Learning Framework

The overall framework of the FL for ML-EDA is described in Algorithm 1. It involves a single administering host and multiple clients with different IC designs. This framework consists of three stages: (1) the Initialization phase, (2) the FL phase, and (3) the Application phase.

*1) **Initialization phase (Algorithm 1, lines 1-15)**:* Initially, clients provide the FL host with CM (circuit size, p, σ) (line 5). Upon receiving metadata from each participant, the host returns processed global metadata (lines 8-9), along with design feature extractors, model architectures, and initialized weights (line 10). Each client then extracts relevant features to train their local ML models (line 13). Models for each client are initialized with the same weights to ensure consistency across participants and promote stable convergence.

*2) **FL phase (Algorithm 1, lines 16-28)**:* After initialization, the local model is trained on the client's database with the local training epoch E and learning rate η. Initial weights w_0 are updated to w^k_1 using *FedEDA* (line 21). Afterward, clients send their locally trained weights to the host for weight aggregation (line 23), where they are merged into the central server model (line 26). These weights are then broadcast to each client, initiating a new global round with a shared checkpoint across all participants. This global round of training is iterated for global epochs T.

*3) **Application phase (Algorithm 1, lines 29-33)**:* In the last phase, the final weights of the converged model are distributed to each client (line 31), where they are applied to their local ML algorithm for inference (line 33). This allows clients to predict EDA outcomes early in the physical design process, helping them assess the design validity.

Security is maintained throughout the process since no raw circuit data is exchanged. By training ML models within the FL framework, clients can enhance their models' performance by leveraging the EDA experiences of others while ensuring their own data's privacy. Furthermore, *FedEDA* can be integrated with complementary approaches, such as FLNet [21] or personalization techniques [31], broadening the applicability of FL in ML-EDA and enabling more versatile solutions.

IV. Experimental Results

A. Experimental Setup

We utilized 21 *OpenCores* [32] circuit designs as shown in Table II to train and test our ML-EDA model. We employed two commercial tools, *Design Compiler v18.06* [33] for synthesis and *Innovus v21.16* [34] for P&R. We generated 160 layouts for each circuit by sweeping the cell height, utilization, top routing layer, and P&R clock period of the layout for our experiment. Designs were processed using the *ASAP 7nm* [35] Process Design Kit (PDK) with 6-track cells and 7.5-track cells. The distributed training environment was simulated on 2.8 GHz AMD EPYC 7402P CPU with 256GB RAM, and experiments were conducted on an NVIDIA Titan RTX GPU.

B. ML Tasks for EDA

We evaluated our framework on various ML-based EDA prediction tasks, including routability, parasitic RC, and Wirelength (WL) prediction, using different ML models such as CNNs, MLPs, and GNNs. The CNN and MLP models were implemented using Python's PyTorch library, while the GNN models were implemented using the deep graph library [36]. The overall prediction task configuration and model architecture for evaluating our framework is shown in Fig. 2.

TABLE II
Benchmark information

		6 Track		7.5 Track	
Design	**# Ports**	**# Cells**	**# Nets**	**# Cells**	**# Nets**
aes_cipher_top	391	9,205	9,464	9,376	9,635
des3	298	59,431	59,665	58,192	58,426
eth_top	211	33,021	33,277	33,658	33,914
fpu	110	16,519	18,982	18,928	21,193
jpeg_encoder	47	60,843	70,718	60,811	70,685
keccak	585	26,645	26,716	26,107	26,178
ldpc_decoder_802_3an	4,100	42,509	44,560	44,452	46,503
mc_top	267	4,070	4,208	4,073	4,211
nova	174	134,355	136,569	136,941	139,203
mpeg2_top	230	384,168	391,765	387,770	395,421
pci_bridge32	369	11,173	11,382	11,012	11,221
point_scalar_mult	932	39,422	39,963	40,460	41,002
sasc_top	28	383	399	382	398
spi_top	116	1,695	1,744	1,890	1,945
tate_pairing	1,943	167,119	167,897	169,570	170,348
tv80s	46	4,432	4,451	4,452	4,485
usbf_top	249	7,683	8,050	8,058	8,425
usb_phy	33	358	376	341	359
vga_enh_top	198	52,585	52,687	59,177	59,287
wb_conmax_top	2,546	17,744	18,874	18,249	19,379
wb_size_bridge	183	334	425	314	405

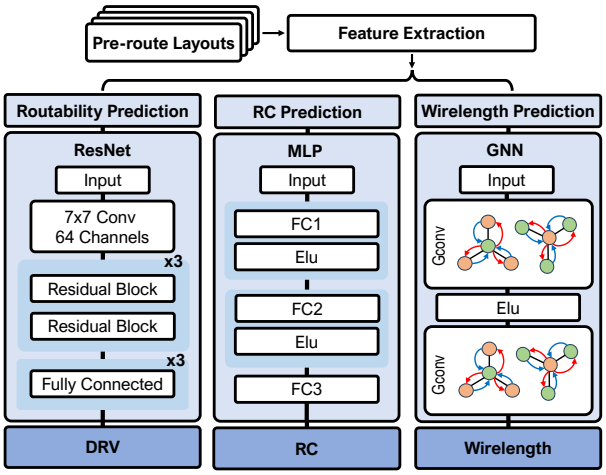


Fig. 2. ML-EDA prediction tasks to evaluate our FL framework.

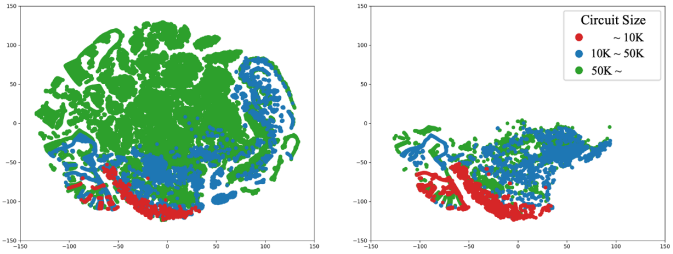
We validated the applicability of our framework to ML-EDA by incorporating it into tile-level routability prediction methods [37]. Instead of directly predicting delay, we applied our approach to RC prediction, which is used for timing prediction [38], to handle extensive output slew data. Furthermore, due to the straightforward conversion of netlists into graphs, we employed GNNs, which are extensively used in the EDA domain [39], to construct the FL framework. Our EDA-based FL framework can be seamlessly applied to various state-of-the-art ML-EDA models, as demonstrated through its application to several commonly used ML models in EDA.

C. Data distribution of EDA Data

We analyzed the distribution of dataset feature space used in ML-EDA through t-SNE visualization. Here, Rent's parameters were calculated by performing min-cut partitioning on each submodule of the gate-level netlist using ML Part [40].

Fig. 3 shows the distribution of data according to circuit size, with small circuits shown in red, medium-sized circuits in blue, and large circuits in green. In Fig. 3 (a), when the standard deviation of the Rent's parameter p of the submodules is 0.05 or higher, indicating that the circuit is composed of various submodules, the feature data distribution illustrates a hierarchical structure from the lower left to the upper area according to circuit size. Conversely, Fig. 3 (b) illustrates that when the standard deviation of the Rent's parameter p is below 0.05, leading to redundant and similar submodule circuits, the feature data distribution remains clustered in the lower-left corner, regardless of increasing circuit size. Here, the standard deviation of 0.05 is the mean of the standard deviations of p for all benches used in the experiment. This highlights the hierarchical nature of EDA data, shaped by design size and complexity. By using *FedEDA* to increase the model's aggregation quality for large and complex designs, it is possible to learn from various data distributions, as shown in the upper region of Fig. 3 (a), thereby improving overall model performance.

This raises the question of whether identifying circuit complexity is meaningful for EDA datasets and how it compares to commonly used ML datasets. To explore this, we compared



(a) (b)

Fig. 3. T-SNE visualizations of DRV prediction feature data based on circuit size. Using a standard deviation threshold of 0.05 for the Rent's parameter p of the submodules, (a) represents designs with a higher standard deviation, while (b) represents designs with a lower standard deviation.

TABLE III
COMPARISON OF ENTROPY VALUES FOR DATASETS ACROSS DIFFERENT ML DOMAINS

Dataset	MNIST [41]	DBpedia [42]	Circuit data
Entropy	3.320	3.807	2.740

the characteristics of commonly used ML datasets with circuit data by analyzing entropy values across three datasets: the MNIST [41] vision dataset, the DBpedia [42] language dataset, and a circuit dataset used for DRV prediction. Higher entropy values indicate a greater degree of randomness in the data distribution [43]. As shown in Table III, the circuit dataset exhibits the lowest entropy value. This suggests a more concentrated distribution with less randomness, indicating that certain features are overly represented. By incorporating key factors contributing to this concentration in *FedEDA*, such as circuit size and complexity, we enhance its evaluation performance.

D. Performance Evaluation

We evaluate the effectiveness of *FedEDA* by comparing its performance against other FL methods on both classification and regression tasks. For the DRV prediction task (classification), we used accuracy and F1 score as evaluation metrics, while for the regression tasks of predicting parasitic RC and WL, we evaluated performance using the R2 score. The training dataset consisted of 18 circuits with a total of 2,280 layouts, while the test dataset included 3 circuits with a total of 480 layouts that were not used in training. For each γ value described in Eq. (6), we optimized the ideal values from the restricted candidate set $\{0.001, 0.01, 0.1, 1\}$. We tested and compared model performance with other weight aggregation techniques, *FedAvg* [20], and *FedProx* [23].

Under the IID condition, the training data was randomly shuffled and distributed in equal quantities among the clients. We adopted three different cases to emulate data heterogeneous environments. For Quantity Skew (QS), the training data was shuffled and randomly allocated to each client in unequal quantities. In the case of Label Skew (LS), the data was distributed in equal quantities to each client with no overlapping circuit designs among them. Lastly, for Label and Quantity Skew (LS+QS), the data was randomly distributed in unequal quantities with no overlapping circuit designs among the clients.

TABLE IV
COMPARISON OF PREDICTION PERFORMANCE ON VARIOUS TRAINING CONFIGURATIONS AND DATA DISTRIBUTIONS: INDEPENDENT AND IDENTICALLY DISTRIBUTED (IID), LABEL SKEW (LS), QUANTITY SKEW (QS), AND LABEL SKEW + QUANTITY SKEW (LS+QS).

Learning Methods	Number of Clients	DRV								RC				WL			
		Accuracy				F1 Score				R2 Score				R2 Score			
		IID	LS	QS	LS+QS	IID	LS	QS	LS+QS	IID	LS	QS	LS+QS	IID	LS	QS	LS+QS
Central Learning	-	0.748				0.585				0.751				0.888			
Local Learning (Averaged)	-	0.371				0.288				0.498				0.585			
FedAvg [20]	2	0.737	0.655	0.717	0.674	0.581	0.514	0.574	0.498	0.716	0.699	0.751	0.637	**0.886**	0.724	0.869	0.653
	3	0.735	0.684	0.709	0.688	**0.620**	0.537	0.586	0.582	0.692	0.677	**0.749**	0.257	0.867	0.797	0.875	0.700
	6	0.731	0.630	0.726	0.690	0.533	0.500	0.539	0.464	0.765	0.764	**0.795**	0.388	0.881	0.791	0.887	0.527
	AVG.	0.734	0.656	0.717	0.684	0.578	0.517	0.566	0.515	0.724	0.713	**0.765**	0.427	0.878	0.771	0.877	0.627
FedProx [23]	2	0.733	0.703	0.715	0.711	0.568	0.443	0.496	0.559	**0.763**	0.731	**0.753**	0.725	0.884	0.811	0.886	0.802
	3	0.742	0.699	0.713	0.688	0.553	0.515	**0.592**	0.585	0.740	0.727	0.738	0.726	0.854	0.819	0.877	0.823
	6	0.736	0.675	0.682	0.702	**0.581**	**0.554**	**0.572**	0.562	**0.775**	**0.768**	0.753	0.713	0.875	0.820	0.883	0.818
	AVG.	0.737	0.692	0.703	0.703	0.567	0.504	0.533	0.569	**0.756**	0.742	0.745	0.721	0.871	0.817	0.882	0.814
FedEDA (Ours)	2	**0.747**	**0.736**	**0.742**	**0.733**	**0.585**	**0.545**	**0.581**	**0.564**	0.743	**0.744**	0.748	**0.741**	**0.886**	**0.833**	**0.887**	**0.824**
	3	**0.748**	**0.733**	**0.739**	**0.731**	0.582	**0.554**	0.574	**0.586**	**0.742**	**0.743**	**0.749**	**0.746**	**0.886**	**0.857**	**0.882**	**0.841**
	6	**0.748**	**0.729**	**0.741**	**0.728**	0.578	0.551	**0.572**	**0.571**	0.738	0.752	0.751	**0.747**	**0.887**	**0.845**	**0.886**	**0.835**
	AVG.	**0.748**	**0.733**	**0.741**	**0.731**	**0.582**	**0.550**	**0.576**	**0.574**	0.741	**0.746**	0.749	**0.745**	**0.886**	**0.845**	**0.885**	**0.833**

The best results are highlighted in **bold**.

To equate the total exposure of the dataset across the three schemes—Federated Learning (FL), Centralized Learning (CL), and Local Training (LT)—the product of the global epochs and local epochs in FL was set to be equal to the number of epochs in CL and LT per client k, as follows:

$$Epoch_{CL} = Epoch_{LT}^{k} = GlobalEpoch_{FL} \times LocalEpoch_{FL}. \quad (7)$$

As shown in Table IV, Local Learning, where each client trains on their own data, results in a 41.39% decrease in average performance metrics compared to the centrally trained model (Central Learning). By applying *FedEDA* in FL under the same data distribution conditions as Local Learning, the average performance metrics decrease by just 1.98%, achieving results comparable to centralized learning. Moreover, *FedEDA* provides a privacy-preserving environment with minimal impact on model performance. Additionally, compared to previous FL model merging algorithms, *FedEDA* achieves the best performance in 12 out of 12 scenarios for DRV prediction (Accuracy), 6 out of 12 for RC prediction (R2), and 12 out of 12 for WL prediction (R2), across various data distributions and client configurations. Notably, in the LS+QS distribution, which is most similar to real-world conditions, the maximum improvement in average (AVG.) F1 score and R2 score for DRV, RC, and WL prediction are 11.5%, 74.5%, and 33.5%, respectively, compared to conventional model merging algorithms [20], [23]. The γ values that yielded the best performance were 0.1 for γ_{size}, and 0.001 for both γ_σ and γ_p, suggesting that circuit size is the primary factor contributing to data heterogeneity in circuit data. These results demonstrate that our proposed model merging algorithm effectively addresses the data heterogeneity challenge in EDA datasets, making it highly effective and broadly applicable in ML-EDA.

E. Routability Prediction Performance Comparison

We applied our proposed *FedEDA* to a recently developed routability prediction model [37] and compared its performance with FLNet [21], since FLNet was originally targeted routability prediction. As shown in Fig. 4, *FedEDA* consistently outperformed FLNet in both accuracy and F1 score, achieving up to an 8.74% improvement in accuracy. These findings emphasize the effectiveness of our model aggregation algorithm in overcoming the limitations of model-specific approaches, which often struggle to adapt to diverse ML models designed for performance optimization. By addressing this challenge, *FedEDA* enhances the flexibility and applicability of FL in the EDA domain. Moreover, this improvement is expected to enable *FedEDA* to support newly developed ML-EDA models, further widening the performance gap compared to model-based approaches.

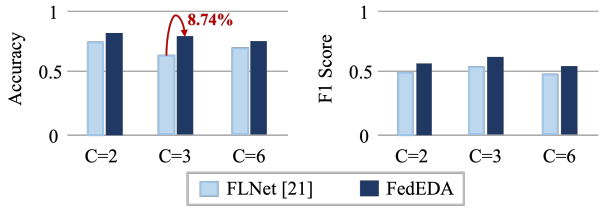


Fig. 4. Comparison of routability prediction performance between FLNet [21] and *FedEDA* in an non-IID (LS+QS) data distribution. C is the client number.

V. CONCLUSION

In conclusion, our study introduces a novel FL aggregation algorithm, *FedEDA*, to address the issues of data heterogeneity and privacy concerns in the ML-EDA domain. To the best of our knowledge, this is the first work proposing an FL **algorithm** specifically for ML-EDA. Unlike previous research focused on model-based solutions [21], *FedEDA* leverages Rent's Rule and circuit size to incorporate the hierarchical characteristics of EDA data into the aggregation process, enhancing the performance of FL models and providing a more general approach. We demonstrate the effectiveness of *FedEDA* by achieving an average improvement of up to 74.5% in R2 score compared to existing aggregation algorithms. [20], [23], and up to an 8.74% improvement compared to the model-based solution, FLNet [21], across a wide range of ML-EDA tasks and various prediction scenarios.

REFERENCES

[1] Lee C, Zhao Y, "Advanced CMOS Devices and Applications," *Electronics*, 13, no. 1: 134, 2024.

[2] A. B. Kahng, "Machine learning for CAD/EDA: The Road Ahead," *IEEE Design & Test*, vol. 40, no. 1, pp. 8–16, 2022.

[3] G. Huang, J. Hu, Y. He *et al.*, "Machine Learning for Electronic Design Automation: A Survey," *ACM TODAES*, vol. 26, no. 5, pp. 1–46, 2021.

[4] Z. Liu, Q. Huang, C. Fang *et al.*, "Improving test chip design efficiency via machine learning," *Proc. ITC*, pp. 1–10, 2019.

[5] Z. Pan, M. Li, J. Yao *et al.*, "Low-cost high-accuracy variation characterization for nanoscale IC technologies via novel learning-based techniques," *Proc. DATE*, pp.797–802, 2018.

[6] A. Halevy, P. Norvig, F. Pereira "The unreasonable effectiveness of data," *IEEE intelligent systems*, vol. 24, no. 2, pp. 8–12, 2009.

[7] C. Sun, A. Shrivastava, S. Abhinav *et al.*, "Revisiting unreasonable effectiveness of data in deep learning era," *Proc. ICCV*, pp. 843–852, 2017.

[8] J. Dean, G. Corrado, C. Rajat *et al.*, "Large scale distributed deep networks," *Proc. NeurIPS*, vol. 25, 2012.

[9] J. Deng, W. Dong, R. Socher *et al.*, "Imagenet: A large-scale hierarchical image database," *Proc. CVPR*, pp. 248–255, 2009.

[10] T. Lin, M. Maire, S. Belongie *et al.*, "Microsoft coco: Common objects in context," *Proc. ECCV*, pp. 740–755, 2014.

[11] J. Pennington, R. Socher, C. Manning "Glove: Global vectors for word representation," *Proc. EMNLP*, pp. 1532–1543, 2014.

[12] X. Jiang, Y. Zhao, Y. Lin *et al.*, "CircuitNet 2.0: An Advanced Dataset for Promoting Machine Learning Innovations in Realistic Chip Design Environment," *Proc. ICLR*, 2023.

[13] V. Hamolia, V. Melnyk, "A survey of machine learning methods and applications in electronic design automation," *Porc. ACIT*, pp.757–760, 2021.

[14] Z. Chai, Y. Zhao, Y. Lin *et al.*, "Circuitnet: An open-source dataset for machine learning applications in electronic design automation (eda)," *arXiv*, 2022.

[15] A. Chowdhury, B. Tan, R. Karri *et al.*, "Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis," *arXiv*, 2021.

[16] D. Kim, S. Lee, K. Min *et al.*, "Construction of realistic place-and-route benchmarks for machine learning applications," *IEEE TCAD*, 2022.

[17] K. Shamsi, M. Li, K. Plaks *et al.*, "IP protection and supply chain security through logic obfuscation: A systematic overview," *ACM TODAES*, vol. 24, no. 6, pp. 1–36, 2019.

[18] J. Knechtel, S. Patnaik, O. Sinanoglu, "Protect your chip design intellectual property: An overview," *Proc. COINS*, pp. 211–216, 2019.

[19] P. SLPSK, S. Ray, S. Bhunia, "Treehouse: A secure asset management infrastructure for protecting 3dic designs," *IEEE Tran. Comput.*, 2023.

[20] B. McMahan, E. Moore, D. Ramage *et al.*, "Communication-efficient Learning of Deep networks from Decentralized Data," *Proc. AISTATS*, 2017.

[21] Pan, Jingyu, *et al.*, "Towards collaborative intelligence: Routability estimation based on decentralized private data," *Proc. DAC*, 2022.

[22] Landman, Bernard S, Russo *et al.*, "On a pin versus block relationship for partitions of logic graphs," *IEEE Tran. Comput.*, vol. 100, pp. 1469–1479, 1971.

[23] X. Yuan, P. Li, "On convergence of FedProx: Local dissimilarity invariant bounds, non-smoothness and beyond," *Proc. NeurIPS*, vol. 35, pp. 10752–10765, 2022.

[24] Sannara, EK, Portes *et al.*, "A federated learning aggregation algorithm for pervasive computing: Evaluation and comparison," *Proc. PerCom*, pp. 1–10, 2021.

[25] Y. Zhao, M. Li, L. Lai *et al.*, "Federated Learning with Non-iid Data," *arXiv*, 2018.

[26] J. Wang, Q. Liu, H. Liang *et al.*, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Proc. NeurIPS*, vol. 33, pp. 7611–7623, 2020.

[27] S. Karimireddy, S. Kale, M. Mohri *et al.*, "Scaffold: Stochastic controlled averaging for federated learning," *Proc. ICML*, pp. 5132–5143, 2020.

[28] L. Hagen, A.B. Kahng, F.J. Kurdahi *et al*, "Partitioning algorithms;Circuit synthesis;Recursive estimation;Testing;Very large scale integration;Phase estimation;Binary trees;Design automation;Computer science;Marine vehicles," *IEEE TCAD*, vol. 13, pp. 27-37, 1994.

[29] C. Niessen, "Hierarchical design methodologies and tools for VLSI chips," *Proc. IEEE*, vol. 71, no. 1, pp. 66-75, 1983.

[30] VanCleemput, W.M., "Hierarchical design for VLSI: Problems and advantages," , 1979.

[31] Agarwal, M., Yurochkin, M., *et al.*, "Personalization in federated learning," *Federated Learning*, 2022.

[32] OpenCores, *Open Source IP-Cores*, http://www.opencores.org.

[33] Synopsys, *Design Compiler User Guide*, http://www.synopsys.com.

[34] Cadence, *Innovus User Guide*, https://www.cadence.com.

[35] L.T. Clark, V. Vashishtha, L. Shifren *et al.*, "ASAP7: A 7-nm FinFET Predictive Process Design Kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[36] Wang, M., Zheng, D., Ye, Z. *et al*., "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv*, 2019.

[37] S. Park, D. Kim, S. Kwon *et al*, "Routability Prediction and Optimization Using Explainable AI," in *Proc. ICCAD.*, pp. 1–8, 2023.

[38] Chhabria, Vidya A, Jiang *et al.*, "A machine learning approach to improving timing consistency between global route and detailed route," in *ACM TODAES*, vol. 29, pp. 1–25, 2023.

[39] Lopera, Daniela Sánchez, Servadei *et al.*, "A survey of graph neural networks for electronic design automation," in *Proc. MLCAD*, pp. 1–6, 2021.

[40] Caldwell, A. B. Kahng, I. L. Markov, "Improved algorithms for hypergraph bipartitioning," in *Proc. ASP-DAC*, pp. 661–666, 2000.

[41] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[42] S. Auer, C. Bizer, G. Kobilarov *et al.*, "DBpedia: A nucleus for a web of open data," *International Semantic Web Conference*, Springer, Berlin, Heidelberg, pp. 722–735, 2007.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning (Adaptive Computation and Machine Learning Series)*, vol. 1, 1st ed. Cambridge, MA, USA: MIT Press, 2017.

AARC: Automated Affinity-aware Resource Configuration for Serverless Workflows

Lingxiao Jin[1*], Zinuo Cai[1*], Zebin Chen[1], Hongyu Zhao[1], Ruhui Ma[1]
[1]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
{jinlingxiao1122, kingczn1314, czb453874483, sjtu-zhy, ruhuima}@sjtu.edu.cn

***Abstract*—Serverless computing is increasingly adopted for its ability to manage complex, event-driven workloads without the need for infrastructure provisioning. However, traditional resource allocation in serverless platforms couples CPU and memory, which may not be optimal for all functions. Existing decoupling approaches, while offering some flexibility, are not designed to handle the vast configuration space and complexity of serverless workflows. In this paper, we propose `AARC`, an innovative, automated framework that decouples CPU and memory resources to provide more flexible and efficient provisioning for serverless workloads. `AARC` is composed of two key components: Graph-Centric Scheduler, which identifies critical paths in workflows, and Priority Configurator, which applies priority scheduling techniques to optimize resource allocation. Our experimental evaluation demonstrates that `AARC` achieves substantial improvements over state-of-the-art methods, with total search time reductions of 85.8% and 89.6%, and cost savings of 49.6% and 61.7%, respectively, while maintaining SLO compliance.**

***Index Terms*—serverless computing, resource configuration, automatization.**

I. Introduction

Serverless computing [1], [2] is emerging as a new paradigm in cloud computing, offering distinct advantages over traditional models like Infrastructure-as-a-Service [3], [4] and Platform-as-a-Service [5], [6]. Unlike these earlier services, serverless computing, or Function-as-a-Service (FaaS), breaks down applications into smaller, more manageable functions. This approach significantly reduces the maintenance burden on developers and enables service providers to maximize the use of underlying hardware resources. Complex applications often require more than one function, which is where serverless workflows come into play [7]. Visualized as directed acyclic graphs (DAGs), serverless workflows can scale based on demand, providing both operational flexibility and cost-effectiveness. Each function within the workflow operates independently, allowing for efficient resource allocation and management.

Despite its reputation for flexible and granular resource allocation, FaaS is still predicated on user-defined quotas for resource configuration. We characterize existing resource management strategies within FaaS frameworks into three primary categories. First, memory-centric configurations in AWS Lambda[1], allow developers to set memory quotas, with vCPU and network bandwidth allocated in proportion to the memory allocation. Second, Google Cloud Functions[2] providers offer a selection of predefined resource combinations. Third, there are flexible yet limited configurations, like Alibaba Cloud Function[3], which enforces a memory-to-CPU ratio within a specified range. It is speculated that these constraints aid cloud providers in the efficient management of resources and potentially reduce the need for extensive physical infrastructure.

However, the existing coupled resource allocation mechanism may not be optimal for all serverless functions. Bilal *et al.* [8] highlight the benefits of resource decoupling, which can potentially reduce execution costs by up to 40% when compared to coupled configurations. However, their study is limited to individual functions and does not extend to workflows. We analyze three serverless workflows [9] with decoupled CPU-memory resources to evaluate their performance in §II-A. Our findings indicate that a coupled CPU-memory mechanism can result in resource inefficiency and increased costs due to the varying resource requirements of different workflows. For instance, in the ML Pipeline workflow, a decoupled configuration of 4 vCPUs and 512 MB memory achieves optimal cost by reducing memory usage by 87.5%, substantially decreasing the overall cost compared to the coupled approach.

While decoupling CPU and memory seems a promising strategy for adaptive resource provisioning, it significantly broadens the configuration space, presenting substantial challenges in finding optimal settings. Previous research [8] suggests a Bayesian optimization-based method for resource allocation, yet this method is geared towards individual functions. We extend the method to workflows and evaluate it with a Chatbot application in §II-B. After 100 rounds of sampling, with the algorithm still not converging, we observe a 32.13% reduction in cost, but the total runtime extends to 9.76 hours, indicating a struggle to reach an optimal solution. Moreover, the cost exhibits frequent fluctuations and nearly half of these changes are increases. These challenges have spurred our efforts to develop a more automated, stable, and efficient resource configuration scheme for the decoupling of resources

This work was supported by Shanghai Key Laboratory of Scalable Computing and Systems. Lingxiao Jin and Zinuo Cai equally contributed to this work. (Corresponding author: Ruhui Ma).

[1]https://aws.amazon.com/lambda/pricing/
[2]https://cloud.google.com/functions/pricing-1stgen
[3]https://www.aliyun.com/product/fc/

and workflows.

In this paper, we introduce AARC, an automated framework designed to configure resources with affinity awareness for serverless workflows. Our key insight lies in the decoupling of computation and storage resources, which enhances flexibility in provisioning resources for serverless workloads. AARC offers three distinct advantages over existing methods. Firstly, AARC surpasses memory-centric allocation schemes [9]–[11] by decoupling memory and CPU, thereby increasing resource flexibility and efficiency through a comprehensive exploration of serverless workflows' resource affinities. Secondly, in contrast to peer works utilizing Bayesian optimization for decoupled configurations [8], our method efficiently manages complex workflows and minimizes the sampling iterations required. Lastly, AARC's resource allocation centers on Service Level Objectives (SLOs), which specify end-to-end latency limits, eliminating manual resource allocation by developers and automating the process.

Specifically, AARC integrates Graph-Centric Scheduler and Priority Configurator to optimize serverless workflow costs using a heuristic algorithm. The Graph-Centric Scheduler starts by breaking down the workflow to find the critical path, which the Priority Configurator then schedules with priority techniques. This is followed by Graph-Centric Scheduler setting sub-SLOs for related sub-paths and determining resource configurations for functions within them. Cloud vendors can use AARC to find optimal resource configurations that meet end-to-end SLOs upon receiving workflows from developers. Our experiment illustrates that during searching for the optimal configuration, AARC reduces the total runtime by around 85% compared to SOTA methods. When comparing the optimal configurations, AARC achieves cost savings of around 50% compared to SOTA methods, while still meeting the SLO requirements.

Our contributions are highlighted as follows.

- Firstly, we identify that memory-centric resource configuration schemes on mainstream serverless computing platforms are not a one-size-fits-all approach, and existing methods for decoupled resource configuration don't work for workflows because of the larger search space.
- Secondly, we propose AARC, a framework for decoupled resource configuration in serverless workflows automatically. Leveraging critical path decomposition and priority scheduling, the framework provides a cost-efficient resource allocation strategy with SLO compliance.
- Finally, our experiment demonstrates that AARC reduces total runtime by 85% and achieves 50% cost savings compared to SOTA methods while meeting SLOs.

II. Motivation and Challenge

A. *Motivation*: Decoupling for Flexibility

While memory-centric resource configurations like AWS Lambda simplify resource management, they may not suit all serverless workloads. Bilal *et al.* [8] demonstrate that resource decoupling can reduce execution costs by up to 40% compared

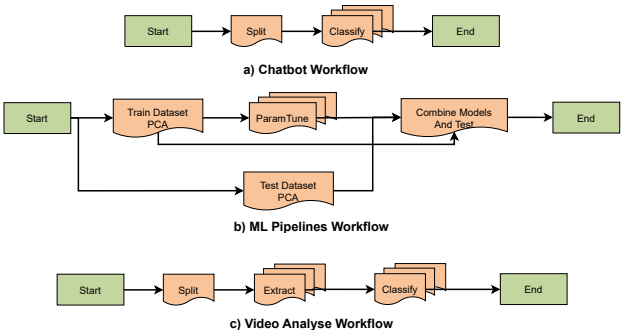


Fig. 1: Architecture of Three Workflows.

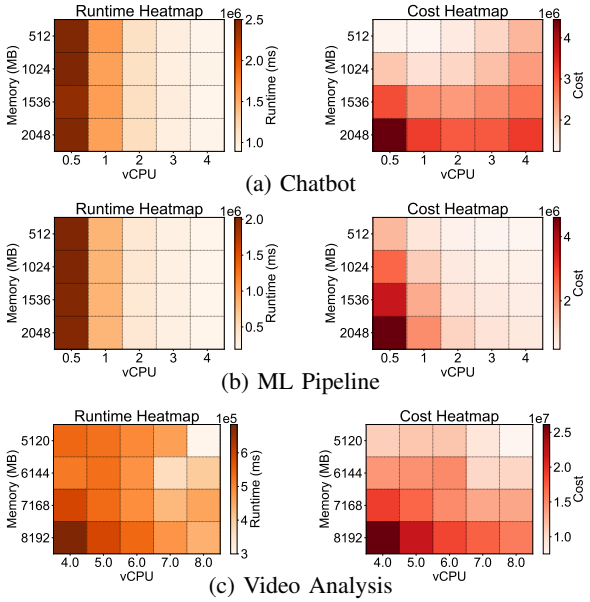


Fig. 2: Runtime and Cost with Decoupled Resources.

to coupled configurations. However, their analysis is restricted to individual functions and does not consider workflows. To explore the impact of decoupled CPU-memory resources for workflows on runtime and cost, we conduct experiments with three different workflows [9], Chatbot, ML Pipeline, and Video Analysis. Fig. 1 illustrates the architecture of these workflows. The Chatbot workflow processes input, trains classifiers in parallel, and uses remote storage for real-time intent detection. The ML Pipeline workflow achieves machine learning by performing dimensionality reduction, model training, and testing. The Video Analysis workflow splits input videos, extracts key frames, and classifies them.

Fig. 2 demonstrates the impact of decoupling CPU and memory on the runtime and cost of the workflows. Fig. 2a and Fig. 2b show that the runtime for Chatbot and ML Pipeline remains unchanged despite memory variations, indicating inefficiency in memory-centric resource allocation for computation-intensive tasks. Notably, in the ML Pipeline workflow, a decoupled configuration of 4 vCPUs and 512 MB memory achieves the lowest cost, reducing memory usage by 87.5% compared to the coupled approach, underscoring the need for decoupled strategies. Additionally, comparing Fig. 2a and Fig. 2c reveals distinct resource affinities across

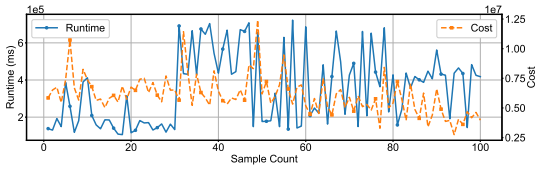


Fig. 3: Bayesian Optimization Search for Chatbot.

workflows. For example, Chatbot minimizes costs with 512 MB memory and 1 vCPU, while Video Analysis achieves cost efficiency with 5120 MB memory and 8 vCPUs. Thus, resource configurations should be tailored to each workflow to minimize costs while meeting SLOs.

B. ***Challenge***: *Larger Configuration Space after Decoupling*

Decoupling CPU and memory improves resource utilization and reduces costs but significantly expands the configuration space, complicating optimal resource provisioning. Profiling-based methods [9], [12], [13] face increased sampling challenges as configurable resources grow. The complexity of serverless applications is further exacerbated by the fact that 46% of applications involve multiple functions [7]. While prior work [8] explores Bayesian optimization for decoupled resource configuration, it focuses on individual functions, overlooking the unique demands of workflows.

We extend the Bayesian Optimization (BO) method from this study to serverless workflows, specifically testing the Chatbot application, to observe changes in runtime and cost with increased sampling. As depicted in Fig. 3, while the cost decreases by 32.13% over 100 sampling rounds without converging, the total runtime extends to 9.76 hours. Additionally, the cost experiences frequent fluctuations, with over half of the changes being increases. The average fluctuation amplitude, calculated as the mean absolute difference between consecutive values, accounts for 18.3% of the mean, which highlights its significant instability. This instability stems from the expansion of the search space when applying decoupling to workflows, which complicates the identification of the optimal solution using Bayesian Optimization. Consequently, current methods struggle to optimize serverless workflows effectively.

III. Design

A. Overview

Fig. 4 depicts the overall architecture of AARC consisting of two main components: Graph-Centric Scheduler and Priority Configurator. Graph-Centric Scheduler decomposes the input workflow and identifies its critical path, along with all the sub-paths linked to it. Priority Configurator configures each function along the critical path and sub-paths.

In our proposed framework, developers submit ❶ their workflow to the cloud platform along with the SLO. First, the Graph-Centric Scheduler profiles the user-defined workflow based on dummy input, calculating the runtime for each function in the workflow. This runtime is then used as the weight of each node, converting ❷ the workflow into a weighted DAG. Next, the Scheduler extracts the critical path and its SLO from the weighted DAG and passes ❸ this information to the

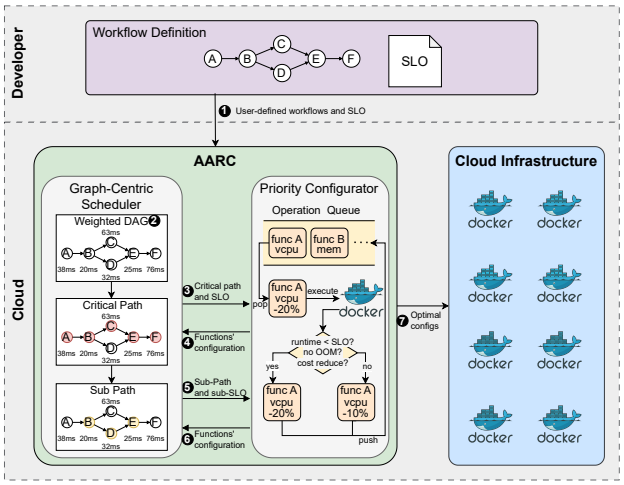


Fig. 4: System Architecture.

TABLE I: Functions in Algorithm 1 and Algorithm 2.

Functions	Description
deallocate(op)	According to the type and step of the given operation op, deprive a portion of the function's resources and return the runtime and cost under this configuration.
allocate(op)	According to type and step of the given operation op, allocate a portion of resources to the function and return a new step and trail of the op.
find_critical_path(G)	Given a weighted DAG, return the critical path of this DAG.
find_detour_subpath(G, critical_path)	Given a DAG and its critical path, return all the sub-paths connected to the critical path.
runtime_sum(path, start, end)	Given the start and the end of a path, calculate the overall duration time between the start and the end.

Priority Configurator. Priority Configurator then incrementally reduces the memory and CPU allocation for each function through priority-based scheduling to determine ❹ the optimal resource configuration that meets the SLO. Based on the optimal configuration of the critical path and the SLO, Graph-Centric Scheduler substitutes functions on the non-critical path onto the critical path to generate ❺ sub-paths and corresponding SLOs. Similarly, Priority Configurator calculates ❻ the optimal resource configuration of the functions in sub-paths, ensuring that the critical path's SLO is not violated. Finally, Graph-Centric Scheduler finalizes ❼ the resource configuration for subsequent container resource allocation, which ensures SLO compliance with optimal cost efficiency.

B. Detailed Design

Graph-Centric Scheduler employs *Overall Scheduling Algorithm* to process the workflow and SLO, performing critical path-based decomposition of the workflow. It then invokes the Priority Configurator to allocate resources for functions along the paths. Priority Configurator uses *Priority Configuration Algorithm* to handle the function paths and SLO, achieving decoupled resource allocation through priority scheduling. The pseudocode for the *Overall Scheduling Algorithm* and

Algorithm 1: Overall Scheduling

Input: function workflow G, end-to-end latency SLO
Output: configurations for each function G_configs

1 **Function** ***schedule***(G, SLO)**:**
2 /* assign base configuration */
3 **foreach** $v_i \in$ G **do**
4 v_i.config ← base_config
5 **end**
6 /* execute to find critical path */
7 execute G
8 L ← ***find_critical_path***(G)
9 G_configs ← {}
10 configs ← ***priority_configuration***(L, SLO)
11 G_configs ← G_configs ∪ configs
12 /* compute configs for subpaths */
13 subpaths ← ***find_detour_subpath***(G, L)
14 **foreach** sp ∈ subpaths **do**
15 SLO′ ← ***runtime_sum***(L, sp.start, sp.end)
16 **foreach** $v_i \in$ sp **do**
17 **if** v_i is scheduled **then**
18 v_i ← ***pop***(sp)
19 SLO′ ← SLO′ − v_i.runtime
20 **end**
21 **end**
22 configs ← ***priority_configuration***(sp, SLO′)
23 G_configs ← G_configs ∪ configs
24 **end**
25 **return** G_configs
26 **End Function**

Algorithm 2: Priority Configuration

Input: function path L, end-to-end latency SLO
Output: configuration for each function configs

1 **Function** ***priority_configuration***(L, SLO)**:**
2 PQ, count ← ***priority_queue***(), 0
3 **foreach** $v_i \in L$ **do**
4 **for** type ∈ [cpu, mem] **do**
5 func, priority ← v_i, ∞
6 step, trial ← 1, FUNC_TRIAL
7 op ← {func, type, step, trail}
8 ***push***(PQ, op, priority)
9 **end**
10 **end**
11 **while** ***len***(PQ) > 0 **and** count $<$ MAX_TRAIL **do**
12 op, count ← ***pop***(PQ), count $+ 1$
13 runtime, cost ←***deallocate***(op)
14 **if** runtime $>$ SLO **or** cost increases **then**
15 op.trail, op.step ←***allocate***(op)
16 **if** op.trail > 0 **then**
17 ***push***(PQ, op, 0)
18 **end**
19 **else**
20 priority ← reduced cost
21 ***push***(PQ, op, priority)
22 **end**
23 **end**
24 configs ← $\{(v_i.cpu, v_i.mem) \mid v_i \in L\}$
25 **return** configs
26 **End Function**

Priority Configuration Algorithm is provided in Algorithm 1 and Algorithm 2, with their functions explained in TABLE I.

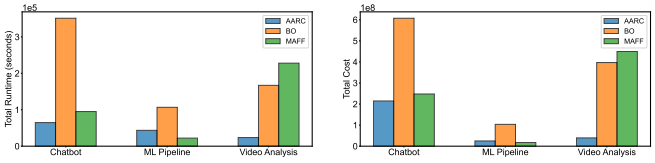
Algorithm 1 presents the whole scheduling process. Given a function workflow and the target end-to-end SLO, the algorithm operates as follows: Firstly, each function in the workflow is initially assigned an over-provisioned base configuration to ensure the workflow meets the SLO in most cases (Line 2-4). The workflow is then executed, and the runtime for each function is used as the weight in a DAG to identify the critical path (Line 5-6). The critical path and the end-to-end SLO are then utilized as inputs for the Priority Configurator, which in turn generates optimized configurations for each function along the critical path (Line 7-9). Through a full DAG traversal, the algorithm identifies sub-paths linked to the critical path, defined by their start and end nodes within the critical path, and no intersections with other nodes (Line 10). For each sub-path, the algorithm calculates the sub-SLO as the time interval between critical path nodes, ensuring critical path consistency during scheduling (Line 12). The Priority Configurator configures each function along the sub-path, following the same method as with the critical path (Line 19-20). To prevent conflicts and ensure each function only schedules once, the algorithm sets a scheduled flag for every function. After scheduling a function, the algorithm removes it from the path and adjusts the SLO accordingly (Line 13-18).

Algorithm 2 presents *Priority Configuration Algorithm*, which adopts priority scheduling to optimize the configuration of sequentially executed functions subject to SLO limitation. The algorithm initializes a priority queue PQ to manage operations for each function (Line 2), which requires two distinct operations to adjust CPU and memory quotas (Line 3-10). Based on the operation type, the algorithm modifies the function's resources, continuously executing operations from PQ and monitoring their impact on runtime and cost (Line 12-13). If an operation violates the SLO, increases cost, or encounters an error, the algorithm reverts the resource (Line 14-18). Simultaneously, an exponential backoff mechanism reduces the step size to ensure convergence while avoiding over-scheduling (Line 15). Operations consistently causing violations (trail $= 0$) are removed from PQ (Line 16), while those with potential are re-enqueued with adjusted priorities (Line 17, 20-21). The loop terminates when PQ is empty or a user-defined iteration limit, $\text{MAX}_{\text{TRAIL}}$, is reached (Line 11).

IV. Performance Evaluation

A. Experiment Setup

a) Environment: The experiments run on a machine with 4-socket Intel Xeon Gold 6248R (96 physical core),



(a) Total Runtime (b) Total Cost

Fig. 5: Overall Sample Cost and Runtime Comparison.

TABLE II: Average Runtime and Cost Comparison.

	Chatbot		ML Pipeline		Video Analysis	
	Runtime (s)	Cost	Runtime (s)	Cost	Runtime (s)	Cost
AARC	103.7±3.2	2390.9k	77.1±2.6	435.0k	316.8±6.6	53.6k
BO	114.7±1.9	4275.2k	60.0±0.7	863.5k	519.9±8.3	82.4k
MAFF	115.3±3.1	3477.5k	109.5±2.0	1136.6k	578.2±19.3	98.8k

and 512GB memory. Workflows execute in separate Docker containers, enabling CPU and memory allocation decoupling.

b) Baselines: We compare our approach with two baselines: Bayesian Optimization (BO) [8] and MAFF gradient descent [14]. Originally designed for single-function resource configuration and memory-centric tasks, these methods have been adapted for workflow optimization. In the Bayesian Optimization method, configuration parameters are discretized to limit the search space. Memory allocation is available in 64 MB increments from 128 MB to 10,240 MB, while vCPU cores range from 0.1 to 10, independently of memory. The MAFF gradient descent method iteratively minimizes cost, allocating vCPU cores proportionally (1 core per 1,024 MB of memory). If a workflow's SLO is violated, the process reverts to the previous step and terminates.

c) Workloads: Our experiments utilize three workflows (Chatbot, ML Pipeline, and Video Analysis), with SLOs set to 120s, 120s, and 600s, respectively. These applications capture key characteristics of serverless DAGs. Specifically, Video Analysis and Chatbot exhibit a scatter communication pattern, while ML Pipeline follows a broadcast pattern [9].

d) Metrics: We evaluate performance using runtime and cost metrics, extending the pricing model of AWS Lambda to decoupled resources. Let $\texttt{cost}_{\texttt{ij}}$ represent the cost of serverless function $\texttt{v}_\texttt{i}$ configured to $(\texttt{cpu}_\texttt{j}, \texttt{mem}_\texttt{j})$ with runtime $\texttt{t}_{\texttt{ij}}$. We denote the price per vCPU-second as μ_0, the price per GB-second as μ_1, and the price for function requests and orchestration as μ_2. All are constants. The cost equation is then $\texttt{cost}_{\texttt{ij}} = \texttt{t}_{\texttt{ij}}(\mu_0 \cdot \texttt{cpu}_\texttt{j} + \mu_1 \cdot \texttt{mem}_\texttt{j}) + \mu_2$. Here, μ_0, μ_1, and μ_2 are set to 0.512, 0.001, and 0, respectively.

B. Effectiveness of Configuration Search

In the configuration search experiment, we set the initial configurations for each workflow. Then, we perform the configuration search using three methods: AARC, Bayesian Optimization, and MAFF.

a) Overall Efficiency: Fig. 5 shows the total runtime and cost of the sampling process. In all workflows, AARC outperforms the Bayesian Optimization, especially in the Video Analysis workflow, reducing runtime by 85.8% and cost by 90.1%. This improvement stems from AARC's priority scheduling strategy for decoupled resources, which expands the configuration search space while requiring fewer samples. In the Chatbot workflow, AARC performs 64 samples, slightly more than MAFF's 61, but achieves a 31.9% reduction in runtime and 13.4% in cost. This is because MAFF's proportional allocation scheme reduces the search space but risks local optima, leading to higher runtime and costs due to resource coupling. In the Video Analysis workflow, AARC reduces runtime and cost by 89.6% and 91.3% compared to MAFF. However, in the ML Pipeline workflow, AARC samples 50 times compared to MAFF's 15, resulting in lower runtime and cost for MAFF. This is due to the ML Pipeline's high CPU and low memory demands, where proportional adjustments often hit local optima, requiring fewer samples to meet exit conditions.

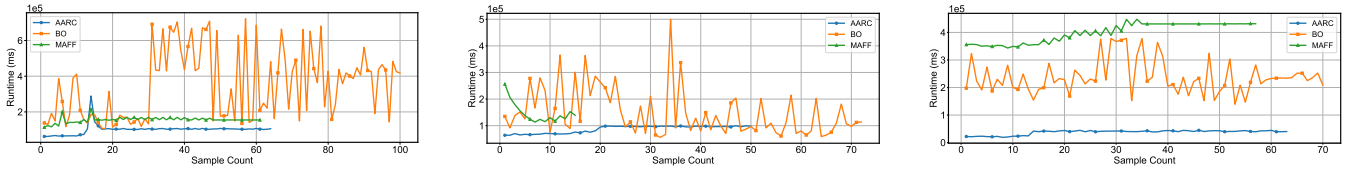
b) Sampling Efficiency: We further illustrate the effectiveness of optimal configurations calculated by each method as the sampling count increases, evaluating these outcomes through workflow runtime and cost. Fig. 6 shows the change in runtime with sampling count under different configurations for each workflow. Our goal is to minimize cost while meeting the SLO, so runtime shows an upward trend using AARC. The Bayesian Optimization method has the highest sampling count and significant instability, as its efficiency decreases due to the large search space created by resource decoupling. In contrast, our priority scheduling strategy maintains search efficiency while accelerating convergence. Fig. 7 depicts the change in cost with sampling count under different configurations. Using AARC, cost shows a downward trend and converges with fewer samples. In the ML Pipeline workflow, the MAFF method quickly falls into local optima due to its coupled resource configuration search approach, making it difficult to discover more cost-effective configurations.

C. Performance of Optimal Configuration

To validate the performance of AARC in meeting the SLO requirements and execution cost, we execute the workflow 100 times using the configurations generated by the aforementioned methods and calculate its average runtime and cost.

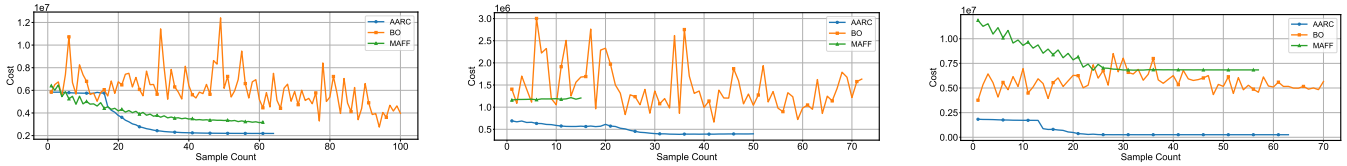
a) SLO Violation: All methods meet the SLO constraints, with the average runtime and standard deviation shown in TABLE II. AARC satisfies SLO because the algorithm reverts resources during configuration search when SLO violations occur and incorporates sub-paths into the critical path, ensuring both critical path consistency and SLO compliance. This demonstrates the effectiveness of our approach in meeting SLO requirements and its potential for integration into real-world systems with high reliability and performance standards.

b) Cost Reduction: TABLE II shows the execution cost of workflows with configurations found by different methods. In the Chatbot, ML Pipeline, and Video Analysis workflows, AARC reduces costs significantly compared to Bayesian Optimization and MAFF. The reductions are 44.0% and 31.2% for Chatbot, 49.6% and 61.7% for ML Pipeline, and 34.9% and 45.7% for Video Analysis, respectively. Compared to the Bayesian Optimization method, our approach uses priority scheduling, which allows for more stable identification of suitable decoupled resource configurations. Unlike MAFF's



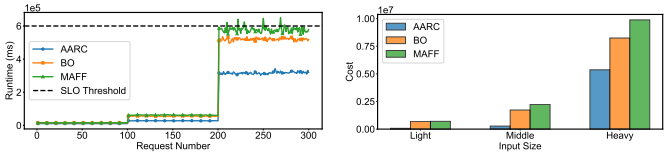
(a) Chatbot (b) ML Pipeline (c) Video Analysis

Fig. 6: Runtime Changing with Sample Counts of Different Methods under Different Workflows.



(a) Chatbot (b) ML Pipeline (c) Video Analysis

Fig. 7: Cost Changing with Sample Counts of Different Methods under Different Workflows.



(a) Runtime (b) Cost

Fig. 8: Performance Across Input Sizes in Video Analysis.

proportional configuration method, our approach achieves decoupled resource allocation, resulting in lower-cost configurations. Since the ML Pipeline has high CPU demands and low memory demands, decoupling resources yields better results.

*D. **Discussion**: Input-Aware Configuration*

Considering that workflow execution efficiency can be input-sensitive, we further enhanced our design by adding an Input-Aware Configuration Engine Plugin. The Video Analysis workflow is input-sensitive, where different input video sizes correspond to different optimal configurations. If developers trigger the plugin, the Engine analyzes the characteristics of the input data, such as video bitrate and duration. Based on the identified features, the Engine sorts the inputs and invokes Graph-Centric Scheduler and Priority Configurator to determine the optimal resource configuration scheme for each input. When a request arrives, the Engine analyzes the input scale and allocates the input to different configurations.

To validate our design, we evaulate Video Analysis workflow using three input sizes: light, middle, and heavy. Fig. 8a shows workflow runtime with light, middle, and heavy inputs in sequence. Since MAFF does not adjust configurations for input-sensitive workflows, it may violate the SLO under heavy inputs, while our method consistently remains within SLO limits. Fig. 8b shows the average cost of the workflow under the three input sizes. Because our method can select configurations based on input size, it optimizes cost by 89.9% and 89.8% compared to MAFF and Bayesian Optimization under light input, and by 45.7% and 34.9% under heavy input.

V. Related Work

a) Serverless Resource Configuration: Configuring resources in serverless computing involves optimizing memory and CPU allocations to balance performance and cost. Offline approaches like MAFF [14] and COSE [12] rely on optimization algorithms and performance modeling to determine configurations. Bilal *et al.* [8] explores decoupling memory and CPU configurations, leveraging diverse VM types to enhance optimization but still faces challenges with workload variability. To address such dynamism, online configuration methods have emerged. Sizeless [15] uses real-time performance data and prediction models to adjust resources dynamically. FaaSDeliver [16] further refines this by monitoring per-invocation metrics and employing advanced estimators for resource allocation. However, these methods introduce overhead from continuous monitoring and data processing, potentially inflating costs for serverless applications.

b) Serverless Workflows Optimization: Due to the prevalence of workflows in serverless workloads, workflow configuration optimization has gained attention from academia and industry [17]–[19]. SLAM [20] uses distributed tracing to detect relationships between functions and leverages execution times under different memory configurations to estimate overall application execution time. Raza *et al.* [21] improve COSE for serverless workflows, optimizing performance and cost across the entire workflow. StepConf [13] provides dynamic memory allocation for serverless workflows, considering intra-function and inter-function parallelism to fully utilize the parallel capacity of serverless functions. However, these methods still do not effectively decouple resources, leaving room for further optimization.

VI. Conclusion

We propose AARC, an affinity-aware resource configuration framework for serverless workflows, capitalizing on the concept of resource decoupling within the serverless paradigm. By leveraging our framework, developers are relieved of the burden of manual serverless workflow configuration. Experimental results indicate that, during the process of searching for the optimal configuration, AARC achieves a total runtime reduction of 85.8% and 89.6% compared to Bayesian Optimization and MAFF, respectively. Additionally, it reduces resource usage by 49.6% and 61.7% while satisfying the SLO requirements.

REFERENCES

[1] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, "The serverless computing survey: A technical primer for design architecture," *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–34, 2022.

[2] A. Mampage, S. Karunasekera, and R. Buyya, "A holistic view on resource management in serverless computing environments: Taxonomy and future directions," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, 2022.

[3] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (IaaS) in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 41, pp. 424–440, 2014.

[4] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (iaas)," *International Journal of engineering and information Technology*, vol. 2, no. 1, pp. 60–63, 2010.

[5] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of art and research challenges," *IEEE Transactions on Service Computing*, vol. 11, no. 2, pp. 430–447, 2017.

[6] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[7] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proceedings of USENIX Annul Technical Conference*, 2020.

[8] M. Bilal, M. Canini, R. Fonseca, and R. Rodrigues, "With great freedom comes great opportunity: Rethinking resource allocation for serverless functions," in *Proceedings of European Conference on Computer Systems*, 2023, pp. 381–397.

[9] A. Mahgoub, E. B. Yi, K. Shankar, S. Elnikety, S. Chaterji, and S. Bagchi, "Orion and the three rights: Sizing, bundling, and prewarming for serverless dags," in *Proceedings of Symposium on Network System Design and Implementation*, 2022.

[10] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, "Ensure: Efficient scheduling and autonomous resource management in serverless environments," in *IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2020.

[11] H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "Reinforcement learning for resource management in multi-tenant serverless platforms," in *Proceedings of European Workshop on Machine Learning and Systems*, 2022.

[12] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "Cose: Configuring serverless functions using statistical learning," in *IEEE International Conference on Computer Communications*. IEEE, 2020, pp. 129–138.

[13] Z. Wen, Y. Wang, and F. Liu, "Stepconf: Slo-aware dynamic resource configuration for serverless function workflows," in *IEEE International Conference on Computer Communications*. IEEE, 2022, pp. 1868–1877.

[14] T. Zubko, A. Jindal, M. Chadha, and M. Gerndt, "Maff: Self-adaptive memory optimization for serverless functions," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2022, pp. 137–154.

[15] S. Eismann, L. Bui, J. Grohmann, C. Abad, N. Herbst, and S. Kounev, "Sizeless: Predicting the optimal size of serverless functions," in *Proceedings of International Middleware Conference*, 2021.

[16] G. Yu, P. Chen, Z. Zheng, J. Zhang, X. Li, and Z. He, "Faasdeliver: Cost-efficient and qos-aware function delivery in computing continuum," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3332–3347, 2023.

[17] Z. Li, Y. Liu, L. Guo, Q. Chen, J. Cheng, W. Zheng, and M. Guo, "Faasflow: Enable efficient workflow execution for function-as-a-service," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 782–796.

[18] Z. Li, C. Xu, Q. Chen, J. Zhao, C. Chen, and M. Guo, "Dataflower: Exploiting the data-flow paradigm for serverless workflow orchestration," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023, pp. 57–72.

[19] W. Wang, Q. Wu, Z. Zhang, J. Zeng, X. Zhang, and M. Zhou, "A probabilistic modeling and evolutionary optimization approach for serverless workflow configuration," *Software: Practice and Experience*, vol. 54, no. 9, pp. 1697–1713, 2024.

[20] G. Safaryan, A. Jindal, M. Chadha, and M. Gerndt, "Slam: Slo-aware memory optimization for serverless applications," in *IEEE International Conference on Cloud Computing*. IEEE, 2022, pp. 30–39.

[21] A. Raza, N. Akhtar, V. Isahagian, I. Matta, and L. Huang, "Configuration and placement of serverless applications using statistical learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1065–1077, 2023.

Curvilinear Optical Proximity Correction via Cardinal Spline

Su Zheng[1], Xiaoxiao Liang[2], Ziyang Yu[1], Yuzhe Ma[2], Bei Yu[1], Martin Wong[3]
[1]Chinese University of Hong Kong, [2]Hong Kong University of Science and Technology (GZ), [3]Hong Kong Baptist University

***Abstract*—This paper presents a novel curvilinear optical proximity correction (OPC) framework. The proposed approach involves representing mask patterns with control points, which are interconnected through cardinal splines. Mask optimization is achieved by iteratively adjusting these control points, guided by lithography simulation. To ensure compliance with mask rule checking (MRC) criteria, we develop comprehensive methods for checking width, space, area, and curvature. Additionally, to match the performance of inverse lithography techniques (ILT), we design algorithms to fit ILT results and resolve MRC violations. Extensive experiments demonstrate the effectiveness of our methodology, highlighting its potential as a viable OPC/ILT alternative.**

I. Introduction

Integrated circuit (IC) manufacturing is a highly complex process that transforms raw materials into tiny chips. It begins with the design of the circuit layout, which is then transferred onto a photomask. Lithography plays a central role in printing the mask patterns onto the silicon wafer. As IC designs become more complex and feature sizes continue to shrink, the demand for well-optimized masks increases significantly.

Mask optimization [1]–[12] is a crucial process in semiconductor manufacturing, involving techniques like rule-based optical proximity correction (OPC), model-based OPC, sub-resolution assist features (SRAF), and inverse lithography technology (ILT). Rule-based OPC [13]–[15] uses predefined rules to correct distortions. Model-based OPC [16]–[18] adopts lithography simulation to guide the correction. SRAFs [19] are small, sub-wavelength features added to the mask that impact the behavior of light passing through them. ILT [20]–[30] treats mask optimization as an inverse problem and iteratively modifies the mask to get desired wafer patterns.

Although ILT is recognized for its high pattern fidelity, model-based OPC offers several advantages in terms of efficiency, reliability, and mask manufacturability. OPC is more computationally efficient and converges better, as the optimization problems of ILT are more complex and resource-intensive. Additionally, OPC primarily relies on geometric operations, whereas ILT typically requires intensive image-based processes. This allows researchers to integrate mask rule checking (MRC) into model-based OPC flows, resulting in mask patterns with improved manufacturability. While multi-beam mask writers (MBMW) can support curvilinear mask patterns, ILT-generated masks often suffer from poor manufacturability due to complex non-Manhattan geometries. Therefore, a methodology that combines the efficiency, reliability, and mask manufacturability of OPC with the high pattern fidelity of ILT would be highly beneficial.

Curvilinear OPC [31], [32] aims to bridge the gap between OPC and ILT by allowing curvilinear mask shapes in OPC. It maintains the ability to handle curvilinear shapes and any-angle edges while having lower computational complexity than ILT. Furthermore, it can produce more manufacturable masks, as the mask patterns can be optimized to better comply with mask rules. This approach strikes a balance between the simplicity of OPC and the precision of ILT, providing a more practical solution for mask optimization.

In this paper, we propose a novel curvilinear OPC framework, **CardOPC**. By representing mask patterns with control points connected through cardinal splines, our approach allows for flexible and precise mask representation. This method leverages the inherent smoothness and adaptability of splines to accurately capture intricate geometries. As illustrated by Fig. 1, the control point representation in curvilinear OPC can inherit the advantage of fewer variables from traditional OPC while gaining the flexibility of curvilinear shapes from ILT. Compared to the existing methods using Bézier splines [31], [32], cardinal splines offer significant advantages, particularly in terms of computational efficiency, without sacrificing the strengths of Bézier splines like local control and smoothness. This reduces the computational overhead and simplifies the optimization process.

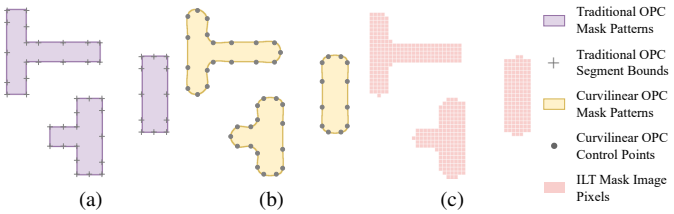


Fig. 1 Illustration of the mask pattern representation in (a) traditional OPC; (b) curvilinear OPC; (c) ILT. Curvilinear OPC combines the reduced number of variables from traditional OPC and the flexibility of curvilinear shapes from ILT.

The iterative adjustment of control points, guided by lithography simulation, achieves high-quality mask optimization. This process ensures that the wafer patterns closely match the target design, minimizing deviations that could affect the quality of products. The utilization of control points and splines can maintain a small number of variables while being able to optimize complex shapes, offering superior efficiency and adaptability in mask optimization.

For the manufacturability of masks, we propose comprehensive methods for checking MRC violations, covering width, space, area, and curvature rules. We demonstrate that the analytical spline-based representation simplifies mask rule checking and makes curvilinear OPC superior to ILT in terms of MRC. Additionally, our framework incorporates techniques to fit ILT results and resolve MRC violations, positioning it as a promising alternative to traditional ILT methods. By integrating ILT fitting capabilities, our approach can leverage the strengths of ILT while mitigating its limitations. This hybrid approach allows for more robust and effective mask optimization, combining the best aspects of both OPC and ILT.

The major contribution of this paper can be summarized as follows:

- We propose a curvilinear OPC framework based on cardinal splines. It combines the computational efficiency of traditional OPC and the flexibility of ILT.
- We derive the methods for curvilinear mask rule checking, which are the foundation of curvilinear OPC's advantages in addressing MRC violations.
- By integrating ILT fitting capabilities, the framework leverages the strengths of ILT while mitigating its limitations, offering a more robust and effective mask optimization solution.
- Extensive experiments reveal the superiority of the proposed framework over existing methods, showcasing higher precision, better scalability, and greater efficiency.

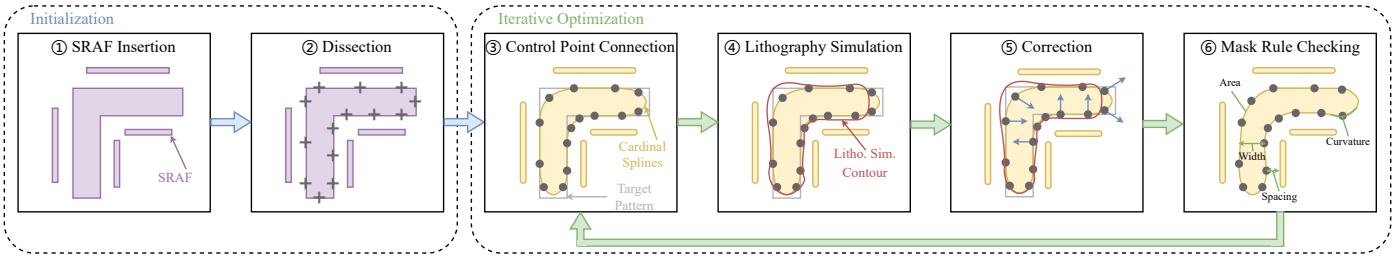


Fig. 2 Overview of our CardOPC. In the initialization phase, we ① insert SRAFs and ② dissect the polygons as traditional OPC does. The segments are converted to the control points of cardinal splines. During the optimization phase, we ③ connect the control points with cardinal splines to form the mask patterns, which are subsequently input to the ④ lithography simulation engine. According to the lithography simulation contours, we ⑤ estimate the edge displacement and correct the distortions by moving the control points. After that, ⑥ mask rule checking and MRC violation resolving are performed to ensure the mask manufacturability.

II. Preliminaries

A. Lithography Simulation

Lithography simulation is a crucial step in mask optimization, essential for obtaining the desired printed patterns on the wafer. The process starts with the input mask $\boldsymbol{M}$, which is projected through an optical system to form the aerial image $\boldsymbol{I}$. For example, the Hopkins diffraction model [33] can be employed to efficiently approximate the projection behavior. In this model, the aerial image $\boldsymbol{I}$ is obtained by applying a set of optical kernels $\boldsymbol{H}$ to the mask $\boldsymbol{M}$ with:

$$\boldsymbol{I}(x,y) = \sum_{k=1}^{N_h} w_k \left| \boldsymbol{M}(x,y) \otimes \boldsymbol{h}_k(x,y) \right|^2 . \tag{1}$$

N_h is the number of optical kernels, $\boldsymbol{h}_k$ is the k-th optical kernel in $\boldsymbol{H}$, and w_k is the corresponding weight. $\boldsymbol{I}$ can be binarized using the intensity threshold I_{th} that indicates the exposure level. After the binarization, the contour $\boldsymbol{C}$ of the lithography simulation result can be extracted and utilized to guide the OPC process.

B. Optical Proximity Correction

When light passes through a photomask and projects onto a wafer, various optical phenomena such as diffraction and interference can cause deviations from the intended pattern. These deviations, known as optical proximity effects, can lead to issues such as line width variations and pattern displacement. These effects become more significant as feature sizes shrink, making it challenging to achieve the desired pattern fidelity. OPC addresses this issue by modifying the mask patterns to compensate for distortions brought by lithography. Model-based OPC [17] typically involves the following steps:

1) Dissection (or fragmentation): The edges of the input polygons are dissected into smaller segments, usually with more segments around corners to allow for finer control.
2) SRAF insertion: SRAFs are inserted based on rules regarding the width and distance of the assist features from the main features, in order to improve the printability and process window.
3) Iterative optimization: The algorithm iteratively retrieves lithography simulation results and adjusts the segments to minimize edge placement error (EPE) violations.

EPE is a metric that evaluates the difference between the lithography simulation contour and the target pattern. To calculate EPE, a series of points are sampled along the edges of the target design, and the distance between each point's position in the target pattern and its position in the lithography simulation contour is measured. If the EPE of a point exceeds a specified threshold, it is counted as an EPE violation. L2 is another way to estimate the error of the printed patterns. It is the sum of the squared errors between the lithography simulation result and the target pattern. Furthermore, under varying lithography conditions, the printed patterns can be different. Therefore, process variation band (PVB) is also widely adopted as a metric of OPC, which evaluates the maximum discrepancy between the printed patterns under varying process conditions.

C. Mask Rule Checking

Mask Rule Checking (MRC) is a common practice in the semiconductor industry to ensure the manufacturability of photomask data before mask writing. Traditional MRC checks the rectilinear mask patterns for compliance with mask rules, such as minimum feature sizes, minimum distances, corner-to-corner distance, etc. As the industry moves towards curvilinear mask shapes to improve printed patterns, the traditional MRC approach needs to be adapted.

Reference [34] introduces the following curvilinear mask rules:

1) Spacing and width constraints: These rules ensure minimum distances between patterns and minimum widths of the shapes.
2) Area constraint: It accounts for the resolution limits, as the patterns may fail to print reliably if they are too tiny.
3) Curvature constraint: This rule limits the range of the curvilinear patterns' curvatures to ensure reliability.

Combining these constraints provides a comprehensive set of rules to ensure the manufacturability of curvilinear mask patterns.

III. CardOPC Methodology

A. Overview

Fig. 2 outlines the proposed curvilinear OPC flow. Initially, we insert SRAFs into the mask and dissect the edges of polygons into segments, following traditional OPC. These segments are then transformed into the control points. During optimization, these control points are connected using cardinal splines to generate mask patterns, which are subsequently fed into lithography simulation. Based on the lithography simulation contours, edge displacements are estimated, and distortions are corrected by adjusting the control points. Finally, mask rule checking and MRC violation resolving are performed to ensure the manufacturability of the mask.

B. Initialization

Fig. 3 summarizes the initialization phase of CardOPC. Fig. 3(a) shows an example of simple rule-based SRAF insertion. It creates an SRAF with a length of $l_s = r \times l_m$, where l_m is the length of the main pattern edge and r is a constant that controls the ratio between the SRAF and main pattern. The SRAF is placed at a distance of d_{ms} from the main pattern. Note that SRAF insertion can

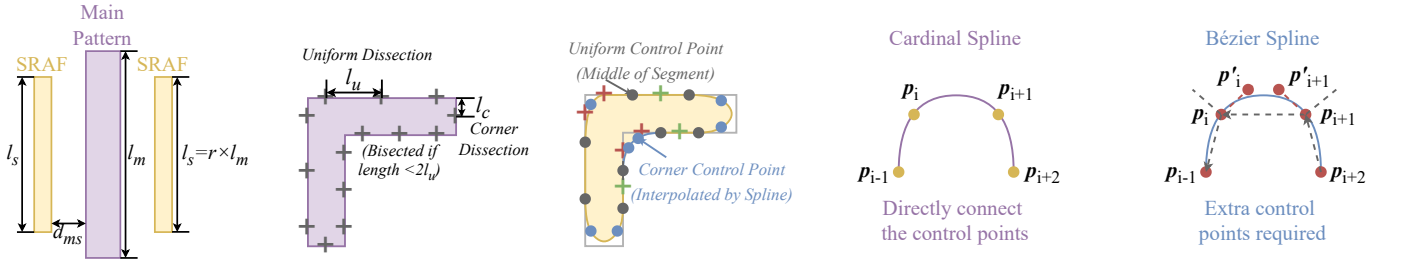


Fig. 3 Illustration of the initialization phase of our curvilinear OPC flow. (a) An example of simple rule-based SRAF insertion. The SRAF is placed at a certain distance from the main pattern, with a shorter length than the main pattern. (b) Basic dissection method of our OPC flow. It creates shorter segments around corners, and longer segments for the remaining parts of the edges. (c) Generation of control points. Most control points are chosen as the middle points of the segments. However, around corners, it interpolates the segment boundary points with cardinal splines to generate the control points.

Fig. 4 Comparison between cardinal splines and Bézier splines in curvilinear OPC. Using cardinal splines, the curve can pass through all the control points $\boldsymbol{p}_{i-1}$, $\boldsymbol{p}_i$, $\boldsymbol{p}_{i+1}$, and $\boldsymbol{p}_{i+2}$. However, when using Bézier splines, two extra control points $\boldsymbol{p}'_i$ and $\boldsymbol{p}'_{i+1}$ should be generated to ensure that the curve can pass through $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$. This introduces non-negligible computational overhead.

also be completed by calling external tools (e.g. Calibre) or fitting ILT results (see Section III-G). Fig. 3(b) demonstrates our basic dissection method. Specifically, it creates shorter segments with a length of l_c around corners, where finer control is needed, and longer segments with a length of l_u for the remaining parts of the edges. Fig. 3(c) illustrates the generation of control points. Most control points are chosen as the middle points of the segments obtained from the dissection step. However, around corners, it interpolates neighboring segment boundary points using cardinal splines to generate the control points for the corner regions. Additionally, we also convert the SRAFs to cardinal splines to achieve a uniform pattern representation.

C. Cardinal Splines for Control Point Connection

Cardinal spline is a type of spline that is used in mathematics and computer graphics to create smooth and flexible shapes. This type of interpolating spline can pass through a given set of control points. This feature makes cardinal splines particularly useful for creating smooth curves that follow a specified path.

In this paper, we adopt cardinal splines for the following reasons:

1) The curve passes through each control point, ensuring that the shape of the curve is influenced directly by these points.
2) Cardinal splines include a tension parameter s that controls the tightness of the curve. This allows users to finetune the curvilinear shapes without moving the control points.
3) The tangents, normals, and curvatures of cardinal splines can be analytically computed, ensuring efficient edge displacement checking and mask rule verification in curvilinear OPC.

To connect two points $\boldsymbol{p}_i = (x_i, y_i)$ and $\boldsymbol{p}_{i+1} = (x_{i+1}, y_{i+1})$ with cardinal spline, we need their predecessor $\boldsymbol{p}_{i-1}$ and successor $\boldsymbol{p}_{i+2}$. Any point between $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ can be given by:

$$\boldsymbol{p}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_{i-1} \\ \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \\ \boldsymbol{p}_{i+2} \end{bmatrix}, \quad t \in [0,1],\ \boldsymbol{p}(0) = \boldsymbol{p}_i,\ \boldsymbol{p}(1) = \boldsymbol{p}_{i+1}. \tag{2}$$

In practice, we connect each pair of control points $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ by sampling multiple points in $t \in [0,1]$ evenly. For simplicity, we can represent equation (2) using $\boldsymbol{p}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \boldsymbol{S}_{card}\, \boldsymbol{P}_{i-1:i+3}$. The computation can be accelerated by GPU using PyTorch.

Fig. 4 compares cardinal splines and Bézier splines in curvilinear OPC. With cardinal splines, the curve can seamlessly traverse all the control points, namely $\boldsymbol{p}_{i-1}$, $\boldsymbol{p}_i$, $\boldsymbol{p}_{i+1}$, and $\boldsymbol{p}_{i+2}$. Conversely, Bézier splines, necessitate the generation of two additional control points, $\boldsymbol{p}'_i$ and $\boldsymbol{p}'_{i+1}$, to guarantee the passage through $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ [34]. This requirement incurs a non-trivial computational overhead.

D. Lithography Simulation

Our experiments involve two lithography simulation tools. One of them is from the ICCAD-13 contest [35], which is implemented using equation (1) with GPU acceleration [36]. The other one is Calibre, with the same settings as the previous works [8], [9].

E. Correction

As introduced by [37], the primary goal of OPC is to minimize the edge placement error (EPE) by modifying mask edges. Given E EPE checking points, we can represent the EPEs with a function $\boldsymbol{f}_{EPE}(\boldsymbol{M})$, which contains the distances between checking points' positions in the target pattern and their positions in the lithography simulation contour. It can be decomposed using Taylor expansion:

$$\boldsymbol{f}_{EPE}(\boldsymbol{M}_{\tau+1}) = \boldsymbol{f}_{EPE}(\boldsymbol{M}_\tau) + \boldsymbol{f}'_{EPE}(\boldsymbol{M}_\tau) \times \boldsymbol{\Delta M}_\tau + \epsilon(\boldsymbol{M}_\tau), \tag{3}$$

where τ is the current iteration, $\boldsymbol{M}_\tau$ is the current mask, $\boldsymbol{\Delta M}_\tau$ is the modification of the mask at the current iteration, and $\epsilon(\boldsymbol{M}_\tau)$ represents the high order terms. To guarantee a manageable complexity, the high-order terms can be ignored, and equation (3) becomes:

$$\boldsymbol{f}_{EPE}(\boldsymbol{M}_{\tau+1}) \approx \boldsymbol{f}_{EPE}(\boldsymbol{M}_\tau) + \boldsymbol{f}'_{EPE}(\boldsymbol{M}_\tau) \times \boldsymbol{\Delta}\text{Positions}_\tau. \tag{4}$$

$\boldsymbol{\Delta M}_\tau$ in equation (3) becomes $\boldsymbol{\Delta}\text{Positions}_\tau$ in equation (4), which represents the moving distance of the segments or control points. The matrix $\boldsymbol{f}'_{EPE}(\boldsymbol{M}_\tau)$ reflects how movements affects the EPE. Given D segments or control points, $\boldsymbol{f}'_{EPE}(\boldsymbol{M}_\tau)$ can be defined as:

$$\boldsymbol{f}'_{EPE}(\boldsymbol{M}_\tau) = \begin{bmatrix} \frac{\partial e_1}{\partial d_1} & \frac{\partial e_1}{\partial d_2} & \cdots & \frac{\partial e_1}{\partial d_D} \\ \frac{\partial e_2}{\partial d_1} & \frac{\partial e_2}{\partial d_2} & \cdots & \frac{\partial e_2}{\partial d_D} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_E}{\partial d_1} & \frac{\partial e_E}{\partial d_2} & \cdots & \frac{\partial e_E}{\partial d_D} \end{bmatrix}, \tag{5}$$

where $\frac{\partial e_i}{\partial d_j}$ is the partial derivative of the i-th EPE with respect to the moving distance of the j-th segment or control point. Since our target

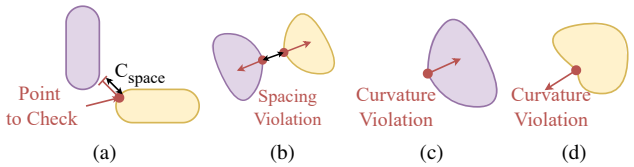


Fig. 5 Illustration of mask rule checking and MRC violation resolving. (a) shows how to create a line segment to test the spacing rule violation. (b) moves the control points to resolve spacing rule violations. (c) and (d) resolve curvature rule violations by moving the control points in and out, respectively.

is to make $\boldsymbol{f}_{EPE}(\boldsymbol{M}_{\tau+1}) = 0$, the moving distance of segments or control points at each iteration can be given by:

$$\boldsymbol{\Delta}\text{Positions}_{\tau} = -\boldsymbol{f}'_{EPE}(\boldsymbol{M}_{\tau})^{-1} \times \boldsymbol{f}_{EPE}(\boldsymbol{M}_{\tau}). \tag{6}$$

In a basic OPC solver, we can check EPE for each segment or control point, which means that $E = D$. Moreover, we can assume that $\frac{\partial e_i}{\partial d_i}$ is a constant and $\frac{\partial e_i}{\partial d_j} = 0$ if $i \neq j$. Under these settings, $\boldsymbol{f}'_{EPE}(\boldsymbol{M}_{\tau})^{-1}$ becomes a diagonal matrix $\boldsymbol{f}'_{EPE}(\boldsymbol{M}_{\tau})^{-1} = \gamma \boldsymbol{I}$, where γ is the moving distance of each segment or control point. To improve the moving smoothness and mimic multi-segment solvers, we can take a weighted average on the moving distances/directions of neighboring points. For instance, given moving distance Δd_i from equation (6) and $2W + 1$ neighboring points $\{d_{i-W}, d_{i-W+1}, ..., d_{i+W}\}$ from the same shape, the final moving distance $\overline{\Delta d_i}$ is:

$$\overline{\Delta d_i} = \sum_{k=-W}^{W} w_k \Delta d_k. \tag{7}$$

In curvilinear OPC, to guarantee the support of any-angle edges, we need to calculate both the moving distance Δd_i and the moving direction $\boldsymbol{n}_i$ for each control point. We use the normal vectors $\boldsymbol{n}(t)$ of the cardinal splines as the moving directions, which can be derived from the derivatives, $\boldsymbol{g}(t) = \boldsymbol{p}'(t)$, with the following equations:

$$\boldsymbol{g}(t) = \begin{bmatrix} 0 & 1 & 2t & 3t^2 \end{bmatrix} \boldsymbol{S}_{card} \boldsymbol{P}_{i-1:i+3}. \tag{8a}$$

$$\overline{\boldsymbol{g}}(t) = (\overline{g}_x(t), \overline{g}_y(t)) = \frac{\boldsymbol{g}(t)}{\|\boldsymbol{g}(t)\|}, \tag{8b}$$

$$\boldsymbol{n}(t) = (n_x(t), n_y(t)) = (-\overline{g}_y(t), \overline{g}_x(t)). \tag{8c}$$

The moving direction $\boldsymbol{n}_i$ can be given by the normal vector at the corresponding t, and the moving step is $\Delta d_i \boldsymbol{n}_i$. In practice, we can take the weighted average among $\Delta d_i \boldsymbol{n}_i$ rather than Δd_i to allow the control points to move in more directions, providing a higher degree of freedom. Note that GPU acceleration is also applicable for computing moving directions. Additionally, our curvilinear OPC framework is compatible with other gradient-based OPC algorithms like the MEEF-based OPC [38] and differentiable OPC [12] methods.

F. Mask Rule Checking and MRC Violation Resolving

Mask Rule Checking. We focus on the following mask rules:

1) Spacing rule: The spacing between two shapes should be larger than the spacing constraint C_{space}.
2) Width rule: The width of each shape should not be smaller than the width constraint C_{width}.
3) Area rule: The area of each shape should exceed the area constraint C_{area}.
4) Curvature rule: The curvature at each point should be less than the curvature constraint C_{curv}.

Algorithm 1 Fitting Method for ILT-Optimized Masks

Input: ILT-optimized mask image $\boldsymbol{M}$, cardinal spline function $\boldsymbol{F}(\cdot)$, ratio r_Q and r_R, learning rate α.

```
//Initialization;
Q = ∅;   //Control point set;
R = ∅;   //Reference point set;
for each shape S_i in the mask image M do
    Extract the boundary points of S_i, denoted by P_i;
    Sample r_Q|P_i| points from P_i evenly, add them to Q;
    Sample r_R|P_i| points from P_i evenly, add them to R;
end for
//Optimization;
for k ∈ {1, 2, ....., K} do //K iterations in total;
    Interpolate Q with F(·) to have |R| points;
    Compute the loss L(Q, R) = ‖F(Q) − R‖²;
    Optimize Q with Q ← Q − α ∂L(Q,R)/∂Q;
end for
return Q;
```

Among these rules, we provide basic solutions for space, width, and area checking, which are common topics in existing works like mask rule checking [39] and design rule checking [40]. We leave the well-optimized checking of these rules for future works. As for the curvature rule, we derive the analytical method for its checking, which is efficient because it can be accelerated by GPU.

For spacing and width checking, we construct an R-tree [41] using all shapes of the mask patterns. R-tree is a type of data structure used for indexing multi-dimensional information such as geographical coordinates, rectangles, or polygons. It stores spatial objects in a hierarchical manner using bounding rectangles. Each node in the tree represents a bounding rectangle that contains its child nodes. This structure allows for efficient querying of spatial data, such as querying the nearest shape and finding all objects that intersect a given area.

As demonstrated by Fig. 5(a), to check whether a point has a spacing violation, we build a line segment with a length of C_{space} that starts from the checking point, with its direction specified by its normal vector. If the line segment touches a shape, it has a spacing violation. It can be done efficiently using R-tree's querying algorithm. We implement width checking in a similar way, but create the line segment in the directions opposite to the point's normal vector.

The area of each shape can be calculated by the shoelace formula, which enables the checking of the area constraint. Both R-tree construction and area computation are provided by Shapely.

The curvature at a point can be defined as:

$$\boldsymbol{\kappa}(t) = \begin{vmatrix} p'_x(t) & p''_x(t) \\ p'_y(t) & p''_y(t) \end{vmatrix} / \|\boldsymbol{p}'(t)\|^3, \tag{9}$$

where $\boldsymbol{p}'(t) = (p'_x(t), p'_y(t))$, $\boldsymbol{p}''(t) = (p''_x(t), p''_y(t))$, $\boldsymbol{p}'(t)$ is calculated by equation (8), and $\boldsymbol{p}''(t)$ is given by:

$$\boldsymbol{p}''(t) = \begin{bmatrix} 0 & 0 & 2 & 6t \end{bmatrix} \boldsymbol{S}_{card} \boldsymbol{P}_{i-1:i+3}. \tag{10}$$

If one point has $\boldsymbol{\kappa}(t) > C_{curv}$, it has a curvature violation. This checking is used to ensure that the mask patterns are smooth enough and friendly for mask manufacturing.

MRC Violation Resolving. When MRC violation occurs, we can address it according to its type. As shown in Fig. 5(b), when a spacing violation occurs between two points, we try to move the corresponding control points in the directions opposite to their normal vectors, enlarging the spacing between the points. In the trials, the

TABLE I Via-layer OPC result comparison on EPE (nm) and PVB (nm^2)

Testcase	#Vias	DAMO [29]		Calibre		RL-OPC [8]		CAMO [9]		CardOPC	
		EPE	PVB	EPE	PVB	EPE	PVB	EPE	PVB	EPE	PVB
V1	2	7	5822	8	5837	6	5730	1	5797	2	5775
V2	2	8	5836	8	5834	7	5813	5	5734	4	5772
V3	3	14	8565	11	8587	13	8594	10	8470	5	8426
V4	3	14	8621	12	8771	14	8679	10	8576	7	8550
V5	4	18	10615	15	10775	16	10772	10	10503	5	10501
V6	4	20	10739	15	10763	19	10659	15	10507	8	10499
V7	5	28	12993	23	12615	23	12485	23	12097	7	12381
V8	5	26	13047	19	12784	24	12547	19	12437	16	12270
V9	6	30	15497	24	15454	26	15414	19	15186	15	15112
V10	6	35	15088	27	15064	33	14588	26	14556	16	14287
V11	6	39	15516	27	15782	31	15538	21	15333	9	15290
V12	6	36	15424	23	15686	24	15464	23	15204	18	15073
V13	6	32	16970	23	17035	39	17440	14	16712	6	16833
Average	4.5	23.6	11902.5	18.1	11922.1	21.2	11824.8	15.1	11624.0	**9.1**	**11597.6**
Ratio	-	156.3%	102.4%	119.9%	102.6%	140.4%	101.7%	100.0%	100.0%	**60.3%**	**99.8%**

TABLE II Metal-layer OPC result comparison on EPE (nm) and PVB (nm^2)

Testcase	#Points	Calibre		RL-OPC [8]		CAMO [9]		CardOPC	
		EPE	PVB	EPE	PVB	EPE	PVB	EPE	PVB
M1	64	49	28728	104	29390	44	27795	37	26387
M2	84	61	37386	117	39139	67	36467	49	34737
M3	88	81	39430	137	41623	59	39451	37	36491
M4	100	89	45741	252	46892	60	44961	44	40528
M5	106	66	47220	336	47041	69	46582	36	45283
M6	112	102	49887	355	51433	78	49438	23	47567
M7	116	89	52584	325	50770	83	49961	57	48794
M8	24	20	11014	32	10770	23	10928	1	10353
M9	72	50	22531	197	22360	42	22032	15	21139
M10	120	91	37546	263	36368	95	36849	11	37727
Average	88.6	69.8	37206.7	211.8	37578.6	62.0	36446.4	**31.0**	**34900.6**
Ratio	-	112.6%	102.1%	341.6%	103.1%	100.0%	100.0%	**50%**	**95.8%**

moving distance is chosen from small to large. The violation can usually be resolved in a few trials. For width violations, we use a similar method but move the control points in the directions specified by their normal vectors.

If an area rule violation occurs after moving the control points, we can cancel the move to avoid the violation. If it occurs after fitting ILT results, we remove the shape since it is usually a small and non-printable pattern.

When a curvature violation occurs at a point, we try to move the corresponding control point in and out according to the normal vector. Fig. 5(c) and Fig. 5(d) visualize this strategy. The violation can usually be resolved within a few attempts.

G. ILT-OPC Hybrid Approach

Continuous transmission mask (CTM) [12], [19], [42] is utilized to generate SRAFs for model-based OPC. Since CTM relies on ILT, it can serve as a hybrid method that combines ILT and OPC. However, existing methods can only generate rectilinear SRAFs according to the continuous mask intensity distribution from ILT, which can not fully utilize the ILT's ability. In this paper, we propose to fit ILT results with cardinal splines, which can mimic the complex curvilinear shapes in ILT-optimized masks, and resolve MRC violations using the strategies described in Section III-F. Our hybrid approach allows for more robust and effective mask optimization, combining the best aspects of both OPC and ILT.

As presented by Algorithm 1, our fitting method consists of the following steps:

1) Initialization: For each shape $\boldsymbol{S}_i$ in the mask image $\boldsymbol{M}$, we find the contour of $\boldsymbol{S}_i$ using the border following algorithm [43] (line 5), which is already implemented in the OpenCV toolbox. After that, we uniformly sample control points $\boldsymbol{Q}$ and reference points $\boldsymbol{R}$ from each shape's contour (lines 6-7). During optimization, we move the control points using gradient descent to fit the reference points.
2) Optimization: At each optimization step, we interpolate $\boldsymbol{Q}$ using the cardinal spline function $\boldsymbol{F}(\cdot)$ given by equation (2) (line 11). With the interpolated results, we can compute the mean squared error loss (line 12) between the interpolated points $\boldsymbol{F}(\boldsymbol{Q})$ and the reference points $\boldsymbol{R}$. Finally, we move $\boldsymbol{Q}$ using gradient descent (line 13) to minimize the distance between $\boldsymbol{F}(\boldsymbol{Q})$ and $\boldsymbol{R}$.

Note that the fitting algorithm can be implemented using PyTorch so that this process can be accelerated by GPU. Optimizers like Adam [44] can be adopted for the gradient descent in our method to accelerate the convergence.

TABLE III Large-scale OPC result comparison on EPE violations and PVB (μm^2)

Testcase	#Tiles ($30 \times 30 \mu m^2$)	Calibre		SimpleOPC [45]		CardOPC	
		EPE	PVB	EPE	PVB	EPE	PVB
gcd	1	3657	35.6651	3454	37.3002	3507	34.2606
aes	144	2722	27.7226	2571	29.4301	2578	27.3485
dynamicnode	144	2088	26.1663	1941	27.1213	1923	25.5011
Average	-	2409	26.9746	2260	28.3069	**2255**	**26.4519**
Ratio	-	100.0%	100.0%	93.8%	104.9%	**93.6%**	**98.1%**

After fitting the ILT-optimized masks, we obtain the control points $\boldsymbol{Q}$ of all mask patterns. The MRC violation resolving algorithms discussed in Section III-F are employed to address the MRC violations in the curvilinear shapes.

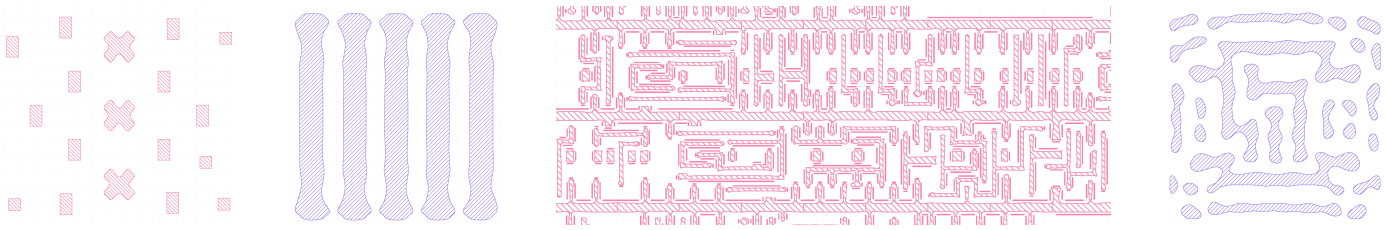
IV. Experiments

A. Comparison with Existing Works

Previous works focus on the OPC for metal-layer and via-layer patterns [8], [9]. We adopt the same testcases as them. For the experiments on via-layer patterns, we use 13 test layout clips [46], each of which is $2 \times 2 \mu m^2$. SRAFs are inserted by Calibre before the OPC algorithm launches. For metal-layer patterns, the dataset has 10 clips, each of which is $1.5 \times 1.5 \mu m^2$. The EPE measure points are selected following conventional strategies. For via-layer patterns, the center of each edge corresponds to a measuring point. For metal-layer patterns, the EPE measure points are evenly placed on the edges along the primary direction with $60nm$ spacing.

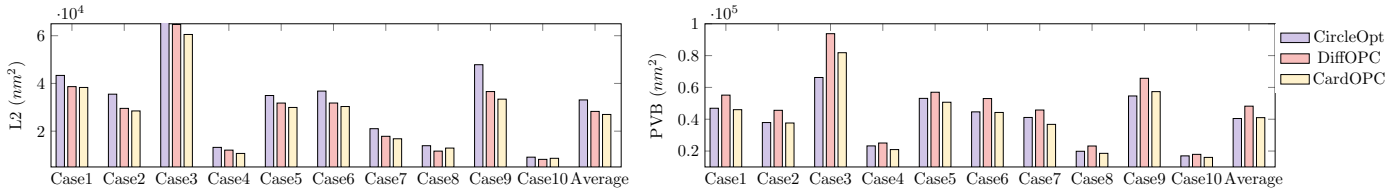
For the via layer, we dissect the polygons with $l_c = 20nm$ and $l_u = 30nm$. The moving distance is $2nm$. For the metal layer, we dissect the polygons with $l_c = 30nm$ and $l_u = 60nm$. The moving distance is $4nm$. We use 32 optimization iterations for both layers, and decay the moving distance by a factor of 0.5 at the 16-th step. A tension parameter $s = 0.6$ is adopted for the cardinal splines.

In our experiments on the via layer, we compare our CardOPC framework with existing OPC methods such as DAMO [29], RL-OPC [8], and CAMO [9], as well as the commercial tool Calibre. For the metal-layer experiments, we compare CardOPC with RL-OPC, CAMO, and Calibre. As shown in TABLE I, our framework outperforms existing methods on the via layer, achieving a 39.7% improvement in EPE and a 0.2% enhancement in PVB. Similarly, TABLE II demonstrates that our framework surpasses existing methods on the metal layer, with a 50.0% improvement in EPE and a 4.2% enhancement in PVB. Compared to these state-of-the-art (SOTA)



(a) (b) (c) (d)

Fig. 6 Examples of our curvilinear OPC results: (a) via-layer OPC; (b) metal-layer OPC; (c) large-scale OPC; (d) ILT-OPC hybrid method.



(a) (b)

Fig. 7 Comparison between our ILT-OPC hybrid approach and SOTA methods. Our method demonstrates superior performance on the L2 metric and delivers competitive PVB results. Additionally, our approach achieves an average of 1.4 EPE violations, outperforming CircleOpt (3.9) and DiffOPC (2.2).

methods, the proposed curvilinear OPC framework offers significant advantages in EPE, along with improvements in PVB. Additionally, Fig. 6(a) and Fig. 6(b) present examples of via-layer and metal-layer OPC results, respectively, which can show that our method is able to produce smooth curvilinear main patterns.

B. Large-Scale OPC Results

We conduct experiments to verify that our CardOPC can be employed in large-scale OPC applications. Following [45], we use the `gcd`, `aes`, and `dynamicnode` layouts generated by OpenROAD [47] with NanGate 45nm PDK [48]. Each layout is cropped into $30 \times 30 \mu m^2$ tiles before we perform OPC for its metal layer. In CardOPC, we dissect the polygons with $l_c = 40nm$ and $l_u = 40nm$. The moving distance is set to $8nm$. We use 10 optimization iterations, and decay the moving distance by a factor of 0.5 at the 8-th step. A tension parameter $s = 0.6$ is adopted for the cardinal splines.

We compare our method with Calibre, which uses the same configuration in [8], [9], and runs for 20 iterations. The PVB and EPE metrics are computed by Calibre. As shown in TABLE III, CardOPC surpasses Calibre, with a 6.4% improvement in EPE and a 1.9% enhancement in PVB. Our method also outperforms the previous work [45]. These results indicate that CardOPC is promising for large-scale applications, offering advantages in EPE and PVB. As an example, Fig. 6(c) shows a local region of `gcd`'s OPC result.

C. ILT-OPC Hybrid Approach Results

Following SOTA methods like [7], [12], [49], we use the lithography source and testcases from ICCAD-13 contest [35] in this experiment. We build our ILT flow based on the OpenILT framework [36] and fit its results using our ILT-OPC hybrid approach described in Section III-G. The proposed method is compared with CircleOpt [49] and DiffOPC [12], each of which has mechanisms to avoid MRC violations like CardOPC. As shown in Fig. 7, our approach excels in the L2 metric and offers competitive PVB results. In term of EPE, it achieves an average of 1.4 EPE violations, surpassing CircleOpt (3.9) and DiffOPC (2.2). Additionally, our MRC resolving strategies successfully reduce the average number of violations from 43.8 to 0. These results can validate the effectiveness of our framework. Fig. 6(d) shows an example of our results, which is similar to the corresponding ILT result but without MRC violations.

D. Ablation Study

In this experiment, we compare Bézier [31] and cardinal splines in curvilinear OPC. The primary difference in runtime between Bézier and cardinal splines lies in the control point connection process. For the `gcd` large-scale testcase, the Bézier method takes 3.6 seconds to connect the control points of 1,776 shapes at the first iteration, while our cardinal method only needs 1.9 seconds. This results in an 89% runtime overhead for the Bézier method compared to our approach. The reason for this overhead is that the Bézier method necessitates the creation of additional points when connecting two adjacent control points, which involves extra operations such as vector rotation. In terms of performance, the Bézier method obtains an EPE of 3,532 and a PVB of 34.9088, while our cardinal method achieves an EPE of 3,507 and a PVB of 34.2606. By combining runtime and performance results, we demonstrate the superiority of our curvilinear OPC framework based on cardinal splines.

V. Conclusion

In this paper, we develop a novel curvilinear OPC framework, CardOPC. After representing the mask patterns with control points, cardinal splines are employed to connect the control points. Guided by lithography simulation, the efficient correction method enables our CardOPC to achieve a 50.0% improvement in EPE and a 4.2% enhancement in PVB when compared to the SOTA OPC method. We further demonstrate the scalability of CardOPC by testing it on large-scale layout tiles. Moreover, by fitting ILT results and resolving MRC violations, our ILT-OPC hybrid method outperforms other tested algorithms. Finally, the comparison between different splines reveals that cardinal splines can offer a faster speed and better performance than Bézier splines. CardOPC shows promise as a viable ILT/OPC alternative and opens various avenues for future curvilinear OPC research, including spline types, MRC methods, MRC violation resolution strategies, and large-scale optimization.

Acknowledgements

The project is supported in part by Research Grants Council of Hong Kong SAR (No. RFS2425-4S02, No. CUHK14211824 and No. CUHK14210723), and the MIND project (MINDXZ202404).

References

[1] J. Ou, B. Yu, J.-R. Gao, D. Z. Pan, M. Preil, and A. Latypov, "Directed self-assembly based cut mask optimization for unidirectional design," in *Proc. GLSVLSI*, 2015, pp. 83–86.

[2] Y. Ma, X. Zeng, and B. Yu, "Methodologies for layout decomposition and mask optimization: A systematic review," in *Proc. VLSI-SoC*, 2017.

[3] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *Proc. ICCAD*, 2017, pp. 81–88.

[4] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," in *Proc. DAC*, 2020.

[5] H. Yang, W. Zhong, Y. Ma, H. Geng, R. Chen, W. Chen, and B. Yu, "VLSI mask optimization: From shallow to deep learning," *Integration, the VLSI Journal*, vol. 77, pp. 96–103, 2021.

[6] W. Zhao, X. Yao, Z. Yu, G. Chen, Y. Ma, B. Yu, and M. D. Wong, "AdaOPC: A self-adaptive mask optimization framework for real design patterns," in *Proc. ICCAD*, 2022.

[7] S. Zheng, H. Yang, B. Zhu, B. Yu, and M. Wong, "LithoBench: Benchmarking ai computational lithography for semiconductor manufacturing," in *Proc. NeurIPS*, 2023.

[8] X. Liang, Y. Ouyang, H. Yang, B. Yu, and Y. Ma, "RL-OPC: Mask optimization with deep reinforcement learning," *IEEE TCAD*, 2023.

[9] X. Liang, H. Yang, K. Liu, B. Yu, and Y. Ma, "CAMO: Correlation-aware mask optimization with modulated reinforcement learning," *Proc. DAC*, 2024.

[10] S. Zheng, Y. Ma, B. Yu, and M. Wong, "EMOGen: Enhancing mask optimization via pattern generation," in *Proc. DAC*, 2024.

[11] W. Zhao, X. Yao, S. Yin, Y. Bai, Z. Yu, Y. Ma, B. Yu, and M. D. Wong, "AdaOPC 2.0: Enhanced adaptive mask optimization framework for via layers," *IEEE TCAD*, 2024.

[12] G. Chen, H. Yang, H. Ren, B. Yu, and D. Z. Pan, "Differentiable edge-based OPC," in *Proc. ICCAD*, 2024.

[13] J.-S. Park, C.-H. Park, S.-U. Rhie, Y.-H. Kim, M.-H. Yoo, J.-T. Kong, H.-W. Kim, and S.-I. Yoo, "An efficient rule-based OPC approach using a DRC tool for 0.18/spl mu/m ASIC," in *Proc. ISQED*, 2000, pp. 81–85.

[14] S. O'Brien, R. Soper, S. Best, and M. Mason, "Rules based process window OPC," in *Design for Manufacturability through Design-Process Integration II*, vol. 6925, 2008, pp. 396–404.

[15] J. Wang, A. Wei, P. Verma, and W. Wilkinson, "Optimization of rule-based OPC fragmentation to improve wafer image rippling," in *European Mask and Lithography Conference*, vol. 9661, 2015, pp. 79–94.

[16] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama, "A fast process variation and pattern fidelity aware mask optimization algorithm," in *Proc. ICCAD*, 2014, pp. 238–245.

[17] J. Kuang, W.-K. Chow, and E. F. Young, "A robust approach for process variation aware mask optimization," in *Proc. DATE*, 2015, pp. 1591–1594.

[18] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. Banerjee, "Fast lithographic mask optimization considering process variation," *IEEE TCAD*, vol. 35, no. 8, pp. 1345–1357, 2016.

[19] Z. Yu, P. Liao, Y. Ma, B. Yu, and M. D. Wong, "CTM-SRAF: Continuous transmission mask-based constraint-aware sub resolution assist feature generation," *IEEE TCAD*, 2023.

[20] Y. Liu, D. Abrams, L. Pang, and A. Moore, "Inverse lithography technology principles in practice: Unintuitive patterns," in *BACUS Symposium on Photomask Technology*, vol. 5992, 2005, pp. 886–893.

[21] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. DAC*, 2014.

[22] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. DATE*, 2021, pp. 1835–1838.

[23] ——, "A GPU-enabled level-set method for mask optimization," *IEEE TCAD*, vol. 42, no. 2, pp. 594–605, 2022.

[24] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018.

[25] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE TCAD*, vol. 39, no. 10, pp. 2822–2834, 2019.

[26] B. Jiang, L. Liu, Y. Ma, H. Zhang, B. Yu, and E. F. Young, "Neural-ILT: Migrating ILT to neural networks for mask printability and complexity co-optimization," in *Proc. ICCAD*, 2020.

[27] B. Jiang, L. Liu, Y. Ma, B. Yu, and E. F. Young, "Neural-ILT 2.0: Migrating ilt to domain-specific and multitask-enabled neural network," *IEEE TCAD*, vol. 41, no. 8, pp. 2671–2684, 2021.

[28] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020.

[29] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full-chip scale," *IEEE TCAD*, vol. 41, no. 9, pp. 3118–3131, 2022.

[30] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. ICCAD*, 2021.

[31] Y. Zhang, L. Wu, J. Rao, and Y. Wang, "Curvilinear OPC mask synthesis flow," in *SPIE Photomask Technology*, vol. 12751, 2023.

[32] Y.-Y. Chen, K.-H. Chang, W.-L. Cheng, and Y.-P. Tang, "Curvilinear mask handling in opc flow," *JM3*, vol. 23, no. 1, pp. 011 203–011 203, 2024.

[33] H. H. Hopkins, "The concept of partial coherence in optics," in *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 208, no. 1093. The Royal Society London, 1951, pp. 263–277.

[34] I. Bork, A. Tritchkov, S. Shang, E. Levine, and M. Zuo, "MRC for curvilinear mask shapes," in *SPIE Photomask Technology*, vol. 11518, 2020, pp. 110–117.

[35] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. ICCAD*, 2013, pp. 271–274.

[36] S. Zheng, B. Yu, and M. Wong, "OpenILT: An open source inverse lithography technique framework," in *Proc. ASICON*, 2023.

[37] N. B. Cobb and Y. Granik, "Model-based OPC using the MEEF matrix," in *BACUS Symposium on Photomask Technology*, vol. 4889, 2002, pp. 1281–1292.

[38] J. Lei, Y. Yang, G. Lippincott, and X. Zhang, "Model-based OPC using the MEEF matrix III," in *DTCO and Computational Patterning III*, vol. 12954, 2024, pp. 343–351.

[39] M. C. Keck, W. Ziegler, R. Liebe, T. Franke, G. Ballhorn, M. Koefferlein, and J. Thiele, "Mask manufacturing rule check: how to save money in your mask shop," in *BACUS Symposium on Photomask Technology*, vol. 4186, 2001, pp. 114–118.

[40] Z. He, Y. Zuo, J. Jiang, H. Zheng, Y. Ma, and B. Yu, "OpenDRC: An efficient open-source design rule checking engine with hierarchical gpu acceleration," in *Proc. DAC*, 2023.

[41] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "STR: A simple and efficient algorithm for R-tree packing," in *Proc. ICDE*, 1997, pp. 497–506.

[42] J.-C. Yu, P. Yu, and H.-Y. Chao, "Model-based sub-resolution assist features using an inverse lithography method," in *Lithography Asia*, vol. 7140, 2008, pp. 254–264.

[43] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[44] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] S. Zheng, G. Xiao, G. Yan, M. Dong, Y. Li, H. Chen, Y. Ma, B. Yu, and M. Wong, "Model-based OPC extension in OpenILT," in *Proc. ISEDA*, 2024, pp. 568–573.

[46] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, E. F. Young, R. Karri, and S. Garg, "Adversarial perturbation attacks on ML-based CAD: A case study on CNN-based lithographic hotspot detection," *ACM TODAES*, vol. 25, no. 5, pp. 1–31, 2020.

[47] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proc. DAC*, 2019.

[48] "Nangate 45nm library," http://www.nangate.com/.

[49] X. Zhang, S. Zheng, G. Chen, B. Zhu, H. Xu, and B. Yu, "Fracturing-aware curvilinear ILT via circular E-beam mask writer," in *Proc. DAC*, 2024.

GSIM: Accelerating RTL Simulation for Large-Scale Designs

Lu Chen*†, Dingyi Zhao‡, Zihao Yu*, Ninghui Sun*† and Yungang Bao*†
*State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, China
†University of Chinese Academy of Sciences, China
‡Beijing Institute of Open Source Chip, China

***Abstract*—Register Transfer Level (RTL) simulation is widely used in design space exploration, verification, debugging, and preliminary performance evaluation for hardware design. Among various RTL simulation approaches, software simulation is the most commonly used due to its flexibility, low cost, and ease of debugging. However, the slow simulation of complex designs has become the bottleneck in design flow. In this work, we explore the sources of computation overhead of RTL simulation and conclude them into four factors. To optimize these factors, we propose several techniques at the supernode level, node level, and bit level. Finally, we implement these techniques in a novel RTL simulator GSIM. GSIM succeeds in simulating XiangShan, the state-of-the-art open-source RISC-V processor. Besides, compared to Verilator, GSIM can achieve speedup of 7.34x for booting Linux on XiangShan, and 19.94x for running CoreMark on Rocket.**

***Index Terms*—RTL simulation, optimization, large-scale**

I. Introduction

Register Transfer Level (RTL) simulation is essential in hardware design. It is widely used in various aspects, including design space exploration, verification, debugging, and preliminary performance evaluation [1] [2] [3]. It has become an indispensable step in the chip development workflow.

There are currently three main approaches to RTL simulation: software simulation, FPGA prototyping, and hardware emulation. Among them, FPGA prototyping is the fastest, but debugging tools are limited. Hardware emulation offers high speed along with good debugging ability, but it is extremely expensive. Software simulation provides 100% signal visibility, low cost and ease of debugging, but is the least efficient. Despite this drawback, software simulation remains the most commonly used method for RTL simulation [2].

With the increasing scale of modern digital circuits, the simulation speed of the software approach decreases dramatically, and becomes the bottleneck in verification [4]. Table I presents the simulation speed of processors at different scales using Verilator [5], a widely used open-source simulator. In this table, "IR node" refers to nodes in the RTL graph (including registers and logic units), and "IR edge" represents the number of connections between nodes, together indicating the complexity of each processor. As shown, the simulation speed for large-scale designs is typically limited to only a few kHz. Therefore, accelerating RTL simulation would significantly benefit the workflow for large-scale designs.

However, it is challenging to accelerate RTL simulation for large-scale designs. A software RTL simulator must simulate the signal propagation across all nodes in the circuit by computation. The heavy computation is the primary cause of the slow speed of RTL simulation. ESSENT [3] proposes an essential signal simulation approach that significantly accelerates RTL simulation. We aim to further explore opportunities to reduce computation demands.

TABLE I: Verilator (single thread) simulation speed of booting Linux on different processors. The host CPU is Intel i9-9900K. stuCore is designed by undergraduate student.

Name	Configuration	IR node	IR edge	Speed
stuCore	In-Order Single-Issue	9933	17369	$\sim$ 900KHz
Rocket [6]	In-Order Single-Issue	234807	293164	$\sim$ 30KHz
BOOM [7]	Out-of-Order Triple-Issue	571038	827619	$\sim$ 9KHz
XiangShan [8]	Out-of-Order Six-Issue	6218427	9007005	$\sim$ 0.9KHz

To identify the sources of computation, we further analyze the essential signal simulation approach. Listing 2 shows the behavior of simulating a single cycle. Each node is associated with an `active` bit. During each cycle, the simulator traverses all nodes in the graph. If the `active` bit of a node is not set (Line 7), its value can be reused and the evaluation is skipped to reduce computation. Otherwise, the value of this node needs to be evaluated (Line 3). If the value changes after evaluation, its successors will be activated (Line 4-5).

```
eval1():
  update new value
cycle():
  eval1()
  eval2()
```

Listing 1: Simulation framework in Verilator.

```
1  eval1():
2    save old value
3    update new value
4    if (old != new)
5      active[succ] = true
6  cycle():
7    if (active[1]) eval1()
8    if (active[2]) eval2()
```

Listing 2: Simulation framework in ESSENT.

From the analysis above, the sources of computation can be concluded to four factors: ❶ the accessing of `active` bits, including checking and updating, ❷ the value evaluation in each node, ❸ the total number of nodes, ❹ the activity factor, which is the ratio of active nodes to the total number of nodes.

This paper proposes a novel RTL simulator, GSIM, which aims to reduce the overhead of the four factors mentioned

above through three levels of granularity, including supernode level, node level and bit level.

❶ **At the supernode level**, one challenge is to determine which nodes should be grouped into a supernode. A supernode is a set of nodes that are activated simultaneously, with the aim of reducing the activation overhead, but it may increase the activity factor. To balance this trade-off, we propose a novel partitioning algorithm to group nodes with strong correlations. This approach reduces the activation overhead while maintaining a low activity factor.

❷ **At the node level**, GSIM employs several techniques. First, GSIM eliminates redundant nodes and optimizes expressions during value evaluation by data flow analysis. Second, we observe that although inlining the value evaluation of one node into another helps to reduce the number of nodes, it may introduce additional computation. To determine whether inlining is beneficial, GSIM constructs a cost model and selects the choice with lower cost. Finally, GSIM moves reset handling to the slow path, which reduces the number of checking reset signals by orders of magnitude.

❸ **At the bit level**, we find that bits within each node may not be accessed simultaneously. By applying data flow analysis to each bit and splitting nodes according to their accessing patterns, GSIM can further reduce the activity factor.

GSIM successfully simulates XiangShan [8], the state-of-the-art open-source RISC-V processor. When booting Linux on XiangShan, GSIM achieves a 7.34x speedup compared to Verilator. When running CoreMark [9] on Rocket [6], GSIM reaches 19.94x of Verilator, outperforming the best between ESSENT [3] and Arcillator [10] by 2.52x.

II. Background and Analysis

A. RTL Simulation

To perform RTL simulation, the input design is represented as a directed graph. In this graph, each node corresponds to a register or logic unit, and each edge represents the propagation of signals between nodes. During simulation, the circuit's behavior is modeled as evaluation of signal values that propagate between nodes along the edges. The work of determining the evaluation order of nodes is called scheduling.

RTL simulation can be categorized into two types based on the scheduling policy [11]. In event-driven simulation, nodes are dynamically scheduled at runtime, with signal changes propagated as events. When the value of a node changes, an event is generated and sent to all its successors. This approach can flexibly model arbitrary delays, and is widely used in post-synthesis gate-level simulation. However, frequent scheduling in complex designs incurs high overhead. Commercial simulators like VCS [12] employ the event-driven simulation model.

In contrast, for full-cycle simulation, nodes are statically scheduled at compile time and evaluated in a fixed order at runtime. This eliminates the overhead of runtime scheduling. Although the minimum granularity of simulation is one clock cycle, it still suffices to perform functional verification for large-scale designs like processors. The open-source simulator Verilator [5] employs full-cycle simulation. Generally, full-cycle simulation is faster than event-driven simulation [13].

In a basic implementation of full-cycle simulation, to build a directed acyclic graph (DAG), registers are often split into two nodes (one for reading and one for writing) to break cycles in the graph. Then a topological sort is performed on this graph. In each cycle, all nodes are evaluated in the topological order, as shown in Listing 1. Verilator employs this approach.

To further improve the efficiency of full-cycle simulation, ESSENT [3] introduces an essential signal simulation approach. This approach is effective, because only a few nodes have their value changed in a cycle. Listing 2 shows the implementation. Each node is associated with an `active` bit. During a cycle, all `active` bits are examined. A node is evaluated only when its `active` bit is set. If the value of the node changes after evaluation, the active bits of its successors will be set to indicate the need of re-evaluation.

B. Modeling and Analysis

Inspired by ESSENT, we further explore optimization potential by identifying the sources of computation. According to the simulation framework in Listing 2, the simulation overhead per cycle T can be modeled as $T = ((E + A_{succ}) * af + A_{exam}) * N$, where E is the computation overhead of value evaluation in a node, A_{succ} is the overhead of activating successors, af is the activity factor, A_{exam} is the overhead of examining the `active` bit, and N is the number of nodes.

E represents the computation required to evaluate the value of a node based on its predecessors. It contains various operations, including arithmetic, shifting, bitwise, and conditional operations. A_{succ} and A_{exam} are introduced by the essential signal simulation approach. Each node requires an additional examination before evaluation and an activation step after evaluation. The former adds a branch instruction, while the latter involves a branch and several memory access instructions. N and af are determined by the circuit design. They together indicate the number of nodes with the `active` bit set in a cycle, which require evaluation. As the complexity of the circuit increases, N may grow a lot, but af remains low in most designs. According to our evaluation, af is about 4.61% when running CoreMark on XiangShan.

Although E, A_{succ}, A_{exam} and af seem negligible, their impact on T is amplified by a large N. Therefore, reducing N can significantly improve simulation performance. Additionally, small improvements on the other factors can still yield substantial performance gains. For example, during the simulation of Xiangshan, 82.26% of all executed branch instructions are dedicated to examining the `active` bits. Therefore, it is important to further optimize these factors.

III. GSIM Design

To accelerate RTL simulation, GSIM employs a series of optimization techniques targeting the factors above at three levels: super-node level, node level, and bit level. Note that some techniques may improve one factor while degrading another. In such cases, GSIM balances the trade-offs among these factors with corresponding strategies.

A. Supernode Level Optimization

Checking the `active` bit of each node separately will incur high overhead for A_{exam} (Listing 2). To reduce A_{exam}, nodes are typically grouped into supernodes [3]. Each supernode is associated with a single `active` bit. Nodes inside a supernode share an `active` bit and are evaluated together (Listing 3). As a result, activating any node within a supernode leads to the evaluation of all nodes in it. We define the size of a supernode to be the number of nodes inside it. A supernode with larger size helps to reduce A_{exam}, but it may increase af.

```
eval_super1():
  eval1()
  eval2()
cycle():
  if (active[1])
    eval_super1()
```

Listing 3: Supernode level activation.

```
if (active[7:0] != 0)
  if (active[1])
    eval_super1()
  if (active[2])
    eval_super2()
  ...
```

Listing 4: Check multiple bits with a single condition.

To keep af low, GSIM tries to group nodes that are likely to be activated simultaneously into the same supernode. We observe that nodes in close proximity in the RTL graph often exhibit strong correlations and tend to be activated together. To find such nodes, a traditional graph partitioning algorithm may be a solution, such as Kernighan's Algorithm [14]. However, the goal of the traditional algorithm usually conflicts with building supernodes. This is because it tries to minimize the number of cut edges between partitions. It tends to divide nodes connected with a small number of edges into different partitions (the left of Figure 1). However, such nodes may be activated together during simulation, and they should be grouped into the same supernodes (the right of Figure 1).

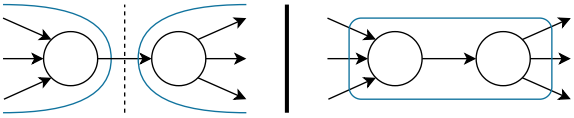


Fig. 1: Traditional partitioning algorithms fail to group nodes connected with few edges into the same partitions.

To address this challenge, GSIM employs an enhanced version of Kernighan's Algorithm. Specifically, the enhanced algorithm initially groups nodes with strong correlations based on certain rules, protecting them from being divided into different supernodes in the subsequent steps. These rules are derived from the following observations: ❶ A node with an out-degree of 1 is typically activated along with its successor. ❷ A node with an in-degree of 1 is usually activated when its predecessor is activated. ❸ Siblings with the same predecessors are always activated simultaneously. Subsequently, GSIM adopts the original Kernighan's Algorithm to construct supernodes. The maximum size of a supernode can be controlled by a parameter in the algorithm.

Additionally, to take advantage of the low af, GSIM speculatively assumes multiple `active` bits are clear, and uses a single condition to examine them on the fast path (Line 1 in Listing 4). If the condition holds, GSIM can skip the individual examination of each `active` bit to reduce A_{exam}.

B. Node Level Optimization

Redundant node elimination. To reduce N, we eliminate the following types of redundant nodes (Figure 2) by applying data flow analysis. ❶ **Alias Nodes** are duplicate nodes representing the same signal. ❷ **Dead Nodes** are nodes not used by other nodes. ❸ **Shorted Nodes** are nodes not selected due to another signal. ❹ **Unused Registers** are registers not used by other nodes, but they may be self-updated.

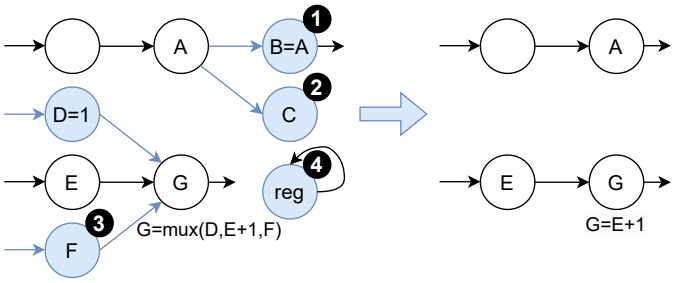


Fig. 2: Examples of eliminating redundant nodes.

Node inline and extraction. For non-redundant nodes, GSIM considers whether to inline the evaluation of one node into its successors. This is a trade-off between N and E. Extracting common parts of evaluation can reduce E by reusing the value stored in an extra intermediate node. For example, in the left part of Figure 3, node B stores the result of f(A), and its successors (C and D) can directly use the value in B without re-evaluating f(A). Conversely, reducing N may increase E. As shown in the right part of Figure 3, nodes C and D must re-evaluate f(A) without the intermediate node.

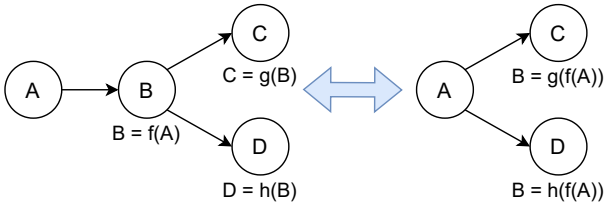


Fig. 3: Transform between node inline and extraction.

To address the trade-off above, GSIM models the cost of each decision and chooses the best one. Specifically, GSIM estimates the evaluation overhead of f(A), $cost(\text{f(A)})$, and the overhead of introducing a new node, $cost_{node}$, in terms of the number of operators involved, and the reference count of f(A), $\#\text{f(A)}$. If $cost(\text{f(A)}) \times \#\text{f(A)} > cost(\text{f(A)}) + cost_{node}$, f(A) is extracted as an extra node to reduce E. Otherwise, f(A) is inlined to reduce N.

Activation overhead optimization. To optimize A_{succ}, ESSENT replaces the condition checking (Line 4 and 5 in Listing 2) with logical operations to relieve the pressure of the host branch predictor [3]. However, we find that it may cause

performance degradation when there are too many successors to activate, since this technique introduces extra operations for each successor to activate. To handle this trade-off, GSIM models the cost of each decision and chooses the optimal one, which is similar to the approach mentioned above.

Reset handling optimization. During the evaluation of each register with reset port, the reset signal must be checked every cycle to determine the value to be written (Listing 5). However, the number of reset signals is usually small, and they are hardly valid during simulation. Based on this observation, GSIM first speculatively assumes reset does not happen, and updates the register with the value evaluated based on its predecessors (Listing 6). This can remove the checking of reset signals from the fast path during evaluation, which helps to reduce E. After that, the reset signals are checked at the end of every cycle. In this way, GSIM can reduce the number of checking reset signals by orders of magnitude, from the number of registers with reset port to the number of reset signals in the design.

```
eval1():
  reg1 = reset ? 0 : new_val
  ...
cycle():
  if (active[1]) ...
  ...
```

Listing 5: Before reset optimization.

```
eval1():
  reg1 = new_val
  ...
check_reset():
  if (reset)
    reg1 = 0
    reg2 = init_val2
    ...
    activate_all_succ()
```

Listing 6: After reset optimization.

Expression simplification. GSIM employs various techniques to optimize expressions during evaluation, such as constant propagation and recognition of complex patterns. For example, it is common to generate a one-hot signal and check each bit of it in RTL design. This may result in two nodes: one to evaluate `B = 1 << A`, and the other to evaluate `C = bits(B, k, k)`, where `k` is a constant. GSIM detects this pattern and optimizes it to `C = (A == k)`.

C. Bit Level Optimization

An activation is unnecessary if the value of the node remains the same after evaluation. We find that in many long signals, only certain bits change each cycle. If a successor only uses the unchanged bits, activating it will be unnecessary. As shown in the left part of Figure 4, if node A changes while B and C are inactive, both D and E will change, leading to the activation of F and G. However, the value of G remains unchanged, since it does not depend on A. Therefore, the activation of G is unnecessary. According to our statistics, among nodes with multiple bits in XiangShan, 23.7% of them are composed of concatenations from other nodes, and 23.2% of the references to these nodes only access a subset of their bits.

To address this issue, GSIM adopts a bit-level node splitting policy. Specifically, GSIM applies data flow analysis to each bit. For a node P_0, if there exists a bit set S in P_0 and a path $P_0P_1 \dots P_n$, where P_n does not depend on $\overline{S}$ (the complementary bit set of S in P_0), GSIM can split all nodes on the path according to S and $\overline{S}$. In this way, we can remove the unnecessary activation of node P_n when only bits in $\overline{S}$ change, thereby reducing af. As shown in the right part of Figure 4, node A will no longer trigger the activation of G.

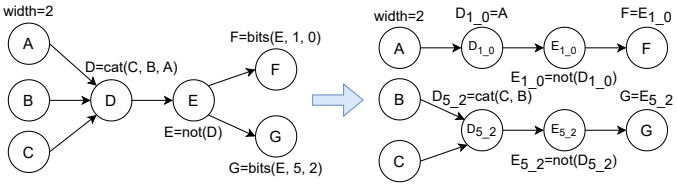


Fig. 4: Example of splitting nodes at the bit level.

Note that although splitting nodes may increase N, we can apply node level optimization techniques mentioned above to the generated nodes.

D. Implementation Details

GSIM accepts circuits in Firrtl [15], an intermediate representation for hardware that various HDL can be converted into. GSIM includes a Firrtl parser that converts the input design into an abstract syntax tree (AST) and further transforms it into a graph. Most optimizations are performed on this graph. After optimization, GSIM emits C++ simulation code. Additionally, GSIM allows users to adjust the maximum size of a supernode from the command line.

IV. Evaluation

A. Evaluation Setup

We compare GSIM with the following simulators. ❶ **Verilator** [5] is the most widely used open-source Verilog simulator. We use version v5.026 with `-O3` flag to generate C++ files. It also supports multi-threaded simulation. ❷ **Arcilator** [10] is an MLIR [16] simulator developed as part of the CIRCT [17] project. ❸ **ESSENT** is a single-threaded Firrtl [15] simulator, with `-O3` flag to generate C++ files.

We evaluate the simulators on processors in Table I. Table II shows their versions. All processors are mainly developed in Chisel [18]. For a few Verilog modules, we replace them with Chisel implementations. As shown in Figure 5, each design is first converted into Chirrtl IR, and then translated into Firrtl, Verilog, and MLIR files using the FIRRTL [15] compiler or CIRCT [17] compiler. ESSENT and GSIM accept Firrtl as input, Verilator accepts Verilog, and Arcilator accepts MLIR.

TABLE II: The version of the processors.

	commit hash (truncated)	commit date
Rocket	e3773366a5c473b6b45107f037e3130f4	Sep 26, 2023
BOOM	9459af0c1f6847f8411622dac770ac78f	Oct 17, 2023
XiangShan	a42a7ffe5e4be903c75d53cf243761ce7	Mar 1, 2024

We choose CoreMark [9] and Linux as the software workloads. The former exhibits hot spots, while the latter does

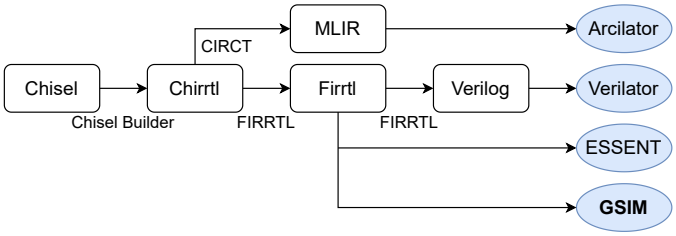


Fig. 5: The input of different simulators.

not. The host configuration is Intel 8-core i9-9900K CPU at 3.60GHz with 64GB of RAM. The host OS is Debian 12.

B. Overall Performance

Figure 6 shows the performance of different simulators on various processors and software workloads. Since Verilator is the most widely used open-source RTL simulator and successfully simulates all selected processors, we use Verilator as the baseline and normalize the other results to it.

In terms of correctness, aside from Verilator, GSIM is the only simulator that can successfully simulate XiangShan. ESSENT and Arcilator fail to simulate some of the designs due to errors during emitting C++ files or wrong simulation results. For performance, GSIM achieves speedup of 7.34x for booting Linux on XiangShan, and 19.94x for running CoreMark on Rocket. We also evaluate Verilator with multi-threading. GSIM significantly outperforms multi-threading Verilator on most of the designs.

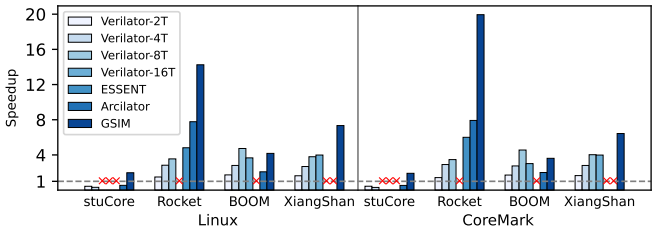


Fig. 6: Overall performance. The speedup is normalized to the single thread version of Verilator. '2T' means simulating with 2 threads. '×' means failing to perform the simulation.

C. Running SPEC CPU2006

In CPU simulation, checkpoints have long been used to accelerate performance evaluation [8] [19]. We sample SPEC CPU2006 [20] using SimPoint [21] and extract 40M-instruction segments to create checkpoints. The selected benchmarks cover various types of workloads, including streaming and irregular memory access, integer and floating-point computation, branch-intensive operations, and instruction-cache-intensive [22]. We run these checkpoints on XiangShan using both Verilator and GSIM. As shown in Figure 7, GSIM is 3.72x faster than single-threaded Verilator across all checkpoints on average, and achieves an average 1.18x of 8-threaded Verilator.

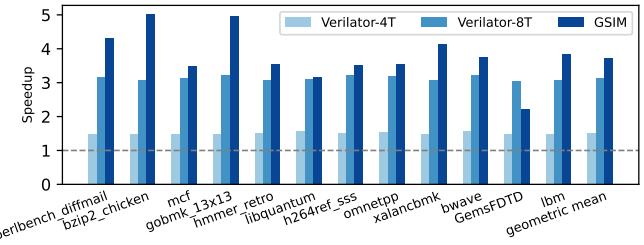


Fig. 7: SPEC CPU2006 performance on XiangShan. The speedup is normalized to the single thread version of Verilator.

D. Performance Breakdown

To evaluate the contribution of each optimization technique adopted in GSIM, we apply all techniques incrementally, starting with a baseline with no optimizations. In this way, we get a series of performance data $P_0, P_1, \ldots, P_n$. In Figure 8, the height of the bar for the ith technique is calculated as $\log_{10}(P_i/P_{i-1})$. As shown in the figure, introducing supernode significantly improves performance for all designs. Splitting nodes at bit level delivers substantial improvement for BOOM and notable gains for XiangShan, but has little impact on stuCore and Rocket.

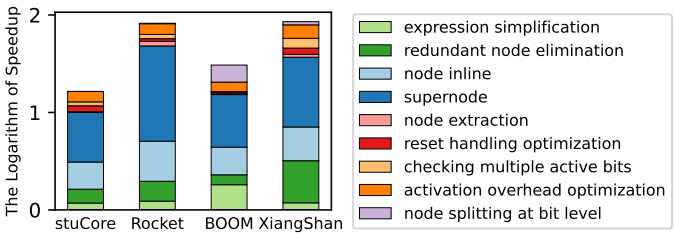


Fig. 8: Performance breakdown for each technique.

E. Selecting the Maximum Size of Supernode

We also investigated the relationship between the maximum size of supernode and the simulation performance. As mentioned above, a larger size of supernode reduces A_{exam} but increases af. The optimal size depends on the circuit design. To find the optimal size, we try different sizes with all other optimization techniques enabled. For the designs we select, the optimal size ranges from 20 to 50, as shown in Figure 9.

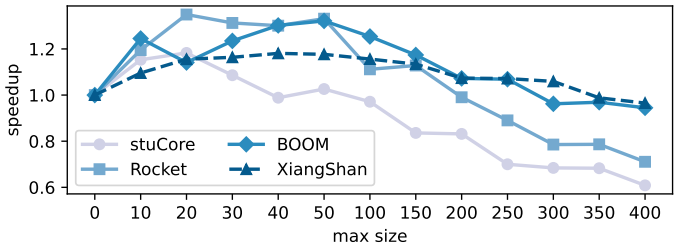


Fig. 9: Performance with respect to maximum supernode size.

F. Different Algorithms to Build Supernode

We compare the enhanced partitioning algorithm used in GSIM against the original Kernighan's Algorithm and ESSENT's partitioning algorithm (noted as MFFC-based). These algorithms are implemented on GSIM with all other optimization techniques disabled to show their effectiveness. They are evaluated by running CoreMark on BOOM under their own optimal parameters. We measure the number of supernodes, the number of times to active a successor, and the number of active node. These metrics can reflect A_{exam}, A_{succ} and E respectively. As shown in Table III, our algorithm performs the best in terms of the simulation speed, by achieving a better balance among the three factors above.

TABLE III: Comparison of different partitioning algorithms by running CoreMark on BOOM.

	partition time (s)	supernode	activation times	active node	speed (Hz)
None	–	536721	64044	42398	913
Kernighan	1.3	21769	13104	93327	4522
MFFC-based	88.9	36478	7310	124202	6802
GSIM	1.6	14261	8169	114255	8184

G. Resource Usage

For the emission time, we measure the time required to emit simulation files. As shown in Table IV, although GSIM outperforms Verilator, GSIM still delivers comparable emission time to Verilator. ESSENT requires more time to generate C++ files. One reason may be the slow execution of Scala programs. Arcilator requires more than 100GB memory to emit code for BOOM, which leads to accessing swap with a very long emission time. It even fails to emit code for XiangShan due to out-of-memory.

For the code size, we measure the size of the `.text` section of the binary reported by `readelf`. As shown in Table IV, GSIM emits the least amount of code among all simulators, as it employs more optimization techniques.

For the data size, we use `sizeof` to obtain the size of the top-level design class in C++ files, whose members contain all variables used for simulation, and we exclude the array used to simulate the main memory (128MB). As shown in Table IV, GSIM generates slightly larger data than Verilator.

V. Related Work

Verilator [5] is a widely used open-source, full-cycle RTL simulator. It supports multi-threading by partitioning the RTL design. Arcilator [10], an RTL simulator based on CIRCT, applies optimizations at various IR levels. Both of them evaluate all signals in the circuit every cycle and improve simulation speed through various expression optimization techniques. In contrast, GSIM skips the evaluation of a signal when it is inactive. Khronos [2] employs cross-cycle optimizations to reduce memory access overhead during simulation, which performs well in circuits with many pipeline registers. However, GSIM targets more general designs. ESSENT [3] leverages the low activity factor in circuits, partitioning the design and filtering out inactive regions to efficiently reduce the computation overhead. GSIM further explores the potential of accelerating RTL simulation. It reduces the computation overhead from four factors with optimization techniques across three levels.

TABLE IV: Comparison of resources across various designs. Instances labeled with '*' mean failing to run.

Design	Simulator	Emission Time(s)	Code Size(B)	Data Size(B)
stuCore	Verilator	1.1	394K	15K
	*ESSENT	7.7	659K	21K
	Arcilator	0.4	131K	13K
	GSIM	0.4	133K	16K
Rocket	Verilator	5.8	2.9M	62K
	ESSENT	37.8	1.6M	107K
	Arcilator	2.9	305K	49K
	GSIM	9.8	1.4M	72K
BOOM	Verilator	22.9	7.7M	954K
	*ESSENT	-	-	-
	Arcilator	4194.6	2.8M	799K
	GSIM	31.3	4.4M	976K
XiangShan	Verilator	374.6	40.7M	22.2M
	*ESSENT	-	-	-
	*Arcilator	-	-	-
	GSIM	389.1	25.4M	22.3M

RepCut [1] is a multi-threaded RTL simulator based on ESSENT. It reduces inter-thread synchronization overhead by introducing redundant computation. [13] proposes a coarse-grained circuit deduplication method to improve throughput for a batch of simulation tasks. GSIM focuses on improving the performance of single-threaded simulation under a single task. These works are orthogonal to GSIM and provide directions for future work.

VI. Conclusion

This paper proposes GSIM, a novel RTL simulator based on the essential signal simulation approach. GSIM employs a series of optimization techniques at the supernode level, node level, and bit level. GSIM successfully simulates XiangShan, the state-of-the-art open-source RISC-V processor. Compared to Verilator, GSIM achieves a 7.34x speedup when booting Linux on XiangShan, and a 19.94x speedup when running CoreMark on Rocket.

Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable feedback and comments. This work is supported by the National Key R&D Program of China (2022YFB4500403).

References

[1] H. Wang and S. Beamer, "Repcut: Superlinear parallel rtl simulation with replication-aided partitioning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 572–585. [Online]. Available: https://doi.org/10.1145/3582016.3582034

[2] K. Zhou, Y. Liang, Y. Lin, R. Wang, and R. Huang, "Khronos: Fusing memory access for improved hardware rtl simulation," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 180–193.

[3] S. Beamer and D. Donofrio, "Efficiently exploiting low activity factors to accelerate rtl simulation," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[4] L. Lavagno, I. Markov, G. Martin, and L. Scheffer, *Electronic Design Automation for IC System Design, Verification, and Testing*. CRC Press, 2017. [Online]. Available: https://books.google.com/books?id=UeobDAAAQBAJ

[5] W. Snyder, "Verilator," https://veripool.org/verilator/.

[6] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[7] C. Celio, D. A. Patterson, and K. Asanović, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," Tech. Rep. UCB/EECS-2015-167, Jun 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[8] Y. Xu, Z. Yu, D. Tang, G. Chen, L. Chen, L. Gou, Y. Jin, Q. Li, X. Li, Z. Li, J. Lin, T. Liu, Z. Liu, J. Tan, H. Wang, H. Wang, K. Wang, C. Zhang, F. Zhang, L. Zhang, Z. Zhang, Y. Zhao, Y. Zhou, Y. Zhou, J. Zou, Y. Cai, D. Huan, Z. Li, J. Zhao, Z. Chen, W. He, Q. Quan, X. Liu, S. Wang, K. Shi, N. Sun, and Y. Bao, "Towards developing high performance risc-v processors using agile methodology," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1178–1199.

[9] "Coremark," https://www.eembc.org/coremark/.

[10] Z. Y. B. H. T. G. Martin Erhart, Fabian Schuiki, "Arcilator: Fast and cycle-accurate hardware simulation in circt," https://llvm.org/devmtg/2023-10/slides/techtalks/Erhart-Arcilator-FastAndCycleAccurateHardwareSimulationInCIRCT.pdf.

[11] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[12] "Vcs: Functional verification solution," https://www.synopsys.com/verification/simulation/vcs.html.

[13] H. Wang, T. Nijssen, and S. Beamer, "Don't repeat yourself! coarse-grained circuit deduplication to accelerate rtl simulation," 2024.

[14] B. W. Kernighan, "Optimal sequential partitions of graphs," *Journal of the ACM (JACM)*, vol. 18, no. 1, pp. 34–40, 1971.

[15] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson *et al.*, "Reusability is firrtl ground: Hardware construction languages, compiler frameworks, and transformations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 209–216.

[16] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 2–14.

[17] "Circt: Circuit ir compilers and tools," https://circt.llvm.org/.

[18] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[19] "Gem5: Checkpoints," https://www.gem5.org/documentation/general_docs/checkpoints/.

[20] "Spec cpu® 2006," https://www.spec.org/cpu2006/.

[21] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 45–57, 2002.

[22] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the spec cpu2006 benchmark suite," in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 412–423.

X-SAT: An Efficient Circuit-Based SAT Solver

Yuhang Qian[1, 2], Zhihan Chen[1, 2], Xindi Zhang[1, 2], Shaowei Cai[1, 2, *]
[1]*Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences*
[2]*School of Computer Science and Technology, University of Chinese Academy of Sciences*
{qianyh, chenzh, zhangxd, caisw}@ios.ac.cn

***Abstract*—In modern digital circuit design, verifying the equivalence of arithmetic circuits is a significant and challenging task. This paper introduces a new circuit solver based on the Conflict-Driven Clause Learning (CDCL) algorithm, which integrates structural elimination techniques to reduce the number of variables and clauses while maintaining the circuit structure. Additionally, branching heuristics have been enhanced specifically for the structure of arithmetic circuits. Experimental results demonstrate that X-SAT significantly outperforms best previous circuit solver could be found on all benchmarks. Further, X-SAT performs better than the state-of-the-art CNF-based SAT solvers on complex arithmetic circuits, underscoring its significant potential in the field of circuit design verification.**

***Index Terms*—Boolean satisfiability, Equivalence checking, Circuit-based SAT, EDA.**

I. Introduction

Boolean Satisfiability (SAT) is a fundamental problem in computer science with wide-ranging applications, especially in electronic design automation (EDA). Logic equivalence checking (LEC) [1], logic synthesis [2], bounded model checking, and automatic test pattern generation (ATPG) [3] commonly rely on SAT solvers. Typically, circuits are encoded into conjunctive normal form (CNF) for SAT solving.

Modern CNF solvers [4]–[6] are based on the Conflict Driven Clause Learning (CDCL) method, and equipped with numerous powerful and efficient techniques, including the two-literal watching scheme [7] for efficient Boolean Constraint Propagation (BCP), Bounded Variable Elimination (BVE) [8], variable selection heuristics such as VSIDS [7], learnt clause reduction, among others. However, these strategies and optimizations are designed based on the CNF data structure and do not take into account any information from the circuit.

When solving problems arising from EDA tasks, the transformation to CNF discards the structural information of the original circuit, limiting optimization potential. To address this issue, QuteSAT [9] improves the BCP process, enabling propagation directly on circuits and supporting both simple and complex gates. Additionally, the solver supports a circuit-structure-based branching strategy called **J-frontier**. Zhang et al. [2] introduced a SAT solver tailored for logic synthesis, which was built upon the CNF-based solver Glucose [5]. Besides performing BCP in circuit format, it also employs improved data structures to maintain the J-frontier branching heuristic. Hu et al. [10] used logic synthesis techniques for circuit rewriting before SAT solving.

*Corresponding author

This paper presents a novel circuit SAT solver that is lightweight and efficient, with the focus on arithmetic circuits. Our contributions are summarized as follows:

- We propose a structural elimination technique to simplify the circuits before SAT solving. Specifically, we reduce variables by identifying XOR gates and synthesizing non-XOR gates into multi-input gates. This technique is effective for both arithmetic and non-arithmetic circuits. It also improves the search efficiency of the CDCL framework through a more compact circuit representation and distinguishes our method from traditional solvers that operate directly on the original circuit structure.
- We design a new branching heuristic method based on VSIDS [7] called **XVSIDS**. During conflict analysis, we increase the branching weights of neighboring XOR gates for the analyzed variables, enhancing the heuristic's locality. It has achieved good performance on arithmetic circuits, more robust than the J-Frontier heuristic.
- We develop a circuit SAT solver based on the CDCL framework, which integrates these two techniques. Our circuit SAT solver has significantly outperformed the best previous circuit solver we could find on both arithmetic and non-arithmetic circuits. Moreover, our solver surpasses the state-of-the-art CNF SAT solver Kissat [4] on arithmetic circuits, noting that Kissat and its variants dominates the main tracks of recent SAT competitions from 2020. This is the first time that a circuit SAT solver is compared to state-of-the-art CNF SAT solver (previous works usually compared to MiniSAT [6] or Glucose [5]) and achieves better results.

The rest of the paper is organized as follows: Section II introduces the background on SAT solving, AIG graph, circuit-based SAT solving and CEC problem. Section III presents the main methods used in the design of our solver, including the motivating example, solving framework and the implementation of structural elimination and the XVSIDS branching strategy. Section IV presents experimental results while Section V concludes the paper.

II. Background

A. Boolean Satisfiability

The Boolean Satisfiability (SAT) asks whether there is an assignment to the variables of a Boolean formula in Conjunctive Normal Form (CNF) that makes it true. A literal is either

Algorithm 1: Basic framework of CDCL Algorithm

```
Input: CNF formula F
Output: SAT or UNSAT
α = {};
dl ← 0;
if BooleanUnitPropagation( F, α ) == CONFLICT
 then
    return UNSAT ;
end
while ∃ unassigned variables do
    (x, v) ← SelectBranchVar( F, α ) ;
    dl ← dl + 1 ;
    α ← α ∪ {(x, v)} ;
    if UnitPropagation( F, α ) == CONFLICT then
        bl ← ConflictAnalysis( F, α ) ;
        if bl < 0 then
            return UNSAT ;
        end
        else
            Backtrack(F, α, bl) ;
            dl ← bl ;
        end
    end
end
return SAT ;
```

a Boolean variable or its negation. A formula is represented as a conjunction of clauses, where each clause is a disjunction of literals:

$$\phi = (C_1 \wedge C_2 \wedge \cdots \wedge C_m), \quad C_i = (l_{i1} \vee l_{i2} \vee \dots)$$

The goal is to find an assignment $\mathbf{x}$ such that $\phi(\mathbf{x}) = 1$ or prove none exists. SAT is NP-complete, making it computationally challenging for large instances, while it remains a fundamental problem in hardware verification.

B. SAT Solving Framework

Modern SAT solvers, such as MiniSAT, Glucose, and Kissat are based on the Conflict-Driven Clause Learning (CDCL) framework [11]. Key techniques in CDCL-based SAT solvers, including efficient Boolean Constraint Propagation (BCP), conflict analysis, clause learning, rephasing, restarts, learnt clause reduction, and Bounded Variable Elimination (BVE), have significantly enhanced their performance in practical applications.

The pseudo-code in Algorithm 1 represents the basic framework of the Conflict-Driven Clause Learning (CDCL) algorithm. The key functions include **BooleanUnitPropagation(**F**,** α**)**, which propagates unit clauses in the formula F based on the partial assignment α and returns 'CONFLICT' if a contradiction occurs; **SelectBranchVar(**F**,** α**)**, which selects an unassigned variable x and assigns it a value v; **ConflictAnalysis(**F**,** α**)**, which analyzes conflicts and determines the backtrack level bl; and **Backtrack(**F**,** α**,** bl**)**, which reverts the assignment to the backtrack level bl. The algorithm starts with an empty partial assignment and decision level, iteratively selects variables, propagates unit clauses, and uses conflict analysis and backtracking to resolve conflicts. It terminates when a satisfiable solution is found (SAT) or the formula is unsatisfiable (UNSAT).

C. And-Inverter Graph

And-Inverter Graph (AIG) is a representation of Boolean logic circuits that uses AND gates and inverters (NOT gates) as the basic components. AIGs are a popular choice for circuit representations because they provide an efficient way to model and manipulate Boolean functions. AIGs are particularly useful for circuit verification, as they allow for efficient manipulation of logic functions and can be easily transformed into Conjunctive Normal Form (CNF) for SAT solvers.

D. Circuit-based SAT Solving

Circuit-based SAT solving shares the same CDCL framework as SAT solvers for CNF formulas, with most techniques being interchangeable. The primary differences lie in how decisions and propagations are performed. In circuit solvers, decisions are made directly on wires in the circuit structure, and assignments are propagated through the circuit, rather than through the CNF representation as in conventional SAT solvers. This approach eliminates the overhead of converting circuits into CNF, making circuit solvers more efficient for scenarios requiring repeated invocations. Additionally, circuit solvers can employ structure-aware branching heuristics, such as the J-frontier heuristic [9], to improve decision-making and enhance solving efficiency.

E. CEC Problem

Combinational Equivalence Checking (CEC) is the problem of determining whether two Boolean circuits or logic functions are functionally equivalent. It is a fundamental task in EDA with applications in formal verification, circuit optimization, and hardware testing. To perform CEC, the two circuits are first combined into a *miter circuit*, where the inputs of both circuits are shared, and their outputs are XORed. The output of the miter circuit is used to detect discrepancies: if it is always 0 for all input combinations, the circuits are equivalent; otherwise, they are not.

Once the miter circuit is constructed, the problem is encoded as a SAT instance and solved using a SAT solver. A key optimization is *SAT-Sweeping* [12], [13], which iteratively calls a SAT solver to verify the equivalence of potentially equivalent internal nodes, and merges equivalent nodes to reduce complexity.

III. Proposed Method

A. Motivating Example

Although AIG provides a convenient expression for circuits, it is usually too basic for complex circuits such as arithmetic

circuits. By identifying the structure and re-rewrite the circuits using more compact gates, we can significantly reduce the number of variables in the circuits, making the circuit SAT solving faster.

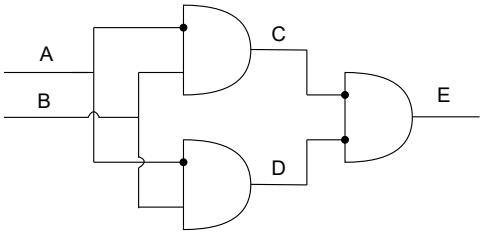


Fig. 1. An example of a small subcircuit in AIG format

As illustrated in Figure 1, the given circuit demonstrates a scenario where the value of E is determined by inputs A and B. In an And-Inverter Graph (AIG), due to its limited expressive power, additional variables C and D must be introduced to construct the complete circuit. However, an alternative approach is to directly construct a 2-input LUT gate with A and B as inputs and E as the output. More generally, by merging the circuit into multi-input LUT gates, it is possible to significantly reduce the number of variables while maintaining the overall structure of the circuit. This observation inspired the development of the **Structural Elimination** method, which will be discussed in detail in the following sections.

Furthermore, XOR gates exhibit unique properties that can be leveraged in circuit analysis. Building upon the VSIDS heuristic, we introduce specific enhancements tailored for XOR gates, leading to a new heuristic named **XVSIDS**.

B. Overview of Our Circuit-based SAT Method

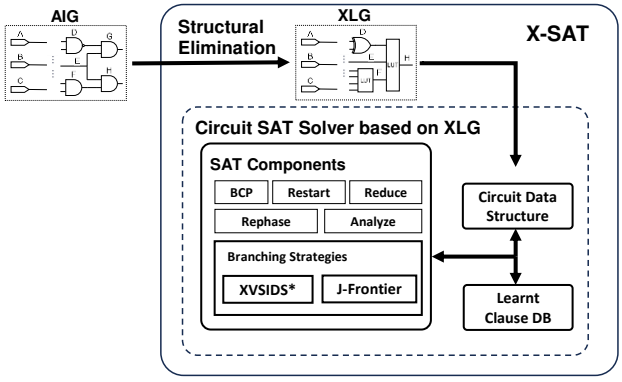


Fig. 2. Main framework of X-SAT

We develop an efficient circuit-based SAT solver. The solver is composed of two parts, a rewriting procedure and a CDCL-based solver.

In contrast to other circuit solvers that operate directly on AIG circuits, our method transforms AIG circuits into XLG circuits, preserving the circuit structure while reducing variables. This lightweight approach is particularly advantageous for iterative and frequent invocations, making our solver well-suited for SAT-sweep processes and model checking. It is particularly effective for arithmetic circuits.

For SAT solving, we introduce the XVSIDS branching strategy, adapted from VSIDS and optimized for the compact XLG circuit representation. For comprehensive benchmarking, we also incorporate a j-frontier branching heuristic, similar to those utilized in other circuit solvers, to enable direct and meaningful comparisons.

C. Circuit Structural Elimination

Our method first identifies the binary XOR relationships in the AIG and then rewrites the original AIG circuit by incorporating XOR gates to express the circuit. This type of circuit is referred to as Xor-And-Inverter Graph (XAG).

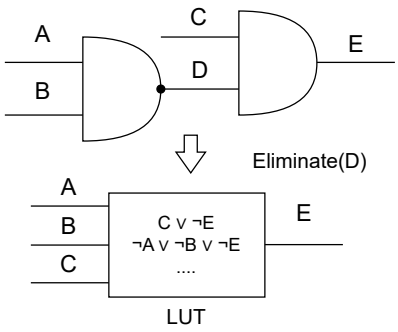


Fig. 3. An example of the elimination process

In addition to XOR gates, the non-XOR parts of the circuit could be further simplified. We merge binary gates into multi-input gates to simplify the circuit. The logical relationships between the inputs and outputs of multi-input gates can be expressed using CNF formulas, which have greater expressive power and are better suited for the CDCL solving framework. As shown in Figure 3, we eliminate the variable D and express the relationship between the inputs and the output of the introduced three-input gate using a CNF formula. Here we borrow a concept from FPGA and refer to the merged multi-input gates as LUTs. Such multi-input gates, represented in CNF, can express any relationship between inputs and outputs, rather than being restricted to well-known logical circuit gates such as multi-input AND, multi-input XOR, or MUX. Ultimately, we obtain a graph that includes both LUTs and XOR gates, which we refer to as the the Xor-Lut-Inverter Graph (XLG).

The selection of variables for elimination and the order in which they are eliminated have significant impact to the algorithm's performance. Our algorithm relies on a concept called *elimination score*, which measures the change on the number of literals, noting that some literals would be deleted while new literals are introduced when a variable is eliminated.

Algorithm 2: Circuit Structural Elimination

Input: AIG format circuit A
Output: XLG format circuit L

```
1  xag ← identify xor gates and rewrite A;
2  heap ← a max heap sorted by the elimination score
     associated with each variable;
3  L ← xag;
4  while heap is not empty do
5      gate ← gate with max elimination score in heap
6      if gate is not a single fan-out gate then
7          continue;
8      end
9      fanout_gate ← single fanout gate of g;
10     if gate or fanout_gate is XOR gate then
11         continue;
12     end
13     new_input_num ← fan-in numbers of gate +
         fan-in numbers of fanout_gate - 1
14     if new_input_num > θ then
15         continue;
16     end
17     L ← perform elimination for gate in L
18     update new elimination score for fan-in and fan-out
         gates of gate
19 end
```

This transformation method is described in Algorithm 1. Initially, we identify the XOR gates and construct an XAG (XOR-AIG) graph (line 1). Next, we compute the elimination scores for all gate variables and maintain a max heap for efficient access (line 2). During each iteration, we select the gate variable with the largest score from the heap for elimination, but those gates with more than one fanout or the inputs of the newly merged gate exceeds a predefined limit (denoted as θ) are not eligible. Based on our observations, setting θ to 7 proves to be effective. When a gate variable is eliminated, the elimination scores of adjacent gate variables are affected and will be recalculated (lines 4-19). At this stage, the XAG is successfully simplified into an XLG graph composed of LUT and XOR gates.

D. XVSIDS: A Branching Heuristic Based on Circuit Structure

SAT solvers usually do not perform well on instances with a large amount of XORs, because the CDCL framework is not well-suited for handling XOR information effectively.

The traditional VSIDS algorithm updates the variable scores during conflict analysis by incrementing the scores of all the variables involved in the conflict [14]. To optimize this process for XOR gates in circuits, we propose a novel branching heuristic called **XVSIDS**.

After converting the circuit into its XOR Logic Graph (XLG) representation, we define "XOR edge" variables as those corresponding to XOR gates or directly influenced by XOR gates in their fan-in or fan-out. Since there are many "XOR edge" variables in arithmetic circuits, most variables have XOR neighboring variables.

To leverage this structure, our XVSIDS strategy modifies the variable score bumping mechanism: when a variable's score is increased, its adjacent XOR gates are also increased with a increment of half value of the primary variable's increment. This adjustment aims to enhance the locality of the CDCL search by prioritizing variables that are structurally significant in arithmetic circuits. Experimental results demonstrate that this approach yields substantial performance improvements.

IV. Experimental Results

In this section, we present the experimental setup, dataset sources and solvers in our experiments. Then, we report the evaluation results and perform experimental analysis of the proposed ideas.

A. Experimental Settings

We implement our X-SAT solver in C++ language and compile it with GNU g++ (version 10.5.0). It can be accessed from our GitHub repository [1]. All tools and competitors are written in C/C++ and compiled in the same environment as X-SAT.

All experiments were conducted on a high-performance computing cluster equipped with two AMD EPYC 7763 CPUs @ 2.45 GHz, totaling 128 physical cores, and 1TB RAM. The system runs Ubuntu 20.04 LTS (64-bit). A single core and less than 10GB was used for any test case considered in this section.

To validate and compare the performance of the solvers, we use Combinational Equivalence Checking (CEC) problems for testing. The dataset for testing is categorized into two groups: arithmetic circuits and non-arithmetic circuits, all in AIG format. The dataset can also be accessed through GitHub [2]. To enable the CNF solver to solve the AIG, we used the AIGER tool [15] to convert them into CNF format.

- **Arithmetic Circuits:** We use 22 industrial instances with the application of long bit-wise optimization, which are datapath circuits with adders and multipliers. Both SAT and UNSAT circuits are included. The vast majority are UNSAT instances, with `b27` being the only SAT instance.
- **Non-Arithmetic Circuits:** We select 13 academic miter circuits which are generated from the ISCAS85, ISCAS89, and ITC99 benchmark suites, emphasizing circuits of moderate complexity for demonstration. The method for generating miter circuits involves first converting the circuit into a combinational circuit using scan chains. Then, logic synthesis is performed using the ABC tool [16] before creating the miter circuit. The logic synthesis commands [17], `&st; &dch; &if -K 6; &mfs;`, are executed three times iteratively. This sequence is executed three times iteratively. Specifically, the selection criterion is based on the solving time of the

[1] https://github.com/shaowei-cai-group/X-SAT
[2] https://github.com/shaowei-cai-group/X-SAT/tree/main/data

moderately capable Glucose solver. Instances are selected if they can be solved within the time limit (2000 seconds) and have a solving time greater than 1 second.

B. Evaluation Metrics and Solvers

We evaluated our proposed solver, X-SAT, against three state-of-the-art CNF solvers and one advanced circuit solver.

- **abc-cirsat** [2]: The latest advanced circuit SAT solver, integrated into ABC. The official name of the solver could not be found, so it is referred to as abc-cirsat in this paper.
- **kissat** [4]: A state-of-the-art CNF SAT solver that dominated the main track of the 2020, 2021, 2022 and 2024 SAT competition. The latest version kissat-sc2024 is selected.
- **glucose** [5]: A widely-used SAT solver that improves upon several techniques introduced by MiniSat.
- **minisat** [6]: A SAT solver widely-used in industries, frequently used as a benchmark for performance evaluation.

Our X-SAT solver and abc-cirsat belong to circuit based SAT solver, while kisssat, glucose and minisat belong to CNF based SAT solvers.

The performance metrics are summarized using PAR2 (penalized average runtime). If a solver fails to solve a case within the 2000-second time limit, its runtime is recorded as 4000 seconds; otherwise, the actual runtime is used.

C. Results and Analysis

Table I summarizes the performance results for arithmetic and non-arithmetic circuits. It provides the following details for each instance: instance name, gate count, and execution times for each solver: kissat, glucose, minisat, abc-cirsat and x-sat.

Arithmetic Circuits: The results show that our proposed X-SAT solver, utilizing the structural elimination technique with the XVSIDS branching strategy, achieves an **38.26x** speedup compared to the other circuit SAT solver abc-cirsat. Furthermore, it delivers a **1.36x** improvement over the state-of-the-art CNF solver kissat.

Non-Arithmetic Circuits: X-SAT achieves a 7.08x speedup over abc-cirsat, while 2.01x slower than the best CNF based SAT solver kisssat on non-arithmetic circuites. However, X-SAT still outperforms glucose by 3.19x and minisat by 17.45x. For circuit-based SAT solvers, we have significantly reduced the performance gap with kissat for the first time.

D. Validation of Effectiveness

As shown in Table II, we compared two different solver configurations: one with structural elimination disabled (No-elim) and one with structural elimination enabled (Elim). For each configuration, we tested three different branching heuristics (J-Frontier, VSIDS, XVSIDS) and recorded the solving time for each instance.

Structural Elimination: Compared to circuits in AIG format, the XLG format reduces the number of variables by an average of 61.39% for arithmetic circuits and 61.90% for non-arithmetic circuits. For arithmetic circuits, enabling structural

TABLE I
COMPARING WITH OTHER SOLVERS (CUTOFF=2000S)

Instance	#Gates	CNF SAT			Circuit SAT	
		kissat	glucose	minisat	abc-cirsat	x-sat
ARITHMETIC CIRCUIT						
aa1	17940	**103.41**	–	–	–	123.04
aa30	17948	63.80	1678.14	–	–	**36.72**
aa38	17998	75.43	1759.38	–	–	**74.57**
ab18	19355	**105.75**	–	–	–	116.00
ab1	19354	75.95	958.29	–	–	**45.39**
ab38	19359	91.50	1451.84	–	–	**76.94**
ab40	19255	99.29	801.23	–	–	**34.42**
ac1	21792	187.40	–	–	–	**181.62**
ac20	21797	100.50	1978.48	–	–	**68.17**
ac2	21511	69.66	839.78	–	696.84	**44.44**
ac33	21024	69.34	331.14	–	399.13	**30.28**
ac3	21812	**130.44**	1798.88	–	–	140.52
ac40	21668	110.81	936.67	–	–	**52.55**
ac42	21286	68.99	463.62	–	408.42	**34.75**
ad14	23307	**168.03**	–	–	–	239.45
ad1	23301	103.22	1190.55	–	–	**62.21**
ad21	23035	90.71	466.48	–	826.44	**39.01**
ad2	22527	80.46	384.01	–	474.50	**36.58**
ad3	23214	115.58	650.99	–	–	**51.85**
ad41	22795	90.98	438.55	–	413.26	**38.12**
ad44	23319	172.05	1591.62	–	–	**120.26**
b27	12372	75.32	31.18	345.88	63.26	**7.20**
PAR2		102.21	1534.13	3833.90	2876.45	**75.19**
NON-ARITHMETIC CIRCUIT						
c3540	2113	0.86	4.43	23.77	1.08	**0.31**
c5315	3636	0.34	1.37	8.76	0.31	**0.26**
c7552	3803	0.33	1.16	2.50	21.09	**0.32**
b14	12033	**3.83**	39.21	291.02	170.16	8.33
b15	18212	**3.81**	13.35	20.62	24.35	4.22
b17	57076	**43.11**	145.42	258.24	203.98	74.47
b20	25255	**7.88**	103.22	1003.84	341.60	21.60
b21	25556	**8.73**	112.77	907.46	101.66	22.32
b22	37903	**20.68**	161.86	728.57	446.56	39.89
s15850	7978	**0.51**	1.24	2.48	1.22	0.63
s35932	19332	**0.57**	1.25	1.56	1.77	2.55
s38417	18703	**0.94**	4.44	7.63	4.65	3.39
s38584	26603	**1.82**	8.54	11.94	8.27	9.03
PAR2		**7.18**	46.02	251.41	102.05	14.41

TABLE II
DIFFERENT VERSIONS OF X-SAT (CUTOFF=2000S)

Instance	No-elim			Elim		
	J-Frontier	VSIDS	XVSIDS	J-Frontier	VSIDS	XVSIDS
ARITHMETIC CIRCUIT						
aa1	–	442.25	219.69	–	328.81	**123.04**
aa30	–	190.38	70.47	522.18	133.16	**36.72**
aa38	–	333.38	118.06	1634.51	257.32	**74.57**
ab18	–	549.97	230.01	–	371.90	**116.00**
ab1	–	140.14	80.67	632.14	98.17	**45.39**
ab38	–	266.47	116.98	1849.23	260.87	**76.94**
ab40	818.54	127.96	62.84	149.68	84.35	**34.42**
ac1	–	815.14	364.85	–	664.78	**181.62**
ac20	–	265.03	113.90	598.81	190.03	**68.17**
ac2	532.56	122.45	69.65	219.89	104.68	**44.44**
ac33	611.07	119.07	55.16	114.22	91.93	**30.28**
ac3	–	576.35	198.07	–	444.73	**140.52**
ac40	1112.11	156.90	97.32	275.65	146.45	**52.55**
ac42	530.51	147.21	56.13	250.23	81.31	**34.75**
ad14	–	730.88	347.58	–	619.14	**239.45**
ad1	–	182.38	144.67	690.73	130.09	**62.21**
ad21	799.64	126.95	82.50	307.20	88.92	**39.01**
ad2	580.60	78.72	58.54	180.11	65.86	**36.58**
ad3	1097.23	129.79	112.38	291.07	100.50	**51.85**
ad41	530.77	103.88	64.91	299.33	84.31	**38.12**
ad44	–	439.39	194.15	–	313.12	**120.26**
b27	51.80	21.09	61.86	55.59	32.30	**7.20**
PAR2	2484.77	275.72	132.75	1457.75	213.31	**75.19**
NON-ARITHMETIC CIRCUIT						
c3540	0.92	0.50	0.65	0.59	0.38	**0.31**
c5315	0.38	0.31	0.32	0.29	0.26	**0.26**
c7552	0.56	0.41	0.43	**0.31**	0.33	0.32
b14	27.72	10.73	16.72	30.18	**7.61**	8.33
b15	17.21	4.47	11.62	10.95	**3.45**	4.22
b17	370.28	125.66	111.01	182.40	88.22	**74.47**
b20	90.48	24.37	24.22	73.84	**13.11**	21.60
b21	134.85	24.45	30.28	46.19	**14.41**	22.32
b22	283.39	53.48	59.21	93.35	**31.30**	39.89
s15850	2.37	1.08	0.79	1.11	0.67	**0.63**
s35932	7.88	7.30	5.72	3.93	3.11	**2.55**
s38417	20.52	6.03	5.35	8.32	3.64	**3.39**
s38584	38.02	17.32	11.03	20.36	**8.57**	9.03
PAR2	76.51	21.24	21.33	36.29	**13.47**	14.41

elimination resulted in improvements of 1.70x, 1.29x, and 1.77x for the three heuristics(J-Frontier, VSIDS, XVSIDS), respectively. For non-arithmetic circuits, the improvements were 2.11x, 1.58x, and 1.48x. This demonstrates that structural elimination is a fundamental optimization, providing a positive impact across all circuits and heuristics.

XVSIDS: With structural elimination enabled, XVSIDS outperforms the traditional J-Frontier branching method, achieving a 19.39x improvement on arithmetic circuits and a 2.52x improvement on non-arithmetic circuits. This demonstrates that XVSIDS has a clear advantage over the traditional method. Similarly, with structural elimination enabled, XVSIDS achieves a 2.84x improvement over VSIDS on arithmetic circuits. On non-arithmetic circuits, the two methods show little difference, with PAR2 values of 13.47(VSIDS) and 14.41(XVSIDS), respectively.

V. CONCLUSION

In this paper, we proposed an efficient circuit SAT solver, named X-SAT. We introduced elimination strategies to rewrite the AIGs into a new format, XLG, which reduces variables. Additionally, we improved the traditional VSIDS strategy by considering XOR gates, leading to the XVSIDS heuristic, which demonstrated substantial performance gains for both arithmetic and non-arithmetic circuits.

X-SAT marks a significant milestone in circuit SAT solving by being the first solver to outperform the state-of-the-art CNF solver, Kissat, on arithmetic circuits. Moreover, its performance on non-arithmetic circuits remains competitive with the most advanced CNF SAT solvers, demonstrating its versatility and robustness across diverse circuit types.

Future work includes extending X-SAT's application to EDA tasks and designing a parallel circuit SAT solver to address even more complex and computationally demanding problems.

ACKNOWLEDGMENT

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300.

REFERENCES

[1] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, "Using sat for combinational equivalence checking," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*. IEEE, 2001, pp. 114–121.

[2] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko, "A circuit-based sat solver for logic synthesis," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–6.

[3] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.

[4] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "Cadical, gimsatul, isasat and kissat entering the sat competition 2024," *SAT COMPETITION 2024*, p. 8, 2024.

[5] G. Audemard and L. Simon, "On the glucose sat solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.

[6] N. Een, "Minisat-a sat solver with conflict-clause minimization," *Proc. Theory and Applications of Satisfiability Testing (SAT 05)*, 2005.

[7] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.

[8] N. Manthey, "Coprocessor 2.0–a flexible cnf simplifier: (tool presentation)," in *Theory and Applications of Satisfiability Testing–SAT 2012: 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings 15*. Springer, 2012, pp. 436–441.

[9] C.-A. Wu, T.-H. Lin, C.-C. Lee, and C.-Y. Huang, "Qutesat: a robust circuit-based sat solver for complex circuit structure," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[10] K. Hu and Z. Chu, "An efficient circuit-based sat solver and its application in logic equivalence checking," *Microelectronics Journal*, vol. 142, p. 106005, 2023.

[11] J. M. Silva and K. A. Sakallah, "Grasp-a new search algorithm for satisfiability," in *Proceedings of International Conference on Computer Aided Design*. IEEE, 1996, pp. 220–227.

[12] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Sat sweeping with local observability don't-cares," in *Proceedings of the 43rd Annual Design Automation Conference*, 2006, pp. 229–234.

[13] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 836–843.

[14] A. Biere and A. Fröhlich, "Evaluating cdcl variable scoring schemes," in *Theory and Applications of Satisfiability Testing–SAT 2015: 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings 18*. Springer, 2015, pp. 405–422.

[15] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 And Beyond," Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 11/2, Jul. 2011.

[16] "ABC," https://github.com/berkeley-abc/abc.

[17] H.-T. Zhang, J.-H. R. Jiang, L. Amarú, A. Mishchenko, and R. Brayton, "Deep integration of circuit simulator and sat solver," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 877–882.

CXL-Interplay: Unraveling and Characterizing CXL Interference in Modern Computer Systems

Shunyu Mao[†*], Jiajun Luo[†§*], Yixin Li[†], Jiapeng Zhou[‡], Weidong Zhang[§], Zheng Liu[§¶], Teng Ma[§**], Shuwen Deng[†‖**]

[†]Department of Electronic Engineering, Tsinghua University
[‡]Institute of Computing Technology, Chinese Academy of Sciences
[§]Alibaba Group
[¶]Zhejiang University
[‖]Zhongguancun Laboratory

***Abstract*—Compute Express Link (CXL) is a promising technology that addresses memory and storage challenges. Despite its advantages, CXL faces performance threats from external interference when co-existing with current memory and storage systems. This interference is under-explored in existing research. To address this, we develop CXL-Interplay, systematically characterizing and analyzing interference from memory and storage systems. To the best of our knowledge, we are the first to characterize CXL interference on real CXL hardware. We also provide reverse-reasoning analysis with performance counters and kernel functions. In the end, we propose and evaluate mitigation solutions.**

I. Introduction & Related Work

Compute Express Link (CXL) [8] provides a comprehensive solution to existing memory wall problems [13] and has garnered significant attention from both industry and academia. CXL offers several advantages, including high bandwidth density and a standardized interface for memory expansion and pooling [23]. These features make it an ideal solution to address the limitations of current cloud storage architectures, and have the potential to revolutionize data center architectures [4]. In industry, leading companies are actively exploring and adopting CXL to improve the efficiency and scalability of their data center products, including Intel [9], Samsung [25], and SK Hynix [19]. Academically, researchers are investigating various aspects of CXL, including its architecture [26], performance implications [28], [29], and potential security concerns [1].

However, in coexisting with current memory and storage systems in the server architecture, CXL faces understudied performance threats from external interference, which are not adequately reflected in existing research. Zooming out from CXL itself and considering its operational environment, CXL can potentially be affected by interference from 1) interactions between Main Memory (**MMEM**) and CXL and 2) interactions between neighboring storage components and CXL. Avoiding interference and maintaining performance isolation are critical for performance-sensitive applications. For instance, MT^2 [34] tries to explore the interference between persistent memory (PMEM) and DRAM, and it identifies noisy neighbors among concurrent applications to mitigate memory traffic interference. To the best of our knowledge, this ***interference issue for CXL has not been extensively studied in existing research***. Current CXL simulators typically introduce delay factors manually to simulate memory delays [20], [33], which neither reflects the real world environment nor considers interference. Moreover, existing hardware profiling studies on CXL primarily focus on evaluating the behavior of CXL under different operations and applications [17], [28], [29], without considering the impact of external environmental factors.

*The authors contributed equally to this work.
**Shuwen Deng and Teng Ma are co-corresponding authors.

Given that, we develop CXL-Interplay, which ***systematically characterizes and analyzes the potential reasoning of interference between memory/storage systems and CXL memory*** using carefully designed microbenchmarks and tailored real applications. We also discuss potential effective regulation that can be applied to fit real CXL interference scenarios.

First, in order to clearly identify and characterize CXL interference with memory and storage system, we develop a configurable microbenchmark and systematically go through the basic interference condition of CXL with MMEM and SSD system using *two different real CXL hardware setups*. As shown in Table I, we are the first to demonstrate and characterize the CXL interference using real CXL devices.

Furthermore, we conduct reverse-reasoning based on profiling evaluation results, exploring kernel functions and hardware performance counters, including the new CXL-related counters.

For the real testing setup, we explore file system, database, and machine learning applications for storage-related testing environment. For the memory-related testing environment, we focus on large-language model (LLM), in-memory database, and graph computing applications.

Given the interplay results, we further explore software and discuss hardware intervening solutions to provide effective regulation for the system to mitigate the interference impact. Based on our evaluation, it is feasible to restore bandwidth of the MMEM by 204GB per second, which reaches 99% of its original level, using software solutions, at the cost of a 5GB per second reduction in CXL bandwidth.

To summarize, this work makes the following contributions:

1) Systematically characterize the interference issue in current server architectures among CXL, MMEM, and storage devices. To the best of our knowledge, this is the first study examining interference of CXL on real hardware.
2) Conduct comprehensive microbenchmarks, evaluate memory-related and storage-related real applications and perform reverse-reasoning analysis for evaluation results.
3) Explore and evaluate software-based intervention solutions to address the interference problem and mitigate its impact on real hardware.

The code used in this paper will be released under open-source license at https://github.com/THU-HAS/CXL-Interplay.

II. Background & CXL-Interplay Setup

A. Compute Express Link (CXL)

Compute Express Link (CXL) [8], first developed in 2019, is an open standard interconnect designed to enhance the performance of

TABLE I: Comparison of CXL-Interplay with state-of-the-art CXL characterization work. To the best of our knowledge, we are the first to investigate interference and regulation of real CXL devices and conduct reverse-reasoning analysis.

	Real ASIC Device	Inter-ference	Regulation	Reverse-reasoning Analysis
Emulation works [16], [27], [33]	✗	✗	✗	✗
Sun [28]	✓	✗	✗	✗
Tang [29]	✓	✗	✗	✗
Liu and Wang [17]	✓	✗	✗	✗
Our Work	✓	✓	✓	✓

data-centric applications by enabling high-speed, low-latency communication between CPU/accelerators, memory, and storage.

The CXL protocol stack is divided into three parts: CXL.io, CXL.cache, and CXL.mem, each serving distinct purposes to facilitate various types of data transmission and memory access. CXL.io ensures compatibility with PCIe, enabling traditional I/O operations and device discovery. CXL.cache supports accelerators accessing CPU's internal caches and host memory. CXL.mem allows direct memory access and pooling, extending memory across multiple devices.

CXL devices are classified into three types, each tailored for specific applications. Type-1 devices, such as SmartNICs, utilize CXL.io and CXL.cache to facilitate communication with host DDR memory. Type-2 devices, including GPUs, ASICs, and FPGAs, use CXL.io, CXL.cache, and CXL.mem to share memory resources with the processor. Type-3 devices rely on CXL.io and CXL.mem to enable memory expansion and pooling. Type-3 devices enable extra DRAM capacity and bandwidth.

There are currently two types of CXL devices: FPGA-based and ASIC-based. Intel Agilex FPGA integrates CXL hard IP, which is a standard component in recent CXL studies (e.g., Sun et al [28], Zhong et al [36]). Besides, emerging ASIC-based CXL memory devices are becoming available, with promises from several vendors such as Samsung [25], Montage [31] and Micron [18].

B. Memory System and CPUless Node

DRAM is widely used in memory systems as the main memory. It is typically interconnected via a parallel interface. However, the limited capacity and bandwidth blocks its scalability for memory-intensive applications.

Secondary memory such as persistent memory (PMEM) can meet the demands of memory-intensive applications, and its large capacity makes it an alternative to DIMM. PMEM can be identified as the memory of an additional node without CPU.

As a type of secondary memory, CXL memory provides CPUless node interface. However, unlike PMEM, CXL devices rely on PCIe, utilizing its physical layer and provide multiplexing features, resulting in physical topologies akin to those of PCIe.

C. CXL-Interplay Hardware Setup

In this work, we systematically characterize and explore how CXL memory interferes with other memory-related access, specifically MMEM access, and storage-related access such as NVMe or SAS SSD, using microbenchmarks and real applications.

Table II demonstrates the experimental setup of CXL-Interplay. We have two systems that will be referred to later as System A and System B respectively. System A has SAS SSD and an ASIC-based CXL memory device manufactured by Montage. System B consists of NVMe SSD and an Intel FPGA-based CXL memory device.

TABLE II: Experimental setup of CXL-Interplay.

	System A	System B
CPU	4th Gen Xeon Platinum 4.0 GHz	4th Gen Xeon Gold 4.0 GHz
Number of physical cores per socket	48	32
Memory per socket Channels Speed	1TB, 8 Channel, DDR5-4800	256GB, 8 Channel, DDR5-4800
Storage	SAS SSD	NVMe PCIe 4.0 MLC SSD
File System	XFS	XFS
CXL memory device	Montage ASIC 64GB	Intel Agilex FPGA 32GB DDR4-2400
OS	CentOS 7	Ubuntu 22.04.4 LTS

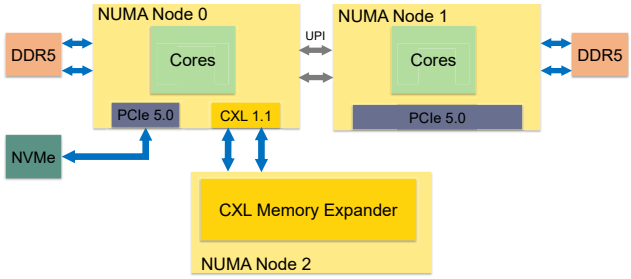


Fig. 1: Schematic illustration of hardware setup

Figure 1 illustrates the topology of our hardware setup, where NUMA node 0 and NUMA node 1 each possess their own cores and DDR5 DRAM, while the CXL Memory Expander is situated on NUMA node 2 with zero processor core, directly connected to Node 0. Therefore, tasks are always allocated to the cores on Node 0.

III. CXL-Interplay Microbenchmarks

In this section, we set up comprehensive microbenchmarks to examine the CXL interference problem with SSD and MMEM.

A. Microbenchmark Setup

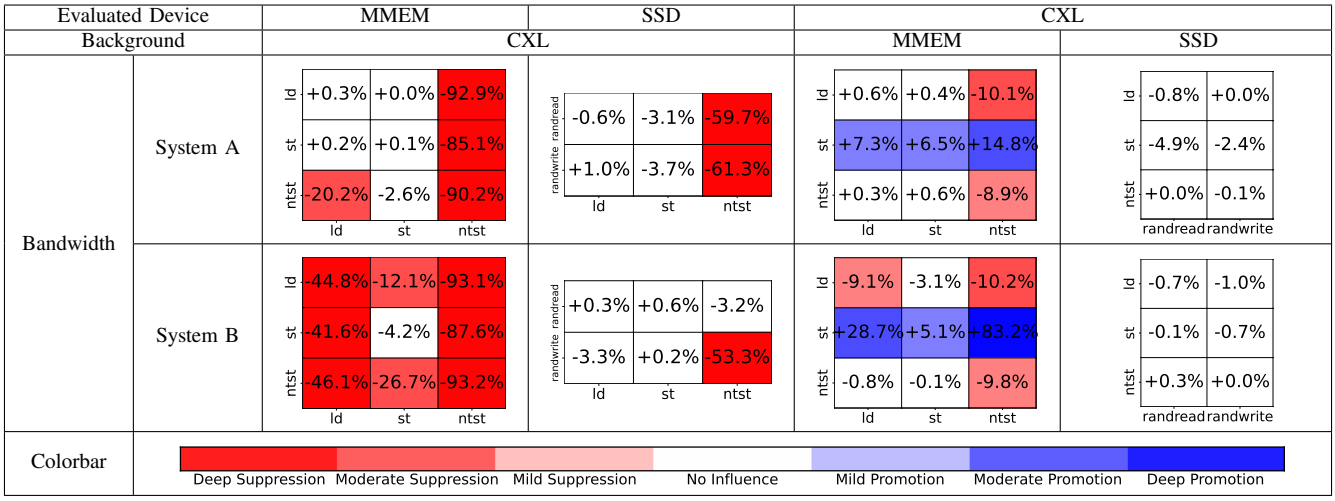
In our microbenchmark, three different memory-related operations (load (*ld*), store (*st*), and non-temporal store (*ntst*)) [1] and two different storage-related operations (random-read (*randread*) and random-write (*randwrite*)) are cross-evaluated.

For memory-related operations, we develop our microbenchmark by measuring the bandwidth of CXL memory and MMEM with cxlMemTest [28]. We carry out our experiments with all possible numbers of threads. For better measurement, we disable Hyperthreading [5], lock CPU's frequency to 2GHz, and clear cache before each test. We also allocate main process and interfering process to separate cores of the same NUMA node. The measurement is repeated multiple times and get average. We also carry out case studies on random/sequential storage access pattern and `MOVDIR64B` instruction.

For storage-related operations, we employ FIO [3] with *libaio* as *ioengine* to perform sustained *randread*/*randwrite* operations with an *iodepth* of 16 and a *block size* of 4KB for 20 seconds.

[1] We do not have non-temporal load mainly considering its sparse applicability, which are demonstrated in a lot of works [12], [21], [22]

TABLE III: The schematic illustration of bandwidth across different systems and configurations. In each box, the x-axis and y-axis represent the operation of the background and the device under test, respectively. We categorize the influence into seven types, each represented by a color in the colorbar: "Deep Suppression", below -40%; "Moderate Suppression", -40% to -10%; "Mild Suppression", -5% to -10%; "No Influence", within -5%; Promotion follows the same pattern.



B. Microbenchmark Characterization and Evaluation

1) MMEM/SSD with CXL Traffic: The left half of Table III illustrates the trends of MMEM/SSD's bandwidth under the interference of CXL. Each value represents the maximum interference observed across all thread count configurations. Our observations are as follows: a) The bandwidth of MMEM/SSD suffers significant interference when CXL performs *ntst*, reaching up to 93.2%. b) CXL *ld* and *st* can also be disruptive to MMEM, especially on System B, where over 40% interference is observed. c) As for SSD, the interference from CXL *ld* and *st* traffic is relatively mild.

INSIGHT #1: CXL traffic (especially non-temporal store (*ntst*)) significantly impairs bandwidth of concurrent MMEM and SSD traffic, with MMEM experiencing greater degradation than SSD.

2) CXL with MMEM/SSD Traffic: The right half of Table III illustrates the bandwidth trends of CXL under the interference of MMEM/SSD. Our observations are as follows: a) CXL bandwidth experiences performance boost when CXL performs *st* operations alongside MMEM traffic; b) In other MMEM cases, *ntst* operations exert a moderate to mild negative impact on CXL bandwidth. c) The bandwidth of CXL is not significantly influenced by SSD.

C. Reverse-reasoning Analysis

Figure 2 illustrates the datapath for CXL memory, SSD and MMEM workloads. Figure 2 depicts workload mixes involving SSD traffic and CXL traffic, where the CXL memory expander and storage device share the PCIe interface. It also shows workload mixes between CXL memory and MMEM, where interfaces are not shared, but sharing of network-on-chip (NoC) can lead to interference.

Performance counters. We discover that when SSD is under interference of CXL, its L2 and local LLC miss average latency becomes much higher than the no interference scenario, which prompts an investigation into the PMU counters listed in Table IV. It can be observed that there is a significant increase in L2 miss latency, up to a factor of 15, accompanied by a notable reduction in bandwidth. However, the L3 miss latency remains relatively constant. This suggests that an examination of network-on-chip (NoC) between the L2 and L3 will be necessary. CPUs need to maintain a record of in-flight transactions that access the L3 slices that are attached to other cores. This record table presumably will, however, become a bottleneck.

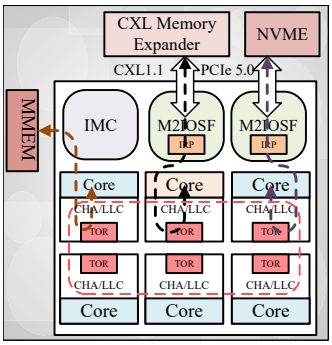


Fig. 2: Datapaths in filesystem, CXL and MMEM workloads.

From the discovery and the results in Table III and the datapaths in Figure 2, we then focus on: a) Table of Request (TOR), a queue between cores and LLC, tracking pending Cache Home Agent (CHA) transactions [15] of each core and PCIe interface; b) some OS *kernel functions*.

Correspondingly, we collect performance monitor unit (PMU) listed in Table IV, where $\frac{unc_cha_tor_occupancy.ia_miss}{unc_cha_tor_inserts.ia_miss}$ represents the average latency from cores that miss the LLC [15]. We can deduce that data transferred by CXL can be propagated throughout the entire LLC, causing congestion in TOR due to TOR's high occupancy behavior and affecting memory accesses of other cores. This can be demonstrated from value change of *unc_cha_tor_inserts.ia_miss* and *unc_cha_tor_occupancy.ia_miss*.

INSIGHT #2: CXL traffic occupies Table Of Request (TOR) for a lengthy period, causing congestion in TOR with other multicore process.

CXL counters. We also evaluate CXL performance in both instruction level (with tools we used above) and hardware level. We collect CXL-related PMU counters to compute CXL bandwidth. The results are consistent. For example, when CXL traffic with 16 *ntst* threads run under the interference of SSD traffic with 16 *randwrite* threads in System B, the bandwidth of CXL measured by cxlMemTest is 13169 MB/s, and the result computed by *cxlcm_txc_pack_buf_inserts.mem_data* is 13342 MB/s. The close agreement between the two values proves the reliability of our data.

TABLE IV: Ratio of related event and function of NVMe.

Related Event and Function	Description	Interference to No-interference Ratio
Load_L2_Miss_Latency	Average Latency for L2 cache miss demand Loads	15.2×
L3_Cache_Access_BW	Average per-core data access bandwidth to the L3 cache	0.14×
Load_L3_Miss_Latency	Average Latency for L3 cache miss demand Loads	1.01×
unc_cha_tor_occupancy.ia_miss	Occupancy number of TOR when request from iA misses LLC	4090×
unc_cha_tor_inserts.ia_miss	Inserts number of TOR when request from iA missses LLC	155×
memmove_sse2_unaligned_erms()	Memory move with Enhanced REP MOVSB, used in memcpy()	1.96×
copyout()	Kernel-to-User buffer copy	1.29×

Kernel functions. By profiling both SSD traffic alone and with CXL, two hotspot functions show significant changes. We notice that the time taken by *memmove_sse_unaligned_erms* and *copyout* (kernel-to-user copy) increases by 96.2% and 28.7%, respectively, when CXL is in the background. These functions, called by *memcpy* to move data in parallel, indicate that memory copy operations of system kernel processing data from devices are influenced.

D. Microbenchmarks on Other Configurations

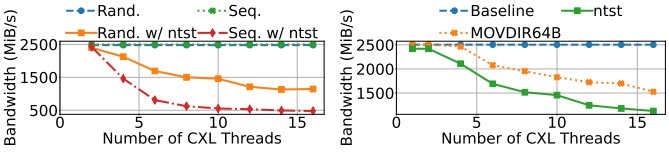
1) Sequential vs. Random Access: We also explore whether the interference differs when SSD performs sequential writes versus random writes. Results in Figure 3a highlight that SSD is more susceptible to CXL interference when performing sequential writes (up to 80.9% performance loss) compared to random writes (up to 54.5% performance loss). The reason is attributed to the greater spatial locality of data in sequential writes. Random operations presumably encounter a large number of cache line conflicts and are limited by the small number of available DDIO ways. Consequently, random operations result in an increased number of cache misses and occupy a smaller proportion of LLC space (2MB) than sequential accesses (4MB), which makes sequential writes more vulnerable to interference from the cache side, originating from CXL.

INSIGHT #3: SSD sequential writes exhibit greater LLC occupancy, which is interfered more by CXL traffic than random writes.

2) `MOVDIR64B` Operations: `MOVDIR64B` [12] is a specialized instruction that moves 64 bytes of data directly from source memory to destination memory with write atomicity. [2] Compared to *ntst*, `MOVDIR64B` adheres to the Write Combining (WC) memory type protocol and is weakly ordered. We wonder whether direct-store instructions like `MOVDIR64B` perform similarly to *ntst*. Figure 3b shows the influence of CXL `MOVDIR64B` operations on SSD *randwrite* in comparison with SSD performance alone and with CXL *ntst*. The results indicate that the interference from `MOVDIR64B` is similar to that of *ntst*, but imposes relatively less suppression on SSD performance than *ntst*. The presumable underlying cause is that unlike *ntst*, `MOVDIR64B` is capable of immediate eviction from the WC buffer [12]. As a result, data of `MOVDIR64B` spends a shorter time within the WC buffer, reducing the cost of tracking outstanding instructions. Therefore, the impact of `MOVDIR64B` is less significant than that of *ntst*. It is noteworthy that when the destination address of `MOVDIR64B` is local memory, thereby performing a normal load operation on CXL, the suppression disappears. This observation corroborates that the primary source of interference from CXL is its impact on the cache side.

INSIGHT #4: CXL `MOVDIR64B` operations cause less SSD performance suppression than *ntst* due to immediate WC buffer eviction, reducing instruction tracking costs.

[2]This operation is available on the 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.



(a) NVMe randwrite and sequential write bandwidth under the interference of CXL non-temporal store.

(b) NVMe randwrite bandwidth across different memory access instructions.

Fig. 3: Microbenchmark case studies

TABLE V: Description of the selected applications.

Category	Application	Description
File-related	Filebench [32]:	An open-source file system benchmark
	fileserver	Large files with 1:2 read/write ratio
	varmail	Small files with 1:1 read/write ratio
	RocksDB [30]	Log-structured merge-tree database 20-byte key size and 100-byte value size **Metric:** Operations per second
	MLPerf Storage [6]	Storage benchmark for deep learning **Metric:** Accelerator Utilization
Memory-related	GAPBS [7]	A graph computing benchmark PageRank Algorithm **Metric:** Trial time (lower is better)
	Redis [24]	An open-source in-memory database 50% read and 50% update **Metric:** Operations per second
	Large Language Model	TinyLlama [35], a decoder-only LLM **Metric:** Inference time (lower is better)

IV. CXL-Interplay Real Application Evaluations

We select a wide range of applications with different characteristics to study the interference problem. There are 4 types of interference scenarios: filesystem-related applications under CXL traffic, CXL-related applications under SSD traffic, MMEM-related applications under CXL traffic and CXL-related applications under MMEM traffic. We denote them as Type A, Type B, Type C and Type D respectively. The description of the selected applications is shown in Table V. The overview of the largest performance decline percentage evaluation result for each application is depicted in Figure 4. The key takeaway from this figure is that in most cases there is solid contention and interference across different access types and systems.

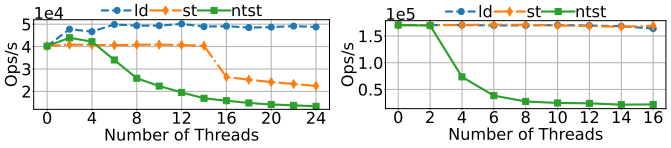
A. Filesystem-Related Applications

We select RocksDB database, Filebench, and machine learning workload MLPerf as our representative applications, while using CXL traffic as background to measure the interference for filesystem.

Figure 4 demonstrates that CXL's traffic has a considerable effect on the performance of most applications (Type A), with an observed

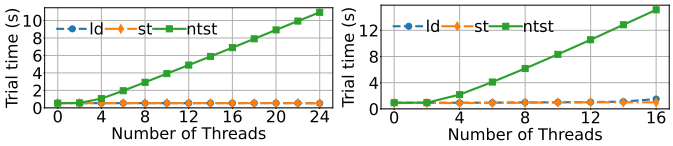


Fig. 4: CXL-related interference on real applications. The suffixes "-A/B/C/D" indicate the type to which the experiment belongs. We use different colors to differentiate background traffic types. For Type A, C and D, the background runs memory traffic which have three different types: *ld*, *st* and *ntst*. As for Type B, its background is file system and has four traffic types: *read*, *randread*, *write* and *randwrite*. We also use hatches to differentiate results from different systems. The height of the bars represents the maximum percentage of performance decline observed across all number of background thread count configurations for the corresponding scenarios. The y-axis is on logarithmic scale. [3]



(a) System A (b) System B

Fig. 5: The performance (operations per second) of RocksDB under CXL traffic. The x-axis indicates the number of background threads.



(a) System A (b) System B

Fig. 6: The performance (trial time) of GAPBS benchmark in Type C. The x-axis indicates the number of background threads.

detrimental impact reaching up to 90% for *ntst*. For *ld* and *st*, the impact can also be as high as 20% and 44% respectively. Moreover, the results presented in Figure 5 indicate an interesting and unexpected phenomenon that CXL's ld traffic can even enhance RocksDB's performance by up to 21.2% on system A.

Discussion. It can be observed that an increase in the thread count—and consequently the bandwidth—of the background CXL *ld* workload is associated with improved performance. To verify this, two other methods are applied to adjust CXL memory bandwidth: periodic cache flushing and injecting numerous `nop` instructions. We observe that the performance enhancement is directly linked to CXL memory bandwidth, occurring only when it exceeds 8 GB/s. Between 8 GB/s and 20 GB/s, the promotion grows linearly, while beyond that, the enhancement plateaus.

Albeit system-specific, we try to explore the root cause for promotion in System A with performance counters. for RocksDB, running a fixed number of operations both with and without CXL *ld* interference. The most notable observation is a 24% reduction in the number of instructions, especially memory instructions, under interference. This reduction in instructions plays a key role in the performance improvement.

INSIGHT #5: SSD workload bandwidth can benefit from a reduced instruction count when having sufficient CXL *ld* memory bandwidth as background.

B. Memory-Related Applications

We select graph benchmark GAPBS, in-memory database Redis and an LLM as our representative memory applications, covering Type B, C and D. It can be clearly observed that CXL memory's operations have a substantial impact on MMEM (Type C), consistent with microbenchmark results. Figure 6 shows a representative with up to 20.8× performance decline.

Conversely, the impact of MMEM on CXL memory is significantly less pronounced than that observed in Type C. We hypothesize that the observed limited impact is due to the fact that, although MMEM traffic is high, the low latency of MMEM does not result in significant consistence overhead, allowing CXL to operate normally.

Regarding SSD applications, their impact on CXL memory is insignificant, potentially due to the fact that the LLC consumed by SSD is a relatively small section (4MB) in comparison to those occupied by MMEM (50MB). Nevertheless, it should be noted that in some cases, particularly under *read* operation background, up to 12% decline in CXL performance can be observed.

INSIGHT #6: The interference from CXL memory to MMEM is significant. However, MMEM and SSD have mild impact on CXL memory in most cases due to DRAM's lower latency and SSD's lower LLC utilization.

V. Intervent the interference and Experiment on CXL-Related Regulation

Given the interference discussed in Section III and Section IV, we have observed and analyzed various interplay between CXL and SSD, as well as CXL and MMEM. In this section, we provide different trials and solutions to mitigate the interference with real CXL hardware. We consider using CPU usage restriction, such as time quota allocating, frequency scaling, memory bandwidth restriction, etc., to demonstrate the effectiveness of regulation on CXL real hardware.

A. CPU Usage Restrictions

To validate the functionality of CPU usage restriction, we choose three scenarios of severe interference: MMEM *ld*, MMEM GAPBS and filebench *varmail*, all influenced by 16 CXL *ntst* threads. We first use the counters mentioned in Section III-C and bandwidth monitoring tools [11] to identify the processes causing high CXL traffic and LLC occupancy periodically. Then we use Linux's *cgroups* [14] to limit their CPU quota. If the process still causes severe impact in the next period, we further adjust its resource allocation to achieve dynamic control.

As illustrated in Figure 7, reducing the allowed CPU usage quota leads to a recovery in MMEM bandwidth and a decrease in latency. Specifically, when the distribution of CPU cycles for CXL is restricted to 100% (1/16 of the time originally occupied), the

[3]In certain scenarios, the results are unstable and exhibit significant fluctuations. In such cases, the maximum decline is not meaningful, and we represent it as 0 in the figure.

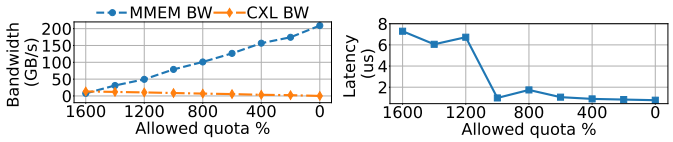
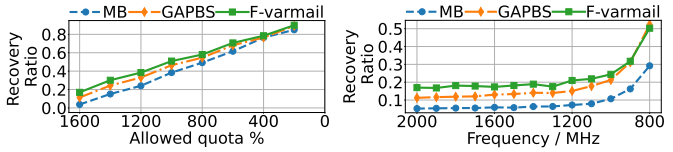


Fig. 7: Bandwidth and latency of MMEM when applying different CPU usage quotas to CXL threads. The horizontal axis is the allowed percentage of CPU usage.



(a) CPU usage restriction (b) Frequency Scaling

Fig. 8: Effectiveness of the two methods. Vertical axis is normalized to the performance without interference.

MMEM bandwidth recovers to 190 GB/s, which is approximately 92% of the bandwidth observed without interference. Additionally, latency returns to nearly its original level. This improvement in MMEM performance comes at the cost of a bandwidth reduction of about 10 GB/s on CXL. The results for the three applications are illustrated in Figure 8a. The linear and consistent impact observed across the three applications is due to the characteristic of CPU usage regulation, which influences a wide range of instructions. This approach facilitates a satisfactory and adjustable degree of recovery, albeit with varying degrees of performance decline in the background process.

B. Frequency Scaling

We also consider frequency scaling as a potential solution to mitigate the interference between CXL and MMEM. As is shown in Figure 8b, the recovery of MMEM threads is observed to be within the range of 30% to 52% when the background is constrained to very low frequency. This phenomenon can be explained by the fact that, except for very low frequency cases, memory instructions continue to fill the load/store queue. Consequently, memory-intensive tasks tend to demonstrate an exponential effect as the frequency decreases, yet still exhibit a suboptimal recovery rate. Although this method is less effective, it does not cause any damage to the memory-only background program.

C. Memory Bandwidth Restriction

The solution of regulating time slice and frequency is purely software-based and universal. But it also restricts the execution of memory-unrelated instructions, which causes additional performance degradation. On the other hand, memory bandwidth restriction presents a promising approach to mitigate interference between CXL and MMEM. We evaluate the effectiveness of using Intel MBA [10], which is a feature that allows users to allocate a portion of the memory bandwidth to specific processes. Limited by the I/O block configuration, it is only possible to restrict the CXL bandwidth to approximately less than half of its original value, which corresponds to a reduction of 5GB/s. This approach proves highly effective, enabling the three applications to recover to 99%, 95%, and 98% respectively, without affecting the compute instructions. However, this method is unable to provide control in a responsive manner and takes seconds to take effect, making it more suitable for implementation on the CXL device side.

D. Discussions on Other Methods

There are multiple solutions proposed to restrict the noisy neighbors and mitigate their negative impact on other threads. Apart from the methods discussed above, other methods include 1) cache partitioning, which is enabled by CPU vendors, such as Intel's Cache Allocation Technology (CAT) and 2) specialized memory bandwidth restriction, such as AMD's Slow Memory Bandwidth Allocation (SMBA) [2] targeting CXL devices, which has not been released. However, cache partitioning has minimal impact on operations that bypass the cache hierarchy, such as *ntst* operations, where it proves ineffective.

For hardware implementation, additional latency can be inserted into CXL device side after CPU has detected actual interference and then write the control register to change the added latency. However, this modification can highly depend on specific hardware which makes it harder to quickly deploy and generalize to different vendors.

VI. Practical Takeaways

Given the characterization and regulation experiments of CXL interference above, we provide the following takeaways for software (S) and hardware (H) developers respectively when developing future CXL-related systems:

(S1) Limit the number of threads or bandwidth of CXL *ntst* operations when possible to minimize their impact on MMEM and SSD workloads. (#1)

(S2) When running under CXL *ntst* traffic, SSD workloads should consider the trade-off between different access patterns such as random vs. sequential writes to maintain bandwidth. (#3)

(S3) For some systems, it is beneficial of lifting bandwidth to schedule SSD workloads with workloads having intensive CXL memory *ld* operations. (#5)

(H1) Hardware vendors can develop specialized TOR for slow tier memory to mitigate CXL contention with MMEM and SSD workloads. (#2)

(H2) To regulate the interference, it is advised to monitor and regulate on the CXL.mem device side with a more fine-grained MBA-like regulator.

VII. Conclusion

Real devices utilizing the emerging CXL technology are now finally commercially available. Researchers have actively characterized their features as isolated components. In this paper, we propose that since CXL devices need to interact with other components, it is necessary to characterize them in a coexisting manner. We develop customized microbenchmarks as well as real applications to characterize the interference problem. Our evaluation shows that this interference can cause a performance drop of up to 93.2%, along with other interesting observations. We also delve deep into the root causes of these observations. We provide concrete insights for readers and takeaways for developers. To mitigate the impact of this issue, we propose software mechanisms to manage CXL traffic. The results demonstrate the efficacy and trade-off of several methods.

Acknowledgments

This work was generously supported by National Key Research and Development Program of China under Grant (2024YFB4405402), Beijing Municipal Science and Technology Project (Nos.Z241100004224028), NSFC (U24A6009), Beijing Natural Science Foundation (L247013), BNRist. This work was also supported by Alibaba Research Intern Program.

References

[1] R. Abdullah, H. Lee, H. Zhou, and A. Awad, "Salus: Efficient security support for cxl-expanded gpu memory," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 1–15.

[2] *AMD64 Technology Platform Quality of Service Extensions*, Advanced Micro Devices, Inc., February 2022.

[3] J. Axboe, "Flexible i/o tester," 2022. [Online]. Available: https://github.com/axboe/fio

[4] E. L. Bai and S. A. Raut, "An analysis on compute express link with rich protocols and use cases for data centers," in *International Conference on Image Processing and Capsule Networks*. Springer, 2022, pp. 787–800.

[5] D. Bakhvalov, *Performance Analysis and Tuning on Modern CPUs: Squeeze the last bit of performance from your application.* Independently published, 2020.

[6] O. Balmau, "Characterizing i/o in machine learning with mlperf storage," *ACM SIGMOD Record*, vol. 51, no. 3, pp. 47–48, 2022.

[7] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2017. [Online]. Available: https://arxiv.org/abs/1508.03619

[8] C. E. L. Consortium *et al.*, "Compute express link specification," 2020.

[9] I. Corporation, "Cxl protocol hard ip functionality accelerates a wide range of data-centric workloads," https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/interface-protocols/cxl-ip.html.

[10] I. Corporation, "Intel® resource director technology (intel® rdt) framework," https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html, 2015.

[11] I. Corporation, "Intel performance counter monitor," 2024, accessed: 2024-08-02. [Online]. Available: https://github.com/intel/pcm

[12] I. Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, M-U*. Intel Corporation, 2024.

[13] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "Ai and memory wall," *IEEE Micro*, 2024.

[14] T. Heo, *Control Groups v2*, 2015. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v2.txt

[15] *4th Gen Intel® Xeon® Scalable Processor XCC (Codename Sapphire Rapids) Uncore Performance - Monitoring Guide*, Intel Corporation, April 2024.

[16] T. Lee, S. K. Monga, C. Min, and Y. I. Eom, "Memtis: Efficient memory tiering with dynamic page classification and page size determination," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 17–34.

[17] J. Liu, X. Wang, J. Wu, S. Yang, J. Ren, B. Shankar, and D. Li, "Exploring and evaluating real-world cxl: Use cases and system adoption," 2024. [Online]. Available: https://arxiv.org/abs/2405.14209

[18] MICRON, "Cz120 memory expansion module," https://www.micron.com/products/memory/cxl-memory.

[19] S. H. NEWSROOM, "Sk hynix introduces industry's first cxl-based computational memory solution (cms) at the ocp global summit," https://news.skhynix.com/sk-hynix-introduces-industrys-first-cxl-based-cms-at-the-ocp-global-summit/, 2022.

[20] A. Puri, K. Bellamkonda, K. Narreddy, J. Jose, V. Tamarapalli, and V. Narayanan, "Dracksim: Simulating cxl-enabled large-scale disaggregated memory systems," in *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2024, pp. 3–14.

[21] A. Raad, L. Maranget, and V. Vafeiadis, "Extending intel-x86 consistency and persistency: formalising the semantics of intel-x86 memory types and non-temporal stores," *Proc. ACM Program. Lang.*, vol. 6, no. POPL, jan 2022. [Online]. Available: https://doi.org/10.1145/3498683

[22] B. Ramesh, N. Contini, N. Alnaasan, K. K. Suresh, M. Abduljabbar, A. Shafi, H. Subramoni, and D. K. D K Panda, "Hint: Designing cache-efficient mpi_alltoall using hybrid memory copy ordering and non-temporal instructions," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 802–813.

[23] S. Ryu, S. Kim, J. Jun, D. Moon, K. Lee, J. Choi, S. Kim, H. Kim, L. Kim, W. H. Choi, M. Nam, D. Hwang, H. Roh, and Y. Joo, "System optimization of data analytics platforms using compute express link (cxl) memory," in *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2023, pp. 9–12.

[24] S. Sanfilippo, "Redis," 2009. [Online]. Available: https://redis.io/

[25] S. Semiconductor, "Samsung develops industry's first cxl dram supporting cxl 2.0," https://semiconductor.samsung.com/news-events/news/samsung-develops-industrys-first-cxl-dram-supporting-cxl-2-0/, 2022.

[26] D. D. Sharma, "Novel composable and scaleout architectures using compute express link," *IEEE Micro*, vol. 43, no. 2, pp. 9–19, 2023.

[27] K. Song, J. Yang, S. Liu, and G. Pekhimenko, "Lightweight frequency-based tiering for cxl memory systems," *arXiv preprint arXiv:2312.04789*, 2023.

[28] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong, R. Wang, J. H. Ahn, T. Xu, and N. S. Kim, "Demystifying cxl memory with genuine cxl-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 105–121. [Online]. Available: https://doi.org/10.1145/3613424.3614256

[29] Y. Tang, P. Zhou, W. Zhang, H. Hu, Q. Yang, H. Xiang, T. Liu, J. Shan, R. Huang, C. Zhao *et al.*, "Exploring performance and cost optimization with asic-based cxl memory," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 818–833.

[30] F. R. team, "A persistent key-value store for fast storage environments," https://rocksdb.org/, 2024.

[31] M. Technology, "Cxl memory expander controller (mxc)," https://www.montage-tech.com/MXC.

[32] T. Vasily, "Filebench: A flexible framework for file system benchmarking.; login," *The USENIX Magazine*, vol. 41, p. 6, 2016.

[33] Y. Yang, P. Safayenikoo, J. Ma, T. A. Khan, and A. Quinn, "Cxlmemsim: A pure software simulated cxl. mem for performance characterization," *arXiv preprint arXiv:2303.06153*, 2023.

[34] J. Yi, B. Dong, M. Dong, R. Tong, and H. Chen, "{MTˆ 2}: Memory bandwidth regulation on hybrid {NVM/DRAM} platforms," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022, pp. 199–216.

[35] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," 2024.

[36] Y. Zhong, D. S. Berger, C. Waldspurger, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury, and A. Cidon, "Managing memory tiers with cxl in virtualized environments," in *Symposium on Operating Systems Design and Implementation*, 2024.

DRAFT: Decoupling Backpropagation from Pre-trained Backbone for Efficient Transformer Fine-Tuning on Edge

Zhirui Huang†, Shiwei Liu†, Haozhe Zhu†, Qi Liu†, and Chixiao Chen†‡*
† *State Key Laboratory of Integrated Chips and Systems (SKLICS), Fudan University, Shanghai, China*
‡ *Shaoxin Laboratory, Shaoxing, Zhejiang, China*
* E-mail: cxchen@fudan.edu.cn

***Abstract*—Transformers have demonstrated outstanding performance across diverse applications recently, necessitating fine-tuning to optimize their performance for downstream tasks. However, fine-tuning remains challenging due to the substantial computational costs and storage overhead of backpropagation (BP). The existing fine-tuning techniques require the BP through the massive pre-trained backbone weights for computing the input gradient, resulting in significant computing overhead and memory footprint for resource-constrained edge devices. To address the challenge, this work proposes an algorithm-hardware co-design framework, DRAFT, for efficient Transformer fine-tuning by decoupling the BP from the backbone weights, thereby efficiently reducing the BP overhead. The framework employs Feedback Decoupling Approximation (*FDA*), an efficient fine-tuning algorithm that decouples BP into two low-complexity pathways: trainable adapter pathway and sparse ternary Bypass Network (*BPN*) pathway. The two pathways work collaboratively to approximate the conventional BP process. Further, a DRAFT accelerator is proposed, featuring a reconfigurable design with lightweight sparse gather networks and dynamic workflows to fully harness the sparsity and data parallelism inherent to the FDA. Experimental results demonstrate that DRAFT achieves a speedup of 4.9× and an energy efficiency improvement of 4.2× on average compared to baseline fine-tuning methods across multiple fine-tuning tasks with negligible accuracy loss.**

I. Introduction

In recent years, Transformers [1] have demonstrated outstanding performance on tasks of multiple domains, including Natural Language Processing (NLP) [2], [3] and Computer Vision (CV) [4]. Owing to their impressive capabilities, it is prevalent to fine-tune the pre-trained Transformers (referred to as ***backbone***) for various downstream tasks. The demand for local fine-tuning on edge hardware is rapidly growing. This enhances user privacy protection and reduces latency and energy consumption associated with data transmission.

The primary challenge of Transformer fine-tuning is that backpropagation (BP) nearly tripled the computational cost compared to feedforward (FF)-only inference due to the BP computation for both weight and input gradient [5], [6]. Unlike conventional full fine-tuning, which updates all the pre-trained weight as illustrated in Fig. 1(a), Parameter-Efficient Fine-Tuning (PEFT) [7]–[9] is a recent research direction aims to limit the number of trainable parameters for online learning. For instance, as illustrated in Fig. 1(a), Adapters [7] are small bottleneck modules inserted into Transformer layers, while LoRA [10] injects trainable parameters alongside the frozen backbone weights. They reduce the associated weight gradient computation and optimizer state storage efficiently.

However, Transformer fine-tuning remains challenging for two main reasons. First, FF accuracy is critical for effective weight updates, restricting the compression feasible for the pre-trained model prior to the fine-tuning process [11], [12]. As a result, the BP through the backbone, comprising millions to billions of parameters, incurs substantial computational overhead. Second, as shown by the red line in Fig. 1(a), existing optimizations primarily address the weight gradient computation yet overlook the input gradient computation, necessitating BP through the heavy but frozen backbone weight. This oversight limits the reduction of fine-tuning computation to a maximum of 31%, as illustrated in Fig. 1(b).

The reliance of BP on the backbone weight makes Transformer fine-tuning computationally intensive for resource-limited edge devices. Previous efficient DNN training accelerators based on asymmetric BP [13], [14] and sparse BP [15], [16] suggest that accurate, symmetric BP may not be essential. Inspired by this line of research, we investigate the potential to uniformly reduce the BP computation cost for both input and weight gradients across all linear layers in the Transformer model, thereby significantly improving fine-tuning efficiency.

We propose an algorithm-hardware co-design framework for edge Transformer fine-tuning called DRAFT. As shown in Fig. 2, the key idea of DRAFT is to decouple the BP through the backbone weight into two lightweight pathways. These pathways jointly approximate pre-trained weight with low complexity, eliminating BP through the backbone weight without trading off fine-tuning accuracy. The major contributions are summarized as follows:

- We propose Feedback Decoupling Approximation (*FDA*), an efficient fine-tuning technique that fully reduces BP overhead by skipping the BP through the backbone weight with a sparse ternary Bypass Network (*BPN*). Decoupled-initialized and trainable adapters bridge the separated FF and BP pathways, reducing the BPN's complexity while maintaining fine-tuning performance.

* Chixiao Chen is the corresponding author.

This work was supported by Shanghai 2024 "Science and Technology Innovation Action Plan" Fundamental Research Program in Integrated Circuit under Grant No. 24JD1400300 and the National Natural Science Foundation of China under Grant No. 62322404.

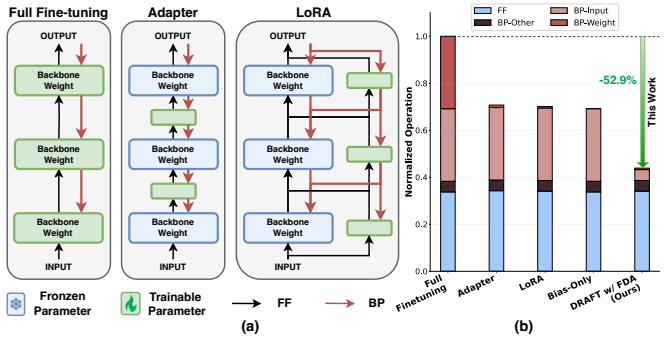


Fig. 1. (a) Illustration of the fine-tuning path of different fine-tuning methods with Full Fine-tuning, Adapter, and LoRA. (The compute paths of the linear layers are illustrated while the attention mechanism propagates primitively). (b) Fine-tuning operations comparison of Full Fine-tuning, Adapter [7], LoRA [10], Bias-only [8], and DRAFT (based on FDA) for the BERT-Base model.

- We design an efficient DRAFT accelerator for Transformer fine-tuning to leverage the sparsity and data parallelism inherent to the FDA. The reconfigurable processing element array supports lightweight gather networks and dynamic workflow to improve the throughput across different fine-tuning phases dynamically.
- We evaluate DRAFT on various models, datasets, fine-tuning algorithms, and hardware accelerators. Evaluation results illustrate that the FDA reduces the computational overhead of BP to below 15% with negligible accuracy loss. The algorithm-hardware co-design enables DRAFT to further reduce fine-tuning latency by 4.9× and substantially improve energy efficiency by 4.2× on average compared to the baselines.

II. Background and Motivation

A. Efficient Transformer Fine-tuning

Recently, numerous efficient fine-tuning techniques have been introduced to enable fine-tuning Transformers on commercial GPUs and even edge devices. A prominent parameter-efficient fine-tuning (PEFT) technique, LoRA [10], reduces the number of trainable parameters by integrating the trainable adapter along the linear layer. The computation output y of a linear layer on input X in LoRA is:

$$y = (W_o + \Delta W)X \approx (W_o + \frac{\alpha}{r} \cdot AB^\top)X \tag{1}$$

where W_o is the pre-trained weight matrix, ΔW is the weight update to W_o, and A and B are the low-rank approximation of ΔW initialized with random data and zeros, with a scaling factor of α and rank r. LoRA achieves near full fine-tuning accuracy by updating only a small fraction of the weights. However, prior efficient fine-tuning solutions [17]–[19] based on PEFT fall short of addressing the BP overhead associated with computing input gradients via the enormous backbone weight, which will be further discussed in Sec. II-C.

B. Training Techniques beyond Backpropagation

Substantial research has explored alternative training techniques to replace traditional BP. For example, the Google Brain team introduced Feedback Alignment [20], [21], which addresses the weights symmetry problem by proposing *asymmetric backpropagation*. This line of research focuses on producing high-quality gradients to achieve comparable performance to the conventional BP algorithm rather than exploring acceleration opportunities. Additionally, some strategies approximate weight gradients directly [22]. For instance, ADA-GP [5] predicts BP gradients using an auxiliary neural network. These techniques reveal that precise gradient computation via the chain rule is not always essential. However, most methods have primarily been validated on small-scale neural networks and suffer from significant accuracy loss when applied to larger, more complex models [21], [22].

C. Motivation

Backpropagation Computation Overhead: We begin by analyzing the network structure as a multi-layer perceptron (MLP): $Y = f_N(f_{N-1}(...f_2(f_1(x))...))$, where the output at layer i is given by $y_i = \mu_i(a_i) = \mu_i(W_i X_i)$. W_i is the pre-trained backbone weight for i-th layer, X_i denotes the input, and μ_i is the activation function, with the bias term omitted for simplicity. As illustrated in Fig. 3(a), the major computation overhead of conventional BP is to obtain the gradients of the input and the weights based on the loss function L:

$$\frac{\partial L}{\partial X_i} = \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial a_i}\frac{\partial a_i}{\partial X_i} = \frac{\partial L}{\partial y_i}\mu_i' W_i^\top \tag{2}$$

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial a_i}\frac{\partial a_i}{\partial W_i} = \frac{\partial L}{\partial y_i}\mu_i' X_i \tag{3}$$

Current fine-tuning algorithms primarily optimize by reducing the number of trainable weights, thus decreasing the computation associated with the weight gradient term $\frac{\partial L}{\partial W_i}$ in Eq. (3) (denoted as G_W in Fig. 3). However, they fall short of addressing the computation of the input gradient $\frac{\partial L}{\partial X_i}$ in Eq. (2) (denoted as G_X in Fig. 3). Consequently, as illustrated in Fig.1 (b), BP computation costs remain nearly equivalent to those of the FF stage. Techniques such as Partial Tuning [23], [24] mitigate this issue by shortening the BP depth and tuning

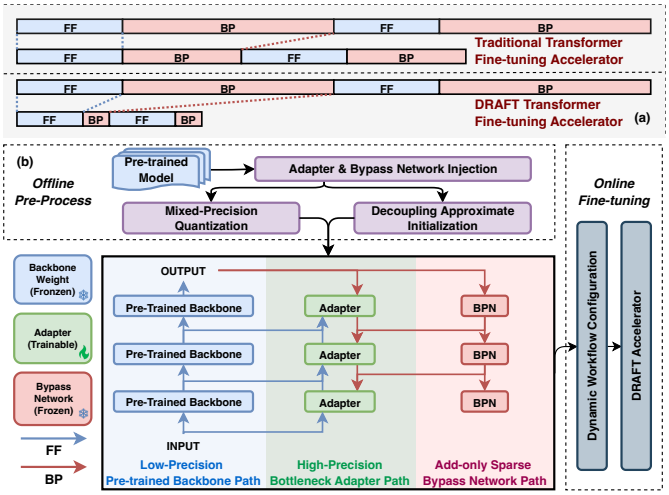


Fig. 2. (a) Comparison of DRAFT design concept with previous fine-tuning accelerator. (b) DRAFT framework overview, where the fine-tuning FF and BP pathways are decoupled and asymmetric compared to conventional BP.

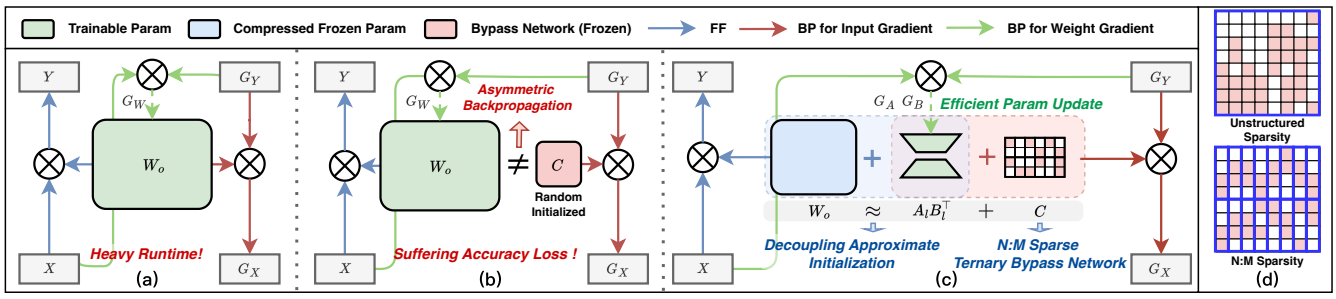


Fig. 3. The linear layer computation graph comparison of (a) conventional backpropagation, (b) asymmetric backpropagation based on Feedback Alignment, (c) approximate backpropagation based on Feedback Decoupling Approximation (FDA). The FDA delivers an accuracy-cost trade-off solution by decomposing and decoupling approximating the pre-trained weight. (d) Illustration of unstructured sparsity and N:M sparsity where N:M sparsity is more hardware-friendly.

the final few layers. However, these approaches limit updates to only a small subset of Transformer blocks, constraining the potential performance improvements. **In contrast, DRAFT fully reduces BP computation cost for input and weight gradient** by decoupling the BP from the backbone weight, striking a balance between hardware and algorithm efficiency.

III. Overview of DRAFT Framework

In this work, we propose DRAFT, an efficient algorithm-hardware co-design framework for Transformer fine-tuning on edge. The fine-tuning process based on DRAFT is illustrated in Fig. 2(b). **In the offline stage**, DRAFT pre-processes the pre-trained model in three steps: (1) Adapter and Bypass Network (*BPN*) Injection, (2) Pre-trained Model Compression, and (3) Decoupling Approximate Initialization. Based on the Feedback Decoupling Approximation (*FDA*) algorithm, DRAFT introduces an asymmetric fine-tuning pathway that skips the BP through the backbone weight with lightweight BPNs. The adapters bridge the gap between the separate FF and BP pathways, preventing accuracy degradation from the asymmetric BP, as detailed in Sec. IV-A. The BPN is an N:M sparse ternary network designed for both hardware and algorithm efficiency, as detailed in Sec. IV-B. **In the online stage,** DRAFT divides fine-tuning into three pathways. During the FF phase, the output is computed through the compressed frozen backbone and the trainable adapter. In the BP phase, the error gradient propagates through the BPN and adapter with minimum computation overhead. The reconfigurable DRAFT accelerator is designed to efficiently leverage the sparsity and parallelism inherent to the FDA, as detailed in Sec. V.

IV. DRAFT Fine-tuning Algorithm

A. Feedback Decoupling Approximation Algorithm

Asymmetric BP based on the Feedback Alignment [20], [21] has long been studied to produce synthetic gradients instead of conventional BP, eliminating the reliance on weight symmetry of BP. As illustrated in Fig. 3(b), this approach demonstrates the potential to bypass the BP through the backbone network by reformulating the input gradient computation in Eq. (2):

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial y}\mu' C \tag{4}$$

In the conventional BP algorithm, $C = W_o^\top$, which is also referred to as *symmetric BP*. However, asymmetric BP uncovers redundancy in the standard approach, relaxing the constraint of weight symmetry, i.e. $C \neq W_o^\top$, thereby presenting an opportunity to perform BP without the backbone weight. Here, C can be a compressed version of the weight W_o or even random numbers, expressed as $C = \phi(W_o)$, where $\phi(\cdot)$ can be different compression algorithms or random number generators [21]. However, asymmetric BP is inherently prone to accuracy degradation as it lacks the ability to leverage pre-trained knowledge.

Feedback Decoupling Approximation for Lightweight Backpropagation: As illustrated in Fig. 3(c), to address the performance degradation of vanilla asymmetric BP, we propose the Feedback Decoupled Approximation (FDA) algorithm, which compensates for compression loss during the Bypass Network initialization with a trainable adapter. Different from the asymmetric BP, the FDA initializes the BPN by decoupling the backbone weight into two components:

$$\begin{aligned}\frac{\partial L}{\partial X} &= \frac{\partial L}{\partial y}\mu' W_o^\top = \frac{\partial L}{\partial y_i}\mu' \left[\phi(W_o) + l\right] \\ &\approx \frac{\partial L}{\partial y_i}\mu'(C + A_l B_l^\top)\end{aligned} \tag{5}$$

where C is the target BPN weight, and the initialization residual loss is denoted by $l = W_o^\top - \phi(W_o)$. We compensate the compression loss during initialization by decomposing the l into its low-rank approximation versions, A_l and B_l, with the singular value decomposition (SVD):

$$\begin{aligned} l &\approx \sum \sigma_{t,i} u_{t,i} v_{t,i}^\top \\ A_l &= [\sqrt{\sigma_{t,1}} u_{t,1}, \ldots, \sqrt{\sigma_{t,r}} u_{t,r}] \\ B_l &= [\sqrt{\sigma_{t,1}} v_{t,1}, \ldots, \sqrt{\sigma_{t,r}} v_{t,r}] \end{aligned} \tag{6}$$

where r is the rank of A_l, B_l. With the decoupling approximate initialization, the matrices A_l, B_l serve not only as the bridge between the separate FF and BP pathways but also as the trainable *adapters* for efficient parameter updates during fine-tuning. As shown by Fig. 3(c), along the blue FF path, the output Y is generated through the backbone and adapter. In contrast, following the red BP path, the input gradient G_X is computed through the adapter and BPN, independently of the backbone weights. The backbone weight and BPN weight are frozen during fine-tuning. By adjusting the adapter's rank, we can balance accuracy and computational complexity to achieve different compensation levels during initialization. Based on the FDA, we optimize input gradient computation through the BPN and alleviate the weight gradient computation cost

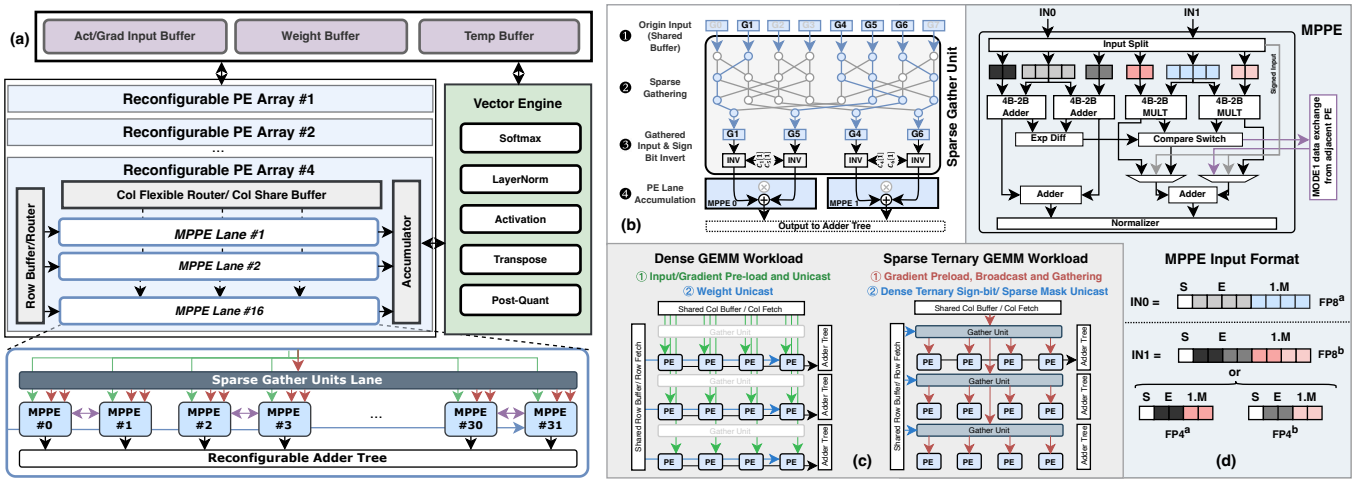


Fig. 4. (a) Overview of DRAFT Accelerator. (b) Illustration of the N:M sparse ternary compute flow in the BPN with an example of $S_t = 0.25$ and $m_t = 8$. (c) Accelerator workflow during the computation of different fine-tuning pathways. *left*: Weight stationary workflow for dense GEMM in backbone and adapter pathway. *right*: Input stationary workflow for sparse ternary GEMM in BPN pathway. (d) The design of MPPE and the input formats.

by the adapter. Collectively, the FDA uniformly reduces the computation overhead of BP, as illustrated in Sec. VI-A.

B. N:M Sparse Ternary Bypass Network

To ensure that the BPN remains lightweight with minimal memory and computational overhead, we employ an N:M sparse ternary quantization method for its initialization [25], [26], as the primary computational burden in BP stems from the floating-point GEMM. As illustrated in Fig. 3(d), this approach eliminates multiplications in the BPN while avoiding substantial routing hardware overheads required by unstructured sparsity, balancing algorithmic and hardware efficiency. We divide the pre-trained weight of each output channel into multiple blocks of size m_t. The BPN ternary weight $c_t^{(i,j,k)}$ for the j-th block in channel i is initialized as follow:

$$c_t^{(i,j,k)} = \begin{cases} 0 & ,|w_o^{(i,j,k)}| \le t^{(i,j,:)} \vee k \notin \text{TopK}(|w_o^{(i,j,:)}|) \\ +s^{(i,j,:)} & ,w_o^{(i,j,k)} > t^{(i,j,:)} \wedge k \in \text{TopK}(|w_o^{(i,j,:)}|) \\ -s^{(i,j,:)} & ,w_o^{(i,j,k)} < -t^{(i,j,:)} \wedge k \in \text{TopK}(|w_o^{(i,j,:)}|) \end{cases} \quad (7)$$

where the $s^{(i,j,:)}$ is the scaling factor of each block. The sparsity ratio of all blocks, denoted as S_t, defines the proportion of non-zero ternary values within the block and is controlled by adjusting the hyper-parameter threshold $t^{(i,j,:)}$ and the TopK selection. The ternary sparsity is determined and encoded at the pre-processing stage.

To further reduce the fine-tuning computation and memory overhead, we employ the widely adopted *Compressing-then-Tuning* scheme [12], [19], and integrate the Microscaling (MX) format [27] to compress each fine-tuning pathway. MX groups m data into a block with a shared exponent (i.e., shared scale 2^s) and normalizes each element in the block with this scale. Each element is either a low bit-width floating point or an integer. We employ mixed-precision MX formats to achieve optimal compression performance, quantizing the pre-trained backbone weights in MXFP4 (E2M1) and the adapter parameters, activations, and gradients in MXFP8 (E4M3). Additionally, the MX block size m is aligned with the N:M ternary quantization block size m_t, both set to 32, ensuring compatibility in hardware implementation.

V. DRAFT Hardware Architecture

A. Overview

Fig. 4(a) illustrates the DRAFT accelerator design, whose primary component is the parallel Reconfigurable Processing Element Arrays (RPEA). Each RPEA contains 16 Mixed-Precision Processing Element (MPPE) lanes, each equipped with 32 MPPEs, 8 Sparse Gather units, and a configurable adder tree. These PEs are interconnected via a dynamic data fetch network. Each RPEA includes row and column ports with local shared buffers and input routers. The pre-compiled configuration enables the scheduling of various workflows to optimize on-chip bandwidth and computational resource utilization. Vector Engines support vector-wise operations such as Softmax, Activation, and Layer Normalization. The total on-chip buffer size is 1 MB. Data is stored across the memory hierarchy using block mapping [28]. The temporary buffer accumulates outputs and supports the Vector Engine. At the start of each batch, weight and input data are pre-loaded into the respective on-chip buffers. DRAFT employs a ping-pong strategy to divide the batch into multiple processing phases if the data exceeds the on-chip buffer capacity.

B. Sparse Gather Unit

To fully leverage the N:M ternary sparsity introduced by the BFN, we design and integrate Sparse Gather units within each PE lane. This unit utilizes an inverse butterfly network [25] to compress gradient data G_Y corresponding to non-zero BFN weights, achieving this in $log(m_t)$ steps. Fig. 4(b) shows the computation process of a 3-stage inverse butterfly network with an input block size of 8 and a BFN sparsity ratio S_t of 0.5. In this setup, gradients with a block size of m_t are broadcast from the column-shared buffer to each gather unit. The gathered gradients are then fetched to the MPPE lane for accumulation directly without multiplication, while their sign bits are adjusted based on the sign bit of dense ternary weight. In the physical design, each lane includes 8 gather units configured with a 32-to-8 structure based on the algorithm sensitivity study in Sec. VI-A.

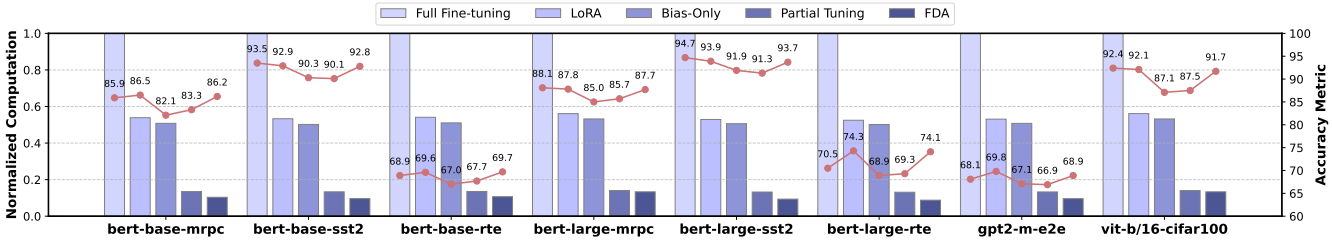


Fig. 5. Fine-tuning accuracy (red dots) and normalized BP computation reduction of different fine-tuning methods on various models and tasks.

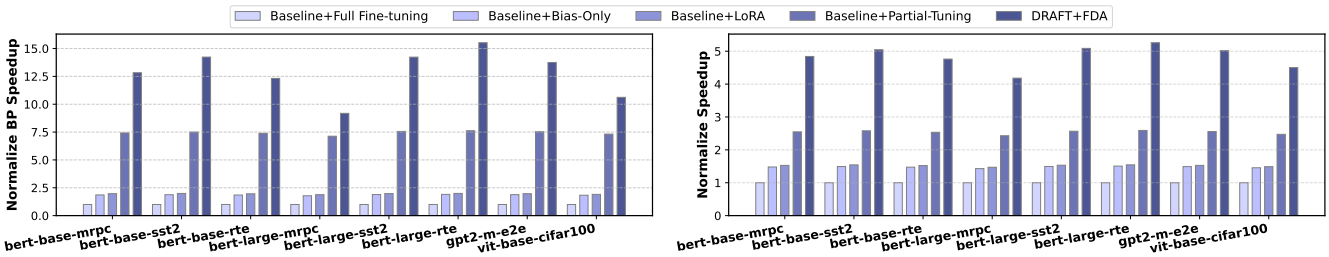


Fig. 6. (a) Normalized BP speedup and (b) normalized end-to-end fine-tuning speedup of DRAFT with FDA, compared to baseline hardware with Full Fine-tuning, LoRA, BitFit, and Partial Tuning (all normalized to Full Fine-tuning running on baseline hardware).

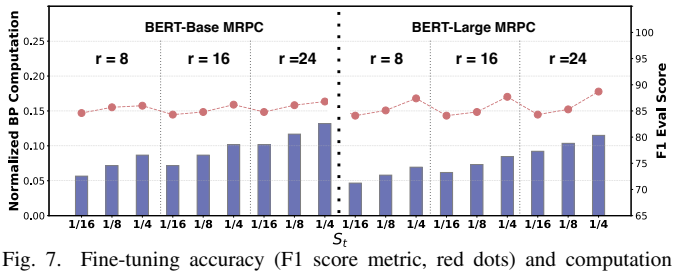


Fig. 7. Fine-tuning accuracy (F1 score metric, red dots) and computation reduction versus various (r, S_t) configurations in FDA.

C. Reconfigurable PE Array

Dual Workflow for Dynamic Fine-tuning Pathway. As illustrated in Fig. 4(c), to fully exploit data parallelism across different fine-tuning pathways, the RPEA can be reconfigured to operate in two distinct workflows: dense GEMM mode and sparse ternary GEMM mode. In the backbone and adapter pathways, all matrix computations are performed in dense mode, utilizing a weight-stationary workflow (shown in Fig. 4(c) left). Weights are pre-loaded into the row buffer and then unicast to each PE, while input data is unicast through a column router to each PE. In contrast, the BPN pathway requires sparse-to-dense gather operations, with each gather unit assigned unique sparsity masks. Sparse ternary matrix multiplication is performed here using an input-stationary workflow (shown in Fig. 4(c) right). Sparse input data is pre-loaded into a column buffer and broadcast to each gather unit, while pre-determined dense ternary sign bits and butterfly control signals are distributed to each gather unit horizontally.

Mixed-Precision Floating-Point PE. The architecture and input format of MPPE is illustrated in Fig. 4(d). The DRAFT fine-tuning process involves three types of matrix operations corresponding to the three modes of the MPPE. *Mode* 0: FP8×FP8, used in the trainable adapter pathway and attention mechanism of the Transformer block; *Mode* 1: FP8×FP4, applied in the pre-trained backbone pathway; *Mode* 2: FP8×Ternary, utilized in the BPN pathway. In Mode 0 and 2, both inputs, IN0 and IN1, are FP8. In Mode 1, IN1 is split into two FP4s to increase the throughput. Each PE receives two FP4 weights corresponding to two output channels, and the adjacent PEs compute the same pair of output channels, exchanging partial results through an inter-PE connection, as illustrated by the purple datapath. The MPPE shares mantissa multipliers and exponent adders across these modes by splitting the MSB and LSB computations.

VI. Evaluation

A. Algorithm Evaluation

Settings: To evaluate the fine-tuning accuracy of FDA, we fine-tuned BERT-Base and BERT-Large [2] models on the MRPC, SST-2, and RTE datasets from GLUE Benchmark [29]; ViT-B/16 model [4] on the CIFAR-100 dataset [30]; and GPT-2 Medium model [31] on the E2E task [32]. Regarding the end-to-end metric, we report the F1 for MRPC, the BLEU for E2E, and the accuracy for the others. We used Full-Finetuning (FT), Bias-Only [8], LoRA, and Partial-Tuning [23] under unified MXFP8 precision as baseline fine-tuning methods. FDA adopts a mixed-precision quantization as introduced in Sec. IV-B. For a fair comparison, the adapter ranks in LoRA, Partial-Tuning, and FDA are set to 16, and all fine-tuning methods have the same hyperparameter setting.

Performance Evaluation: Fig. 5 shows the overall comparison of accuracy metric and normalized BP operations across different fine-tuning methods with S_t in FDA set to 1/4. FT achieves the best accuracy across most tasks but incurs the heaviest BP computational load. LoRA outperformed FT on certain tasks and reduced BP computation to nearly half of FT. The Bias-Only achieved a lighter BP overhead by eliminating weight gradient computation, and Partial-Tuning approached FDA-level reductions by tuning the last few layers. However, both methods exhibited noticeable accuracy losses. In contrast, the FDA maintained an accuracy loss within 1% compared to FT, demonstrating robust performance while reducing BP computation to below 15%. This highlights the effectiveness of the FDA in improving the efficiency of edge fine-tuning.

TABLE I
AREA AND POWER BREAKDOWN OF DRAFT ACCELERATOR (500MHZ)

Module	Area (mm^2)	Power (mW)
RPEA - MPPE & Adder Tree	6.51	729.45
RPRA - Sparse Gather Unit	1.65	142.91
RPEA - Others	1.25	100.28
Vecer Engine	2.01	142.87
SRAM Buffer	2.86	146.06
Others	0.47	26.15
Total	**14.74**	**1287.72**

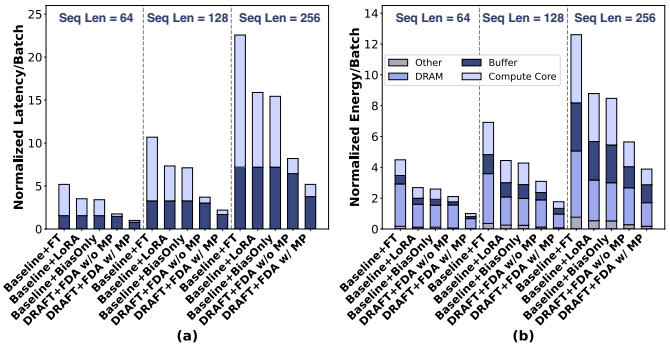


Fig. 8. (a) Latency and (b) Energy breakdown over different hardware with various fine-tuning algorithms for BERT-Base with different batch sequence lengths.

Fig. 7 shows the sensitivity analysis of S_t and r, conducted on BERT-Base and BERT-Large (MRPC). The result indicates that the FDA is more sensitive to BPN sparsity. Notably, when S_t reaches 1/16, a significant accuracy loss is observed, especially in the BERT-Large model. Moreover, a lower S_t incurs additional Sparse Gather unit overhead in hardware implementation. To balance hardware and algorithm efficiency, we keep S_t=1/4 for the following hardware evaluations.

B. Hardware Evaluation

Baseline Setting: We design the baseline hardware based on the SoTA systolic accelerator [18] dedicated to Transformer inference and fine-tuning, which can naturally run all the baseline fine-tuning methods. The baseline hardware is configured to match DRAFT in FP8 compute throughput and memory capacity under the same technology and frequency.

Hardware Implementation: We perform RTL design for the DRAFT and baseline accelerator and synthesized them using Synopsys Design Compiler with a 28nm CMOS technology. The entire DRAFT system operates at 500MHz, with on-chip buffers utilizing foundry-provided SRAM. Power consumption and area results were estimated from the synthesis reports. Additionally, we developed a cycle-accurate simulator and verified it against the RTL implementation. The simulator generates DRAM access traces, used with Ramulator [33] to evaluate DRAM access latency, and DRAMPower [34] to capture the corresponding DRAM power consumption.

Speedup: Fig. 6 illustrates the speedup of DRAFT compared with the baselines. We report both the speedup in the BP phase and the overall fine-tuning speedup. Full-Finetuning, LoRA, Bias-Only, and Partial-Tuning run on the baseline hardware, while FDA runs on DRAFT. During the BP phase, DRAFT achieved an average speedup of 14.1× and 8.3× compared to FT and LoRA running on baseline hardware, respectively. This improvement stems from the N:M sparse ternary BPN, where computational reductions at the algorithmic level align effectively with hardware efficiency. The fine-tuning end-to-end speedup is lower than that of the BP as the DRAFT is tailored to reduce the cost of the BP. DRAFT achieves an average overall speedup of 4.9× by leveraging mixed-precision quantization and the reconfigurable MPPE to improve the throughput of the FF phase.

Hardware Overhead: The hardware area and power consumption of DRAFT after synthesis are shown in Table I. The total power and area are $1.29W$ and $14.74mm^2$, respectively. The MPPE in the RPEA and the buffer significantly contribute to both area and power.

Latency and energy breakdown: Fig. 8 presents the fine-tuning latency and energy breakdown of each batch for BERT-Base with three typical sequence lengths across different fine-tuning algorithms and accelerator hardware. As shown in Fig. 8(a), DRAFT significantly lowers the computational latency of the BP phase, reducing batch latency close to the level of executing only the FF phase. Combined with Mixed-Precision (MP) optimization, DRAFT further reduces latency across all fine-tuning phases.

Fig. 8(b) illustrates the energy breakdown across the compute core, buffer, off-chip memory access, and other modules. By leveraging the FDA, DRAFT achieves 4.2× and 2.3× improvements in energy efficiency compared to FT and LoRA on the baseline hardware, respectively. Based on the algorithm-hardware co-optimization, DRAFT takes full advantage of the parallelism of each fine-tuning pathway, significantly compressing fine-tuning latency and reducing power consumption. Simultaneously, enhanced throughput and improved data reuse alleviate the repeated loading of the substantial weights and batch inputs, thereby lowering the power consumption associated with the expensive on-chip and off-chip data access. In summary, DRAFT significantly improves the throughput and power efficiency of Transformer fine-tuning by fully reducing the BP computational overhead.

VII. CONCLUSION

We propose DRAFT, an algorithm-hardware co-design framework for efficient transformer fine-tuning on edge. The proposed Feedback Decoupling Approximation (*FDA*) algorithm decouples the BP through the backbone weight into the adapter pathway and sparse ternary Bypass Network (*BPN*) pathway. Two lightweight pathways jointly approximate the information of pre-trained weight, uniformly lightening the BP computation overhead for input gradient and weight gradient. An efficient DRAFT accelerator with reconfigurable workflows and sparse gather networks is designed to support fine-tuning with the FDA. Experimental results demonstrate that DRAFT achieves an average speedup of 4.9× and an energy efficiency improvement of 4.2× across multiple fine-tuning tasks with negligible accuracy loss.

REFERENCES

[1] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," May 2019.

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," Jul. 2019.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[5] V. Janfaza, S. Mandal, F. Mahmud, and A. Muzahid, "Ada-gp: Accelerating dnn training by adaptive gradient prediction," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1092–1105.

[6] H. Shao, J. Lu, M. Wang, and Z. Wang, "An Efficient Training Accelerator for Transformers With Hardware-Algorithm Co-Optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 11, pp. 1788–1801, Nov. 2023.

[7] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. D. Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-Efficient Transfer Learning for NLP," in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 2790–2799.

[8] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 11 285–11 297.

[9] J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian, "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection," Mar. 2024.

[10] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," Oct. 2021.

[11] Y. Li, Y. Yu, C. Liang, P. He, N. Karampatziakis, W. Chen, and T. Zhao, "LoftQ: LoRA-Fine-Tuning-Aware Quantization for Large Language Models," Nov. 2023.

[12] Y. L. H. Y. R. G. X. Z. S. R. B. Y. K. Z. Y. C. L. Zhongzhi Yu, Zheng Wang, "Edge-llm: Enabling efficient large language model adaptation on edge devices via layerwise unified compression and adaptive layer tuning voting," 2024.

[13] Z. Hong and C. P. Yue, "Efficient-grad: Efficient training deep convolutional neural networks on edge devices with grad ient optimizations," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 2, pp. 1–24, 2022.

[14] D. Han, J. Lee, and H.-J. Yoo, "Df-lnpu: A pipelined direct feedback alignment-based deep neural network learning processor for fast online learning," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 5, pp. 1630–1640, 2021.

[15] M. Giordano, K. Prabhu, K. Koul, R. M. Radway, A. Gural, R. Doshi, Z. F. Khan, J. W. Kustin, T. Liu, G. B. Lopes, V. Turbiner, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, G. Lallement, B. Murmann, S. Mitra, and P. Raina, "Chimera: A 0.92 tops, 2.2 tops/w edge ai accelerator with 2 mbyte on-chip foundry resistive ram for efficient training and inference," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2.

[16] L. Zhu, L. Hu, J. Lin, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "Pockengine: Sparse and efficient fine-tuning in a pocket," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1381–1394.

[17] K. Prabhu, R. M. Radway, Y. Jeffrey, K. Bartolone, M. Giordano, F. Peddinghaus, Y. Urman, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, S. Mitra, and P. Raina, "Minotaur: An edge transformer inference and training accelerator with 12 mbytes on-chip resistive ram and fine-grained spatiotemporal power gating," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2024, pp. 1–2.

[18] J. Yu, K. Prabhu, Y. Urman, R. M. Radway, E. Han, and P. Raina, "8-bit Transformer Inference and Fine-tuning for Edge Accelerators," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. La Jolla CA USA: ACM, Apr. 2024, pp. 5–21.

[19] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[20] T. H. Moskovitz, A. Litwin-Kumar, and L. Abbott, "Feedback alignment in deep convolutional networks," *arXiv preprint arXiv:1812.06488*, 2018.

[21] Q. Liao, J. Leibo, and T. Poggio, "How important is weight symmetry in backpropagation?" in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[22] J. Launay, I. Poli, F. Boniface, and F. Krzakala, "Direct feedback alignment scales to modern deep learning tasks and architectures," *Advances in neural information processing systems*, vol. 33, pp. 9346–9360, 2020.

[23] R. Zhang, J. Han, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, P. Gao, and Y. J. Qiao, "Llama-adapter: Efficient fine-tuning of language models with zero-init attention," *ArXiv*, vol. abs/2303.16199, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257771811

[24] Y.-L. Sung, J. Cho, and M. Bansal, "Lst: Ladder side-tuning for parameter and memory efficient transfer learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 991–13 005, 2022.

[25] S. Liu, P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu, and M. Liu, "16.2 a 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 250–252.

[26] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1573–1586, 2022.

[27] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf *et al.*, "Microscaling data formats for deep learning," *arXiv preprint arXiv:2310.10537*, 2023.

[28] S. Q. Zhang, T. Tambe, N. Cuevas, G.-Y. Wei, and D. Brooks, "CAMEL: Co-Designing AI Models and Embedded DRAMs for Efficient On-Device Learning," Dec. 2023.

[29] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[30] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[31] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[32] J. Novikova, O. Dušek, and V. Rieser, "The e2e dataset: New challenges for end-to-end generation," *arXiv preprint arXiv:1706.09254*, 2017.

[33] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[34] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www. drampower. info*, vol. 22, 2012.

INFSCALER: Enabling Efficient ML Inference Serving on Multi-Accelerator Edge Devices via Asymmetric Auto-Scaling

Borui Li, Tiange Xia, Shuai Wang, Shuai Wang✉
School of Computer Science and Engineering, Southeast University, Nanjing, China
Email: {libr, xia_tg, shuaiwang_iot, shuaiwang}@seu.edu.cn

***Abstract*—Nowadays, there is a growing trend to deploy machine learning (ML) models on edge devices. To cope with the increasing resource requirements of current ML models, multi-accelerator edge devices that integrate CPU, GPU, NPU, or TPU in a single SoC gain popularity. However, we observe that existing ML inference serving frameworks are poor in utilizing the unique hardware architecture of these edge devices. In this paper, we present INFSCALER, an efficient ML inference serving framework tailored for multi-accelerator edge devices. INFSCALER discovers the architectural bottleneck of ML models and designs a bottleneck-aware asymmetric auto-scaling technique to facilitate efficient resource allocation for ML models on the edge. Furthermore, INFSCALER capitalizes on the hardware's unified memory feature inherent to edge devices, ensuring efficient data sharing between the asymmetrically scaled model partitions. Our experimental results show that INFSCALER achieves up to 126.59% throughput improvement and 27.32% resource reduction while satisfying the latency requirements compared with the state-of-the-art inference serving approaches.**

I. INTRODUCTION

Recently, deploying machine learning (ML) models on edge computing devices rather than the cloud has gained significant attention [1], [2], [3]. Building on this momentum, various merchants such as NVIDIA [4], AMD [5], and Google [6] present multi-accelerator edge devices to provide better ML inference performance. For example, the NVIDIA Jetson series leverages System-on-Chip (SoC) architecture that integrates a CPU, GPU, and NVIDIA Deep Learning Accelerator (NVDLA) onto a single chip. These multi-accelerator devices are now widely used in typical edge environments such as autonomous driving [7], [8] and edge video analytics [9], [10].

Nevertheless, the energy and space limitation of edge environments makes it hard to match the increasing resource demand of modern ML algorithms such as large language models (LLMs). Prior attempts for efficient inference serving on resource-constrained edge devices leverage model quantization [11], [12], pruning [13] or sparsity [14]. These approaches require modifying or retraining the original model, which will lead to an accuracy drop or excessive training overhead.

If we refer to ML serving approaches without modifying the model, which are mostly designed for the cloud, the primary solutions are request batching [15], [16] and instance auto-scaling [17]. The ingress requests are organized into batches to leverage the parallel processing capabilities of hardware accelerators to improve efficiency. Once the request bursts and results in the resource usage of an inference instance exceeding a pre-defined resource cap, the cloud server will scale out, i.e., spawn new serving instances, for higher parallelism.

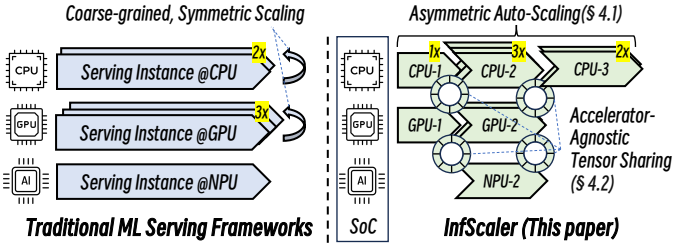


Fig. 1. Illustration of the difference between INFSCALER and existing works.

However, directly adopting these cloud-based solutions is problematic for edge environments because they do not take the unique hardware characteristics of edge devices into consideration. (1) Existing works [15], [16], [17] replicate the *whole ML model* to serve additional requests when scaling out, but edge devices may not be facilitated with enough resources for this coarse-grained scaling. For example, inferencing the `gpt2-medium` model requires ∼6GB memory. While NVIDIA Xavier NX, a typical edge device, is only equipped with 8GB which could not scale the whole model out for bursty requests. (2) Existing ML serving frameworks [18] treat accelerators such as GPU and NPU as peripherals with independent memory management. However, there is a new opportunity for multiple accelerators to cooperate more tightly due to the hardware unified memory design at the edge.

Towards the above problems, we present INFSCALER, an efficient ML inference serving framework tailored for edge devices. Specifically, INFSCALER aims to achieve better inference serving performance (i.e., throughput) while satisfying the latency constraint of requests under a given resource limitation. Considering the multi-accelerator architecture and resource-constrained nature of edge devices, the key idea of INFSCALER is scaling each group of ML operators, i.e., the building blocks of an ML model, with different scaling ratios for better resource efficiency. Unlike symmetrically scaling all operators as a whole model, we call this approach *asymmetric auto-scaling*, as shown in Fig. 1. This approach is based on our observation (cf. §II-B) that ML models inherently contain architectural bottlenecks due to the resource demands of ML operators and their affinity towards accelerators, which negatively impact overall inference performance. Selectively scaling out the bottlenecks can achieve similar, even better,

We thank all the reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China under Grant 62302096, 62272098, and U24B20152; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20230813; and in part by the Zhishan Young Scholar Program of Southeast University under Grant 3209002402A2. Shuai Wang is the corresponding author.

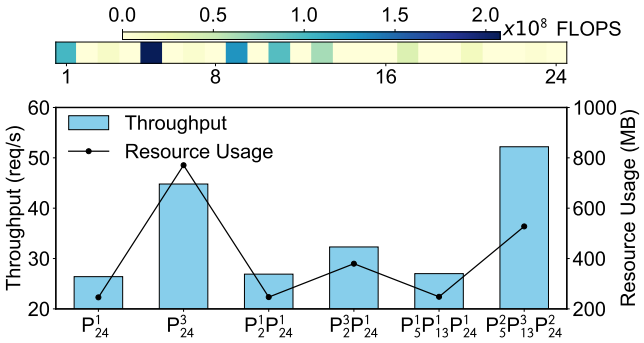


Fig. 2. Illustration of the computation complexity of AlexNet (top) and its inference performance under different settings (bottom). P_x^a denotes the model is partitioned at the end of operator #x and scaled out to a instances.

performance compared to whole model replication.

To facilitate asymmetric auto-scaling, we first identify the bottlenecks through offline profiling and partition the ML model into several groups of operators with fully aware of the bottleneck. At runtime, we perform the asymmetric auto-scaling and also take advantage of the hardware unified memory between heterogeneous accelerators to achieve efficient *accelerator-agnostic tensor sharing* between the partitioned models. Our contributions can be summarized as follows:

- We discover the architectural bottleneck of ML models and its negative impact on the inference performance. Based on this, we propose a *bottleneck-aware asymmetric auto-scaling* strategy to utilize the precious resources on the edge wisely for better performance.
- We advocate an *accelerator-agnostic tensor sharing* mechanism by taking advantage of the hardware unified memory property on edge devices.
- We implement INFSCALER and evaluate its performance extensively. Results show that INFSCALER can improve the inference throughput by 34.78%-126.59% and reduce 10.49%-27.32% resource usage while satisfying the latency constraints.

II. BACKGROUND AND MOTIVATION

A. Hardware Architecture of Edge Devices

Nowadays, edge devices are equipped with AI accelerators like GPUs, TPUs, and FPGAs to enhance performance while keeping energy efficiency in ML processing. The most prevalent edge ML devices are NVIDIA Jetson series, which are widely used in autonomous driving, VR/AR, and robotics. These devices include three generations identified as NVIDIA Jetson, Xavier and Orin. For example, on NVIDIA AGX Xavier, an 8-core Armv8.2 CPU, an integrated NVIDIA Volta GPU (iGPU), and two NVDLA accelerators are located on the same SoC, which is different from the desktop and servers.

Unified memory is originally a software concept in CUDA programming aimed at simplifying CPU RAM and dGPU VRAM management through unified addressing. NVIDIA Jetson series leverages a *hardware unified memory* architecture where all the accelerators share a common hardware main memory. From the second generation (Xavier) on, hardware cache coherence between CPU and iGPU further accelerates memory sharing. This distinct feature provides a promising starting point for our data-plane acceleration for serverless inference on edge devices.

B. Architectural Bottleneck of ML Models

ML bottlenecks. ML models include multiple computation operators such as convolution, pooling, and normalization. However, the computational complexity disparity among operators and their affinity towards heterogeneous accelerators results in some of them serving as a bottleneck of the whole model. To examplify, we use a representative ML model, AlexNet, and study its inference performance on the NVIDIA Xavier AGX multi-accelerator edge device in Fig. 2.

> **Observation 1:** ML models inherently contain bottlenecks due to the unbalanced distribution of the computation complexity, which hinders the overall inference performance.

We can see from the figure that certain operators (e.g., #1, #5, #13) exhibit notably higher complexity compared to others, resulting in their computation time being prominently longer than other parts, which lowers the overall inference throughput. We identify these layers are the "bottleneck" of AlexNet. To make things worse, the traditional auto-scaling mechanism of ML serving directly scales the whole model when confronted with burst requests. However, on resource-constrained edge devices, this coarse-grained scaling mechanism uniformly allocates additional resources to all layers, potentially resulting in a waste of precious resources. *Consequently, a novel fine-grained scaling approach is imperative for serverless ML inference on edge devices.*

Impact of asymmetric auto-scaling. Based on the above observation, we further investigate how to deal with the architectural bottleneck of ML models for better inference. We quantify the inference throughput and resource usage under the same predefined resource cap. Results in Fig. 2 show that $P_5^2P_{13}^3P_{24}^2$ achieves the best performance among the six configurations. It exhibits 16.5% throughput improvement against the second-performed baseline P_{24}^3, i.e., no partition and scaling to three instances, with 31.5% less resources used. The key takeaway of this improvement is this asymmetric scaling can leverage wiser resource usage, which is allocating more resources to the bottlenecks instead of distributing them equally. Moreover, this improvement is also fragile. Also, from the results of other partitioned baselines, we can see that it requires a prudent model partition and scaling policy to achieve optimal performance.

> **Observation 2:** Asymmetric auto-scaling, i.e., selectively scaling out the bottleneck with an appropriate ratio, can enhance the throughput performance with the same resources.

The above results demonstrate the potential for fine-grained, asymmetric auto-scaling on edge devices. *However, how to wisely partition the model considering bottleneck and scales out for optimized performance remains unsolved.*

III. INFSCALER OVERVIEW

A. Design Principles

Throughout our system design of INFSCALER, we adhere to the following guiding principles. **(1) User-transparent.** As an ML inference serving framework, INFSCALER should ensure users remain unaware of underlying complexities such

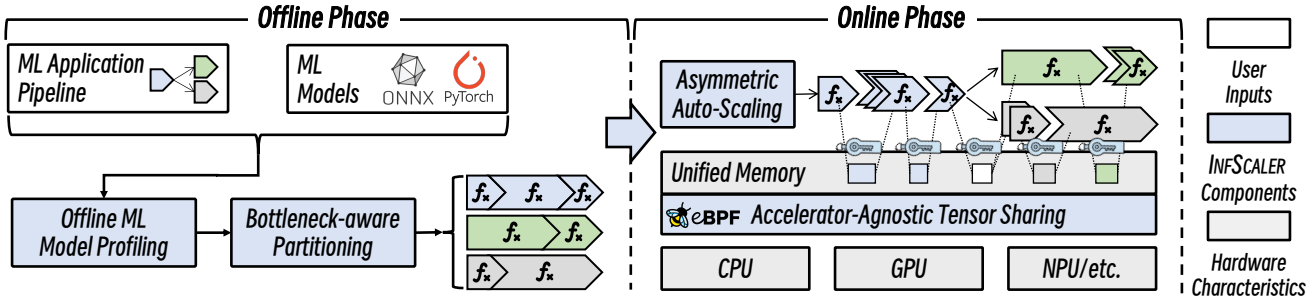


Fig. 3. System overview of INFSCALER framework.

as model partitions and fine-grained scaling mechanisms. Additionally, INFSCALER should maintain model accuracy without altering its structure to achieve this transparency. **(2) Hardware-oriented.** INFSCALER targets the ML inference on edge devices facilitated with heterogeneous accelerators such as the NVIDIA Xavier series. Hence, we should fully utilize the hardware characteristics of edge devices by design to achieve better performance. **(3) Throughput-aware.** Maximizing throughput ensures INFSCALER achieves an efficient usage of the limited resources on edge devices to process a high volume of concurrent requests. Hence, we design INFSCALER to maximize the throughput while satisfying the latency service level objective (SLO) specified by users.

B. INFSCALER *in a nutshell*

Fig. 3 depicts the bird's-eye view of INFSCALER system. It has an offline phase to generate the optimized function deployment and an online phase to further improve the performance.

Offline deployment optimization. In the offline phase, users submit their ML application to be deployed and the corresponding ML models. Similar to the existing practices such as the inference graph of KServe [19] and SageMaker Pipeline of AWS [20], INFSCALER supports submitting an ML application including multiple ML algorithms that formulate a *pipeline*. For example, an NLP pipeline needs to first run document classification, and then perform named entity detection downstream based on the previous classification results. As for ML models, INFSCALER do not require any modification to ML models and users can directly submit the *unmodified* ML models such as ONNX or PyTorch. One of the main design principles of INFSCALER is to maximize the inference throughput while satisfying the latency constraint. Hence, it is necessary to address the architectural bottlenecks of ML models during inference. A naïve idea is to allocate more resources to the bottleneck layers for better performance. However, traditional ML serving frameworks cannot precisely allocate additional resources only to the bottleneck because they leverage a coarse-grained scaling mechanism. Hence, to obtain the opportunity for precise resource allocation, INFSCALER first profiles for detailed performance metrics (e.g., FLOPs, execution time) and partitions the ML model into slices for asymmetric auto-scaling.

Online ML inference serving. During the online serving process, INFSCALER optimizes the inference performance in two ways. First, INFSCALER leverages asymmetric auto-scaling in correspondence to the ingress request variations. With the help of bottleneck-aware partitioning, INFSCALER can only scale up the bottleneck part of the model to achieve better resource efficiency. Second, INFSCALER achieves systematic support for multi-accelerator edge devices by allowing the asymmetric auto-scaling to scale part of the model to all accelerators. Furthermore, INFSCALER designs an accelerator-agnostic tensor sharing mechanism that leverages the hardware unified memory to efficiently share outputs between model partitions.

IV. DESIGN OF INFSCALER

A. Asymmetric Auto-Scaling of ML models

As we demonstrated in §II-B, ML models include (multiple) architectural bottlenecks that impede inference efficiency. In this section, we show our detailed designs about how to achieve asymmetric auto-scaling while fully aware of the bottlenecks of ML models. We first detect the bottleneck and figure out the optimal scaling-out policies. After that, we move on to the resource allocation problem with a given partition because the allocation of CPU and memory poses a significant influence on the overall performance.

Offline bottleneck-aware partitioning. Designing a wise partition algorithm requires us to consider the following trade-off, where more partitions lead to an excessive overhead for inter-partition communication but fewer partitions are not fine-grained enough for efficient resource allocation. According to §II-B, we find that computation complexity can be a source of inference bottlenecks. To address the partition granularity and overhead tradeoff, the main idea of INFSCALER's partitioning is to balance the computation complexity among the operators in a partition. The intuition here is each partition is also a minimum resource allocation unit. Operators inside a partition scale up and down at the same time, which means the resources are allocated evenly in a partition. If the partition contains several operators with massive different complexities, the largest operator will further become the new bottleneck in the partition that impedes the overall performance.

Therefore, our bottleneck-aware partitioning problem could be formulated as an optimization problem that minimizes the variance of the computation complexity of each group while keeping the latency satisfies the user-specified SLO:

$$\arg\min_{P_1,\cdots,P_k} \sum_{k=1}^{K} \sigma(P_k)$$
$$\text{s.t.} \begin{cases} \hat{T_o} + (K-1)T_p \leq SLO \\ P_k = \{c_n | n \in [p_{k-1}+1, p_k], c_n > \overline{C_N}\} \\ P_1 \cup \cdots \cup P_K = C_N \end{cases}, \quad (1)$$

Algorithm 1: Asymmetric auto-scaling policy

Input: $\mathcal{B} = \{B_1, \cdots, B_K\}$: set of bottleneck indicator;
$A_k^m \in \mathcal{A}_k$, u_k^m: hardware affinity and expected resource usage of partition k on accelerator m;
U^m, R^m: resource used and limit of accelerator m;
Output: Number of scaled instance $\mathcal{S} = \{S_1, \cdots, S_K\}$

1 **foreach** *SLO unsatisfaction detected* **do**
2 $\quad k^* \leftarrow \arg\max_{k=1}^{K} B_k, B_k^* \leftarrow \max_{k=1}^{K} B_k$;
3 $\quad$ **foreach** $A_{k^*}^m \in sort(\mathcal{A}_{k^*})$ **do**
4 $\quad\quad$ **if** $U^m \leftarrow U^m + r_{k^*}^m < R^m$ **then**
5 $\quad\quad\quad B_{k^*} \leftarrow \frac{B_{k^*} S_{k^*}}{S_{k^*}+1}$;
6 $\quad\quad\quad U^m \leftarrow U^m + r_{k^*}^m$; $S_{k^*} \leftarrow S_{k^*} + 1$;
7 $\quad\quad\quad$ **break**;

which partitions an ML model with N operators into K partitions. $\sigma()$ stands for the calculation of standard variance. P_k and p_k are the computation complexity set and the index of the last operator of partition k. $c_n \in C_N$ is the computation complexity of n-th operator. $\hat{T}_o$ means the predicted model execution time, T_p denotes the overhead introduced by partitioning. Instead of using all layers, INFSCALER only considers the representative layers whose complexity is bigger than the model average in this optimization (constraint #1). This is because compared with the bottleneck layers, these small layers have a very limited impact on the throughput and resource allocation.

Online asymmetric auto-scaling policy. By "asymmetric," we mean that (1) the scaling factors of each partition can differ from one another, and (2) partitions can be scaled to different types of accelerators.

Algorithm 1 depicts the auto-scaling policy. Once detected the serving performance cannot meet the user-specified latency SLO, INFSCALER starts auto-scaling. First, we identify the partition k with the maximum *bottleneck indicator* B_k (line 2), which is defined as the sum of the computation complexity of each operator in the partition $\sum P_k$, serving as the target for scaling out. This is because the partition with the highest B_k has the greatest computational complexity and thus requires scaling first. Then, INFSCALER determines on which accelerator to scale partition k with the help of the profiled hardware affinity, e.g., execution latency, and checks if the accelerator has enough resources to accommodate it (lines 3, 4). If yes, we reduce the bottleneck indicator of partition k to adjust its future priority in scaling (line 5), record the resource usage, and finally perform scaling (line 6). With this asymmetric auto-scaling policy, INFSCALER dynamically scales out the partition that needs more resources the most, which leads to better resource efficiency and inference performance.

B. Accelerator-Agnostic Tensor Sharing

Although asymmetric auto-scaling brings intrinsic performance gains, it also incurs more Inter-partition data transmission. Therefore, it is imperative to meticulously devise a systematic data flow design for sharing the input and output tensors of each function.

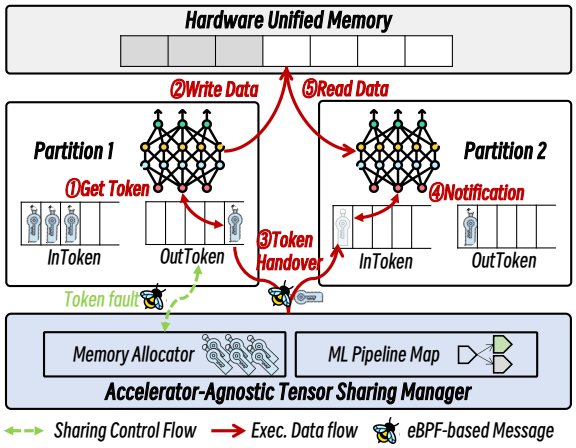


Fig. 4. Illustration of the accelerator-agnostic tensor sharing.

To improve data sharing efficiency, numerous works [21], [17] leverage the zero-copy technique for tensor sharing. Zero-copy means transferring solely an address pointer to shared data instead of transferring all the data itself. Nonetheless, these prior works of zero-copy data sharing devote less attention to the following two unique problems in INFSCALER. (1) The asymmetric auto-scaling technique leads the tensor sharing problem to a multi-producer, multi-consumer (MPMC) memory sharing scenario. For example, once a partition of an ML model scales out asymmetrically to multiple instances, its subsequent partition should acquire data from both the scaled instances, i.e., multi-consumer, and its preceding partition should share data with both instances, i.e., multi-consumer. For example, when a partition of an ML model scales out asymmetrically across multiple instances, it acts as multiple data producers to distribute data to subsequent partitions. by acquiring data from all involved instances. Conversely, its preceding partition also needs to share data to multiple consumers. (2) To make things worse, the data producers and consumers can reside on heterogeneous accelerators. Hence, INFSCALER proposes an *accelerator-agnostic tensor sharing* mechanism to address the aforementioned problems.

eBPF-based event-driven sharing. Towards the MPMC zero-copy sharing problem and preventing the data race between multiple instances, INFSCALER adopts an event-driven sharing mechanism. Specifically, to share the data between instances, the data producer should acquire a token (i.e., memory address) from INFSCALER before it can proceed with writing data. After completing the write operation, the producer then sends this token to the consumer as an event to trigger the reading.

This token-based sharing has two merits. On the one hand, compared with the widely-used polling-based zero-copy sharing approach [17], it reduces the frequency of memory access and achieves lower runtime overhead which is suitable for edge environments. On the other hand, it transforms the ad-hoc usage of shared memory into centralized management, which gives us the chance to holistically manage the load balancing among different scaled instances on heterogeneous accelerators. We further optimize the centralized scheduling overhead by leveraging eBPF for all the control flow communications. An eBPF map is introduced to intercept the

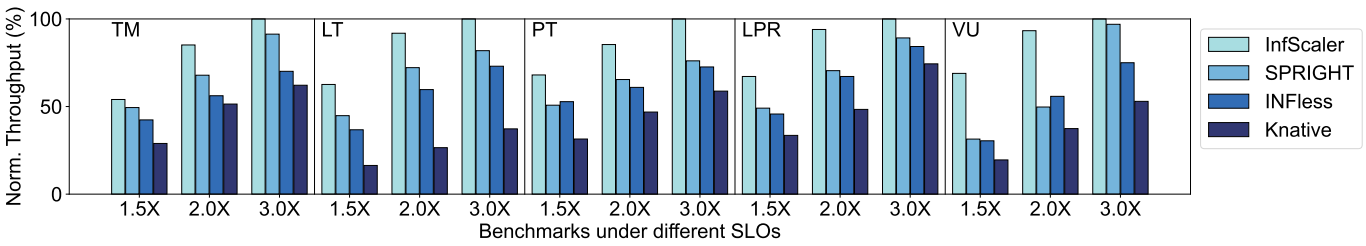


Fig. 5. Comparison of the normalized throughput of INFSCALER under different latency SLO settings against baselines. Here, 1.5× denotes that the SLO is set to 1.5 times of its normal inference time, which we refer to as high pressure. Similarly, 2.0× and 3.0× stands for medium and low pressures, respectively.

token fault for sidecar control flow and the token handover for execution control flow of the inter-partition data sharing. Here raises a new question: why not implement the memory sharing manager directly in an eBPF function? This is because eBPF functions reside in the kernel, which poses strict constraints on execution time. If we push all the logic in the memory sharing manager into an eBPF function, it incurs much overhead for the kernel and may expand the attack surface. Moreover, eBPF functions are not adaptive to the function deployment changes. As INFSCALER is a general ML serving system, applications deployed on top of our framework are versatile. Implementing the memory sharing manager in a separate service in user space is more adaptive for ever-changing applications.

Lazy sharing mechanism. Towards the data sharing between heterogeneous accelerators, we implement the above zero-copy mechanism on the hardware unified memory which all accelerators can access directly. Furthermore, INFSCALER adopts a lazy allocation mechanism to prevent excessive allocations of limited hardware unified memory on edge devices. Lazy means instead of allocating all the shared memory at once, INFSCALER allocates the memory when instances report a memory shortage. We argue that this approach will reach a dynamic balance of the input and output between consecutive partitions.

The whole tensor sharing process is illustrated in Fig. 4. The tensor sharing manager handles the overall new memory allocation and inter-instance token exchange. When partition 1 tends to share the output with partition 2, it first checks if the `OutToken` queue is empty. If there is a token, it writes data to the corresponding region and hands over the token to the consumer. If the `OutToken` is empty, partition 1 will send a token fault to the tensor sharing manager. The sharing manager will select an idle memory region and send its address back for further usage.

V. EVALUATION

A. Experimental Setup

Testbed. We conduct the experiments on the NVIDIA Xavier AGX development board, which is a widely used multi-accelerator edge device.

Baselines. We compare INFSCALER against the following baselines: (1) *INFless* [15]: An ML inference serving framework that leverages auto-scaling and batching to improve resource efficiency. (2) *SPRIGHT* [17]: A state-of-the-art application serving framework that focuses on efficient memory sharing mechanism for pipelined applications. (2) *Knative* [18]: A widely-used open-source ML serving system based on Kubernetes. We apply our partitioned and configured functions on Knative without the hardware-oriented memory sharing to see how it performs.

TABLE I
APPLICATION BENCHMARKS USED FOR EVALUATION.

Benchmark	ML Pipeline	Involved Models
Traffic Monitoring (TM)	❶→❷	Vehicle detection(❶SSD), Vehicle model recognition(❷ShuffleNet)
Language Translation (LT)	❶→❷→❸	Semantic feature extraction(❶Transformer), Cross-language conversion(❷Transformer), Target statement decoding(❸Transformer)
Person Tracking (PT)	❶→❷	Human detection(❶TinyYOLOv3), Person identity finding(❷VGG)
License Plate Recognition (LPR)	❶→❷	Image super-resolution(❶ResNet100), Character recognition(❷MobileNet)
Video Understanding (VU)	❶→❷	Video content analysis(❶Inception), Content Understanding(❷BERT)

Benchmarks. We excerpt five production ML applications from [15], [22] as benchmarks in our evaluation, as shown in Table I. Furthermore, to evaluate INFSCALER under real-world workloads, we replay the Microsoft Azure Functions (MAF) dataset [23] which records approximately 46,000 requests of Azure cloud services.

B. Evaluation Results

Throughput optimization. Fig. 5 shows the results of how INFSCALER optimizes the model inference throughput compared with existing approaches. We can see from the result that INFSCALER achieves 34.78%, 48.14%, and 126.59% average improvement on throughput compared with SPRIGHT, INFless, and Knative. We can also observe that INFSCALER perfoms better in harsh workloads. For example, under the low workload pressure (SLO=3.0x), INFSCALER achieves 23.87% throughput improvement against the best-performed counterparts, SPRIGHT. When it comes to high workload pressure (SLO=1.5x), INFSCALER can achieve 40.82% performance gain. This is because higher pressure makes efficient resource usage more critical; even slight inefficiencies can lead to substantial performance degradation.

We then elaborate more on the different improvements against different approaches. The most noticeable improvement is when compared with Knative and INFless. INFless leverages batch mechanism to improve resource efficiency, but its coarse-grained auto-scaling hinders the performance on edge devices. On the other side, SPRIGHT achieves the best performance among the baselines. As we all know SPRIGHT does not include a partitioning algorithm, it focuses on the

TABLE II
RESOURCE USAGE (MEMORY IN MB) COMPARISON AGAINST BASELINES UNDER THE REAL-WORLD MAF WORKLOAD [23].

Benchmark	INFSCALER	SPRIGHT	INFless	Knative
TM	**7570.85**	9732.45	8489.23	10416.96
LT	**13808.67**	14735.2	15848.44	N/A
PT	**10935.03**	13979.08	12216.01	13383.16
LPR	**5573.82**	6882.41	7007.03	6993.59
VU‡	**15706.17**	N/A	N/A	N/A

‡ Under the real-world workload, the state-of-the-arts fails to meet the SLO of VU application while INFSCALER succeed.

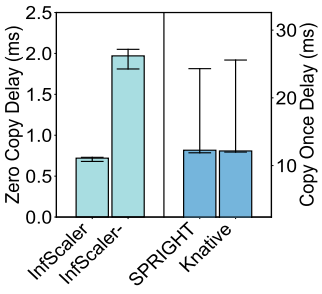


Fig. 6. Tensor sharing overhead against baselines.

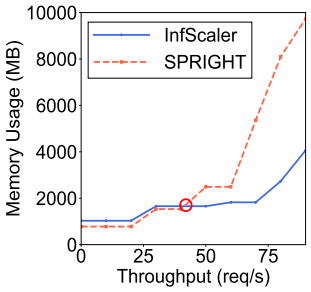


Fig. 7. Latency comparison among INFSCALER and baselines.

efficient data sharing between different serving instances. The ML applications we used as baselines are composed of several steps, which are suitable for SPRIGHT. However, similar to INFless, it also leverages coarse-grained scaling which misses the opportunity to achieve efficient resource usage.

Resource usage efficiency. We then move on to evaluate INFSCALER under real-world workloads and compare the resource usage efficiency against baselines. Table II shows the resource usage, i.e., memory, comparison against baselines. The workload pressure is set to medium, i.e., SLO=2.0x. We can see from the table that INFSCALER achieves 10.49%-27.32% of resource usage reduction against state-of-the-arts. More importantly, INFSCALER is the only framework that finishes the VU application under the SLO constraint. We attribute this improvement to the asymmetric auto-scaling which facilitates an unbalanced but efficient resource usage of different parts of ML models. Furthermore, the second-best approach varies between SPRIGHT and INFless, suggesting that solely focusing on scaling and data sharing may not always yield optimal performance. In addition to asymmetric auto-scaling, INFSCALER also optimizes the tensor sharing overhead, ultimately resulting in improved performance.

Overhead analysis. The overhead of INFSCALER mainly comes from the tensor sharing latency and the additional resources used for supporting multiple instances after partition.

We first evaluate the tensor sharing overhead in Fig. 6. INFSCALER– indicates the latency without our eBPF-based control-plane optimization. We can see that the data transferring overhead is reduced to less than 1 ms with our accelerator-agnostic tensor sharing mechanism, which is neglectable compared with tens or hundreds of milliseconds of inference latency. As for the resource overhead, we present a detailed measurement of the resource usage under different throughputs in Fig. 7. Comparing with SPRIGHT, the best-performed baseline, we observe that at low throughputs, the memory overhead leads to excessive resource usage because SPRIGHT does not include a partitioning strategy. As the throughput increases, the benefit of our asymmetric auto-scaling outweighs the overhead. We also argue that this runtime overhead could be further reduced with the runtime sharing approaches [24], which we identify as the future direction of INFSCALER.

VI. RELATED WORK

INFSCALER builds upon various projects and research. In this section, we summarize the existing works on ML inference serving that shed light on the design of INFSCALER.

To optimize the inference latency and achieve cost efficiency, some existing works use adaptive batching [25], [16], [26] while others focus on how to allocate resources of each ML serving instances [27]. The most related work to INFSCALER is INFless [15]. Instead of "patching" existing ML serving systems such as AWS Lambda [28] or Azure Functions [29], INFless presents a domain-specific inference system leveraging adaptive batching and cold-start mitigation mechanism to support ML inference tasks better. Compared with existing works, INFSCALER adopts an orthogonal view on inference serving, which is optimizing the structural bottleneck of AI models with auto-scaling. Another difference is INFSCALER targets the inference on edge devices that exhibit different hardware characteristics against cloud servers.

Due to the limited resources of edge devices, plenty of research on edge ML inference focuses on reducing resource usage or involving additional computing devices [1]. One of the well-studied optimizations is model pruning [13], [30]. It could reduce the model size and execution time to adapt to edge or IoT devices. For example, DyGNN [13] presents a software-hardware co-optimization approach for efficient GNN pruning and inference. However, model pruning approaches mostly need re-training or sacrifice accuracy limiting its utilization. There are also many works leveraging edge-cloud collaboration to improve inference performance. Several works [31], [32] split the ML model and distribute the ML partitions to edge and cloud for better latency and energy. Orthogonal to the above optimization attempts, INFSCALER takes the advantages of the auto-scaling technique and proposes an asymmetric auto-scaling approach with the help of bottleneck identification to make a wiser usage of limited edge resources.

VII. CONCLUDING REMARKS

This paper presents INFSCALER, a machine learning (ML) inference serving framework designed to address the challenges of inference on edge devices. By identifying architectural bottlenecks within ML models and implementing a bottleneck-aware asymmetric auto-scaling technique, INFSCALER facilitates fine-grained resource allocation on the edge, resulting in significant throughput improvements while adhering to latency SLOs. Leveraging the unified memory characteristic of multi-accelerator edge devices, INFSCALER efficiently reduces the partitioning overhead by an accelerator-agnostic tensor sharing mechanism. The experimental results underscore the effectiveness of INFSCALER by achieving 69.8% throughput improvement on average and up to 27.3% resource usage reduction, showcasing its potential to enhance edge-based ML inference capabilities while meeting stringent performance requirements.

REFERENCES

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] Y. Gong, P. Zhao, Z. Zhan, Y. Wu, C. Wu, Z. Kong, M. Qin, C. Ding, and Y. Wang, "Condense: A framework for device and frequency adaptive neural network models on the edge," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[3] A. Mascolini, S. Gaiardelli, F. Ponzio, N. Dall'Ora, E. Macii, S. Vinco, S. Di Cataldo, and F. Fummi, "Varade: a variational-based autoregressive model for anomaly detection on the edge," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*, 2024.

[4] "NVIDIA Jetson Xavier," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[5] "AMD Zynq UltraScale+ MPSoCs," https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html.

[6] "Google Edge TPU," https://cloud.google.com/edge-tpu.

[7] Y. He, L. Ma, Z. Jiang, Y. Tang, and G. Xing, "Vi-eye: Semantic-based 3d point cloud registration for infrastructure-assisted autonomous driving," in *Proc. of ACM MobiCom*, 2021, pp. 573–586.

[8] S. Shi, J. Cui, Z. Jiang, Z. Yan, G. Xing, J. Niu, and Z. Ouyang, "Vips: Real-time perception fusion for infrastructure-assisted autonomous driving," in *Proc. of ACM MobiCom*, 2022, pp. 133–146.

[9] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. of USENIX NSDI*, 2022, pp. 119–135.

[10] M. Xu, T. Xu, Y. Liu, and F. X. Lin, "Video Analytics with Zero-streaming Cameras," in *Proc. of USENIX ATC*, 2021, pp. 459–472.

[11] Z. Yu, Z. Wang, Y. Li, R. Gao, X. Zhou, S. R. Bommu, Y. Zhao, and Y. Lin, "Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*, 2024.

[12] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: quantum noise-aware training with noise injection, quantization and normalization," in *Proceedings of the 59th ACM/IEEE design automation conference*, 2022, pp. 1–6.

[13] C. Chen, K. Li, X. Zou, and Y. Li, "Dygnn: Algorithm and architecture support of dynamic pruning for graph neural networks," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021.

[14] H. Tang, S. Yang, Z. Liu, K. Hong, Z. Yu, X. Li, G. Dai, Y. Wang, and S. Han, "Torchsparse++: Efficient training and inference framework for sparse convolution on gpus," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 225–239.

[15] Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li, "INFless: A native serverless system for low-latency, high-throughput inference," in *Proc. of ACM ASPLOS*, 2022, pp. 768–781.

[16] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," in *Proc. of IEEE SC*, 2020, pp. 1–15.

[17] S. Qi, L. Monis, Z. Zeng, I.-c. Wang, and KK. Ramakrishnan, "SPRIGHT: Extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing," in *Proc. of ACM SIGCOMM*, 2022, pp. 780–794.

[18] "Knative Project," https://knative.dev/.

[19] "KServe Project," https://github.com/kserve/kserve.

[20] "AWS SageMaker Serverless Endpoint," https://docs.aws.amazon.com/sagemaker/latest/dg/serverless-endpoints.html.

[21] M. Yu, T. Cao, W. Wang, and R. Chen, "Following the data, not the function: Rethinking function orchestration in serverless computing," in *Proc. of USENIX NSDI*, 2023, pp. 1489–1504.

[22] S. S. Shubha and H. Shen, "AdaInf: Data Drift Adaptive Scheduling for Accurate and SLO-guaranteed Multiple-Model Inference Serving at Edge Servers," in *Proc. of ACM SIGCOMM*, New York, NY, USA, Sep. 2023, pp. 473–485.

[23] M. Shahrad, R. Fonseca, Í. nigo Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," in *Proc. of USENIX ATC*, 2020, pp. 205–218.

[24] J. Li, L. Zhao, Y. Yang, K. Zhan, and K. Li, "Tetris: Memory-efficient serverless inference through tensor sharing," in *Proc. of USENIX ATC*, 2022.

[25] C. Zhang, M. Yu, W. Wang, and F. Yan, "MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving." in *Proc. of USENIX ATC*, 2019, pp. 1049–1062.

[26] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Optimizing inference serving on serverless platforms," *Proc. of ACM VLDB*, vol. 15, no. 10, 2022.

[27] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," in *Proc. of IEEE ICDCS*, 2021, pp. 138–148.

[28] "AWS Lambda," https://aws.amazon.com/lambda.

[29] "Azure Function," https://azure.microsoft.com/en-us/products/functions/.

[30] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. of ACM SenSys*, 2017, pp. 1–14.

[31] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. of ACM ASPLOS*, vol. 45, 2017, pp. 615–629.

[32] L. Capogrosso, E. Fraccaroli, S. Chakraborty, F. Fummi, and M. Cristani, "Mtl-split: Multi-task learning for edge devices using split computing," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*, 2024.

On Design Space Exploration of Cache System in Multi-Chiplet Systems

Yan Zhang[1,2], Xiaohang Wang[1,2,*], Yingtao Jiang[3], Amit Kumar Singh[4]
[1]State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China
[2]Hangzhou High-Tech Zone (Binjiang), Institute of Blockchain and Data Security
[3]University of Nevada, Las Vegas, USA [4]University of Essex, Colchester, UK

***Abstract*— While multi-chiplet based many-core systems have emerged as a viable solution for heterogeneous integration and addressing manufacturing and technological challenges in the post-Moore's Law era, their design and optimization remain highly complex and challenging. Among the various subsystems, the cache hierarchy has significant implications for overall system performance, yet its vast design space presents substantial optimization challenges. This complexity arises from factors such as the large number of chiplets in the system, the number of cores per chiplet, memory hierarchy variations, cache size variability, caching strategies, and inter-chiplet interconnection networks. Existing design space exploration methods, such as NN-Baton and IntLP, fail to optimize cache subsystem performance or thoroughly explore the design space. To address these limitations, we propose a novel design space exploration method for cache subsystem optimization. Our approach models cache miss rates and network latency as functions of cache hierarchy and inter-/intra-chiplet interconnection network parameters. We then define an optimization problem to minimize the concurrent average memory access time (C-AMAT) under cost and power consumption constraints. This problem is addressed using a bilevel optimization algorithm, which iteratively solves two independent subproblems: (1) cache subsystem optimization, and (2) inter-chiplet interconnection network optimization. Experimental results show that our method reduces the application execution time by 39.7% and 39.2%, on average, compared to architectures similar to AMD Zen 4 and Intel Sapphire Rapids, respectively, and by 25.91% over IntLP. These results underscore the potential of the proposed method for optimizing cache subsystems in future multi-chiplet based many-core systems.**

***Index Terms*—Many-core Systems, Multi-chiplet System, Design Space Exploration.**

I. Introduction

Multi-chiplet based many-core systems have been a new design paradigm to overcome the area wall to provide high computation power for various applications. In multi-chiplet systems, multiple chiplets or dielets can be integrated on interposers or substrates, and chiplets communicate by die-to-die (D2D) interfaces. A key to multi-chiplet based many-core systems is the cache hierarchy design. The difference in the cache hierarchy between a multi-chiplet based many-core system and conventional single chip (SoC) many-core systems is two folds: (1) The chiplets are "clusters" where the inter-chiplet communication is much more inefficient than that of the intra-chiplet communication. For example, the transmission latency between two adjacent chiplets is an order of magnitude longer than those of two adjacent tiles inside one chiplet. Due to the much higher latency and much lower bandwidth in inter-chiplet communication, the cache access traffic should be localized as much as possible to avoid high inter-chiplet communication latency [1]. (2) The problem of designing an optimal cache hierarchy has numerous parameters and extremely high complexity, especially in a multi-chiplet system.

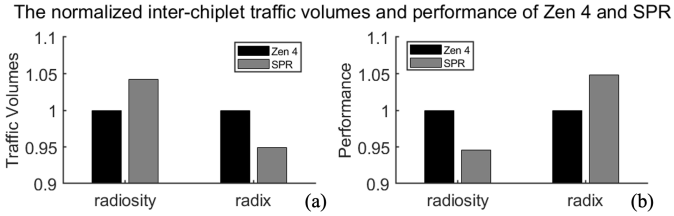


Fig. 1. Comparison of (a) inter-chiplet traffic volumes and (b) performance of AMD Zen 4 and Intel Sapphire Rapids.

Fig. 1 shows inter-chiplet traffic volumes for radiosity and radix applications on AMD Zen 4 [2] and Intel Sapphire Rapids (SPR) [3] systems. Zen 4 and SPR have different cache settings, e.g., whether the LLC in each chiplet is shared or not, and whether there is a centralized IO die for the memory controller or not. According to Fig. 1, for radiosity and radix, the inter-chiplet traffic volume of SPR is 1.042x and 0.949x of Zen 4 respectively, and correspondingly the performance of SPR is 0.946x and 1.048x of Zen 4 respectively. From this example, two observations can be made: (1) Higher inter-chiplet traffic volume leads to lower performance. (2) The configuration of cache and memory subsystem impacts the inter-chiplet traffic volume and thus performance. Therefore, it needs to explore the cache subsystem design space.

However, the design space of cache hierarchy is extremely huge, leading to an optimization problem with high complexity. For example, the design variables include chiplet number, the number of cores in each chiplet, memory hierarchy in each chiplet, cache size, different inter-chiplet interconnection networks, etc. Existing design space exploration methods like IntLP [4], NN-Baton [5], and Monad [6] cannot solve the above challenge since they cannot explore such a huge space.

Therefore, in this paper, we propose an efficient optimization algorithm for design space exploration of cache subsystems in multi-chiplet systems. In essence, (1) An analytical cache performance model based on C-AMAT [7] is proposed for multi-chiplet systems. (2) An optimization problem is formulated to select the parameters to optimize cache performance under the cost and power budgets. (3) A bilevel algorithm to efficiently solve the problem is proposed, where the optimization problem is decomposed into two subproblems: (a) cache subsystem optimization, and (b) inter-chiplet interconnection network optimization, and those two subproblems are optimized iteratively.

* Xiaohang Wang is the corresponding author.

This work was supported in part by the National Natural Science Foundation of China under Grants 92373205 and 62374146, in part by the National Key Research and Development Program of China No. 2023YFB4404404, in part by the key R&D programme of Zhejiang Province No. 2024C01012, in part by Ant Group Research Fund, in part by the Key Technologies R&D Program of Jiangsu (Prospective and Key Technologies for Industry) under Grant BE2023005-2, in part by CIE-Smartchip Research Fund No. 2023-004, and in part by the Fundamental Research Funds for the Central Universities (Zhejiang University NGICS Platform).

We would like to thank Xiaodong Li, Xin Tang, and Jie Song from Ant Group for their invaluable help.

TABLE I
NOMENCLATURE

Symbol	Definition
v_{ijk}	The i-th cache in level j in chiplet k.
$e_{ijk,mno}$	The link connecting v_{ijk} and v_{mno}.
$w(v_{ijk})$	The weights of node v_{ijk}.
$n_{p,q}$	The p-th node in chiplet q.
$c(n_{p,q}, n_{r,s})$	The link between adjacent nodes $n_{p,q}$ and $n_{r,s}$.
$\omega(c(n_{p,q}, n_{r,s}))$	The transmission latency between $n_{p,q}$ and $n_{r,s}$.
$\|R_i\|$	The number of cores in chiplet i.
n_α	The chiplet number.
h_i	The total number of memory hierarchy in chiplet i.

The remainder of the paper is organized as follows. Section 2 reviews the related work. Section 3 analyzes and models multi-chiplet systems, and defines the cache performance optimization problem. Section 4 presents a bilevel optimization algorithm to solve the optimization problem. Section 5 provides experimental results. Finally, Section 6 concludes the paper.

II. Related Work

Concurrent average memory access time (C-AMAT) [7] refines the traditional average memory access time (AMAT) model by accounting for memory concurrency and categorizes memory cycles to better model memory access performance. Layered performance matching (LPM) enables hierarchical memory performance analysis, transforming overall optimization into a set of local optimizations for each cache level. In [8], researchers developed models to enhance cache miss rate estimation and cache partitioning for multi-core systems, addressing data-sharing, coherence, and runtime monitoring for efficient cache use. In [5], [6], [9], several methodologies, including NN-Baton, SASMM, and Monad, were proposed to optimize chiplet-based system design, focusing on functional partitioning, resource allocation, and performance optimization. IntLP [4] proposed an integer linear programming model for the design of multi-chiplet systems, which determines the optimal partitioning of functional components and chiplet selection. Existing methods struggle to optimize multiple cache parameters or explore sufficiently large design spaces, and most focus on narrow objectives. This paper proposes a comprehensive optimization to overcome these challenges.

III. Performance Modeling and Problem Definition

A. Systems Modeling

As shown in Fig. 2, the multi-chiplet based many-core system is modeled by two graphs: memory hierarchy graph and inter-chiplet interconnection network topology graph, characterizing its cache and interconnection subsystems.

Definition 1. Memory hierarchy graph $G_M(V, E)$. $G_M(V, E)$ is an undirected graph, where each node $v_{ijk} \in V$ is either a core, a cache unit, or a DRAM unit. v_{ijk} is the i-th cache in level j in chiplet k. The edge $e_{ijk,mno} \in E$ indicates the link connecting v_{ijk} and v_{mno} (between two levels of cache units, or the reachability of core). If v_{ijk} is connected to v_{mno}, then $e_{ijk,mno} = 1$, indicating v_{ijk} is related to v_{mno} (for example, inclusiveness), and 0 otherwise. As shown in Fig. 2 (a), each core with an L1 cache, an L2 cache, and the two cores sharing an L3 cache are shown as the memory hierarchy graph. There is an edge between v_{131} and v_{121}, indicating the inclusiveness of v_{131} over v_{121}. Otherwise, there is no connected edge between v_{131} and v_{321}, indicating that v_{131} is not inclusive over v_{321}.

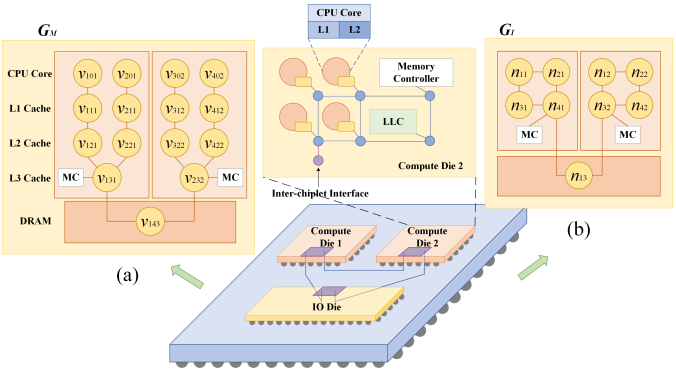


Fig. 2. The multi-chiplet based many-core system graph, (a) the $G_M(V, E)$ graph example and (b) the $G_I(N, C)$ graph example.

Define h_k as the total number of cache hierarchy in chiplet k. The weights $w(v_{ijk})$ of node $v_{ijk} \in V$ are as follows. The node v_{ijk} with $j = 0$ is a core, where $cpi(v_{ijk})$ is the ideal CPI (assuming no cache miss), which is the average number of clock cycles per instruction. The node v_{ijk} with $0 < j < h_k$ is a cache unit, where $cs(v_{ijk})$ is cache size, $cl(v_{ijk})$ is cache line size, $as(v_{ijk})$ is associativity, $rs(v_{ijk})$ is replacement strategy, and $ac(v_{ijk})$ is the cache access rate. The node v_{ijk} with $j = h_k$ is a DRAM unit, where $bw(v_{ijk})$ is DRAM bandwidth.

$$w(v_{ijk}) = \begin{cases} \langle cpi(v_{ijk}) \rangle, & j = 0 \\ \langle cs(v_{ijk}), cl(v_{ijk}), as(v_{ijk}), \\ rs(v_{ijk}), ac(v_{ijk}) \rangle, & 0 < j < h_k \\ \langle bw(v_{ijk}) \rangle, & j = h_k \end{cases} \tag{1}$$

Definition 2. Inter-chiplet interconnection network topology graph $G_I(N, C)$. $G_I(N, C)$ is also defined as an undirected graph where each node $n_{p,q} \in N$ is a node in chiplet q on the inter-chiplet network, which is composed of a core, an L1/L2 bank, a network interface, and a router, or is a D2D interface only. C is the set of network links, $c(n_{p,q}, n_{r,s}) \in C$ is the link between adjacent nodes $n_{p,q}$ and $n_{r,s}$. The chiplets in G_I are denoted as $R_1, \ldots, R_{n_\alpha}$, where each $R_i \subset N$, $i = 1, \ldots, n_\alpha$. n_α is the total number of chiplets. The number of cores in chiplet i is denoted by $|R_i|$.

Define $\omega(c(n_{p,q}, n_{r,s}))$ as the transmission latency between $n_{p,q}$ and $n_{r,s}$. If $q = s$, nodes $n_{p,q}$ and $n_{r,s}$ are in the same chiplet; otherwise, they are in different chiplets. Fig. 2 (b) shows an example of a three-chiplet system connected by an inter-chiplet network and its corresponding interconnection graph.

B. Miss Rate Modeling

In this paper, we model the miss rate with G_M where the parameters relate to cache miss rate of different cache levels.

Given G_M, the miss rate of node v_{ijk} $(0 < j \leq h_k - 1)$ is $M(v_{ijk})$. Define $\Pi(v_{ijk}) = \{\pi_1(v_{ijk}), \ldots, \pi_n(v_{ijk})\}$ as the set of nodes on entire reachable path from any core to v_{ijk} (excluding the cores), where $\pi_x(v_{ijk})$ is the x-th node on the reachable path and n is the total number of nodes. All upper-level nodes in cache hierarchy with connection relationships in v_{ijk} are included in $\Pi(v_{ijk})$.

According to [10], the cache miss rate $M(v_{ijk})$ of v_{ijk} is modeled as:

$$\Lambda(v_{ijk}) = \sum_{x=1}^{n} a_x \cdot cs\big(\pi_x(v_{ijk})\big)^{\alpha_x} + \sum_{x=1}^{n} b_x \cdot cl\big(\pi_x(v_{ijk})\big)^{\beta_x} + \sum_{x=1}^{n} c_x \cdot as\big(\pi_x(v_{ijk})\big)^{\gamma_x} \quad (2)$$

$$M(v_{ijk}) = \sum_{x=1}^{n} d_x \cdot \mathcal{S}\big(\Lambda(v_{ijk}), rs\big(\pi_x(v_{ijk})\big)\big)^{\delta_x} \quad (3)$$

$$\forall\, 0 < i \le |R_k|,\ 0 < j < h_k,\ 0 < k < n_\alpha,\ \pi_x(v_{ijk}) \in \Pi(v_{ijk})$$

where $\Lambda(v_{ijk})$ is a polynomial function of cache size, cache line size, and associativity. a_x, b_x, c_x, and d_x are miss rate model cache correlation coefficients, α_x, β_x, γ_x, and δ_x are polynomial orders, and $\mathcal{S}\big(\Lambda(v_{ijk}), rs\big(\pi_x(v_{ijk})\big)\big)$ is a mapping function that returns the miss rate, given $\Lambda(v_{ijk})$ and replacement strategy.

C. Delay Modeling

The delay model is used to estimate the transmission latency between two cores, which consists of two parts: (1) zero-load latency, which is related to the distance between the source and destination cores, (2) queuing delay, which is modeled by a queuing theory model.

1) Zero-load latency: By modeling the latency using G_I, $L(n_{p,q}, n_{r,s})$ is defined as the zero-load latency for node $n_{p,q}$ to access node $n_{r,s}$. The latency consists of two parts: the header packet latency $L_h(n_{p,q}, n_{r,s})$ and the serialization latency $L_s(n_{p,q}, n_{r,s})$ as follows.

$$L(n_{p,q}, n_{r,s}) = L_h(n_{p,q}, n_{r,s}) + L_s(n_{p,q}, n_{r,s}) \quad (4)$$

where the header packet latency $L_h(n_{p,q}, n_{r,s})$ is determined by the shortest path $l(n_{p,q}, n_{r,s})$ between node $n_{p,q}$ and node $n_{r,s}$ and the latency on the path. The serialization latency $L_s(n_{p,q}, n_{r,s})$ is related to the number of packet size (flit size). The formula for $L_s(n_{p,q}, n_{r,s})$ and $L_h(n_{p,q}, n_{r,s})$ is as follows.

$$L_s(n_{p,q}, n_{r,s}) = \mu \cdot \frac{s_i}{f} \quad (5)$$

$$L_h(n_{p,q}, n_{r,s}) = \eta \cdot l(n_{p,q}, n_{r,s}) \quad (6)$$

where μ is the transmission rate, s_i is the data packet size, f is the flit size, and η is the number of router pipeline stages.

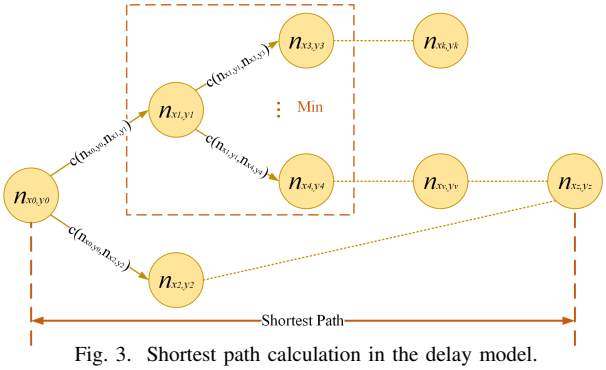


Fig. 3. Shortest path calculation in the delay model.

As shown in Fig. 3, the transmission latency of the shortest path is recursively derived using the Bellman equation [11], where the shortest path from node $n_{x_0,y_0} = n_{p,q}$ to node $n_{x_k,y_k} = n_{r,s}$ is referred to a sequence of nodes $\Gamma(n_{p,q}, n_{r,s}) = \Gamma(n_{x_0,y_0}, n_{x_k,y_k}) = \{n_{x_0,y_0} = n_{p,q}, n_{x_1,y_1}, \ldots, n_{x_z,y_z}, \ldots, n_{x_k,y_k} = n_{r,s}\}$. By comparing all possible paths between nodes, the shortest distance is ensured to be selected at each step of the recursion, as follows.

$$l^{(z)}(n_{x_0,y_0}, n_{x_z,y_z}) = \min\big\{l^{(z-1)}(n_{x_0,y_0}, n_{x_{z-1},y_{z-1}}) + c(n_{x_{z-1},y_{z-1}}, n_{x_z,y_z})\big\} \quad (7)$$

$$\forall n_{x_z,y_z} \in \Gamma(n_{x_0,y_0}, n_{x_k,y_k})$$

where $l^{(z)}(n_{x_0,y_0}, n_{x_z,y_z})$ is the shortest distance from n_{x_0,y_0} to n_{x_z,y_z} in the z-th step of recursion. $l^{(z)}(n_{x_0,y_0}, n_{x_z,y_z})$ is recursively updated until the optimal state is achieved.

Therefore, the calculation of $l(n_{p,q}, n_{r,s})$ follows the Bellman equation as follows.

$$l(n_{p,q}, n_{r,s}) = l(n_{x_0,y_0}, n_{x_k,y_k}) = \sum_{z=1}^{k} \min\big\{\omega\big(c(n_{x_{z-1},y_{z-1}}, n_{x_z,y_z})\big)\big\} \quad (8)$$

$$\forall n_{x_z,y_z} \in \Gamma(n_{x_0,y_0}, n_{x_k,y_k})$$

where $l(n_{x_z,y_z}, n_{x_z,y_z}) = 0$, and $\omega\big(c(n_{x_{z-1},y_{z-1}}, n_{x_z,y_z})\big)$ is the transmission latency between adjacent nodes $n_{x_{z-1},y_{z-1}}$ and n_{x_z,y_z}.

2) Queuing delay: The queuing delay of two nodes $n_{p,q}$ and $n_{r,s}$ is the summation of the queuing delay of each router n_{x_z,y_z} in the path $\Gamma(n_{p,q}, n_{r,s})$, obtained by the routing algorithm. In the G/G/1 model, a single-server queue with general arrival and service distributions, the flows $f_{a_c,b_c} = \{f_{a_1,b_1}, \ldots, f_{a_g,b_g}\}$ traversing n_{x_z,y_z} are analyzed. A router n_{x_z,y_z} consists of multiple input channels $IC_i^{n_{x_z,y_z}} = \{IC_1^{n_{x_z,y_z}}, \ldots, IC_q^{n_{x_z,y_z}}\}$ competing for a single output channel $OC_j^{n_{x_z,y_z}}$. The average queuing delay $T_{i\to j}(n_{x_z,y_z})$ of router n_{x_z,y_z} for a packet from $IC_i^{n_{x_z,y_z}}$ to $OC_j^{n_{x_z,y_z}}$ is as follows [12].

$$T_{i\to j}(n_{x_z,y_z}) = \begin{cases} \dfrac{\rho_j(n_{x_z,y_z})\left(C_A^2(n_{x_z,y_z}) + C_{S_j}^2(n_{x_z,y_z})\right)}{2\left(\mu_j(n_{x_z,y_z}) - \lambda_{i\to j}(n_{x_z,y_z})\right)}, & i = 1 \\ \dfrac{\lambda_j(n_{x_z,y_z})\left(C_A^2(n_{x_z,y_z}) + C_{S_j}^2(n_{x_z,y_z})\right)}{2\left(\mu_j(n_{x_z,y_z}) - \sum_{l=1}^{i-1}\lambda_{l\to j}(n_{x_z,y_z})\right)^2}, & 2 \le i \le q \end{cases} \quad (9)$$

where $\rho_j(n_{x_z,y_z})$ is the fraction of time that the $OC_j^{n_{x_z,y_z}}$ is occupied by packets from $IC_i^{n_{x_z,y_z}}$ in router n_{x_z,y_z}, $\lambda_j(n_{x_z,y_z})$ is the average packet rate to $OC_j^{n_{x_z,y_z}}$, $\lambda_{i\to j}(n_{x_z,y_z})$ is the average packet rate from $IC_i^{x_z,y_z}$ to $OC_j^{x_z,y_z}$, $\mu_j(n_{x_z,y_z})$ is the average service rate of the $OC_j^{n_{x_z,y_z}}$. $C_A^2(n_{x_z,y_z})$ and $C_{S_j}^2(n_{x_z,y_z})$ are the correlation coefficients between the corresponding packet arrival rate and service rate obtained based on the Allen-Cunneen approximation formula [12].

The total queuing delay $W(n_{p,q}, n_{r,s})$ is the sum of queuing delays of each router $n_{x_z,y_z} \in \Gamma(n_{p,q}, n_{r,s})$ from node $n_{p,q}$ to node $n_{r,s}$ for $OC_j^{n_{x_z,y_z}}$, calculated by the following formula.

$$W(n_{p,q}, n_{r,s}) = \sum_{z=0}^{k}\sum_{i=1}^{q} T_{i\to j}(n_{x_z,y_z}) \quad (10)$$

$$\forall n_{x_z,y_z} \in \Gamma(n_{x_0,y_0}, n_{x_k,y_k})$$

In summary, $D(n_{p,q}, n_{r,s})$ is the sum of zero-load latency $L(n_{p,q}, n_{r,s})$ and queuing delay $W(n_{p,q}, n_{r,s})$, represented as follows.

$$D(n_{p,q}, n_{r,s}) = L(n_{p,q}, n_{r,s}) + W(n_{p,q}, n_{r,s}) \quad (11)$$

D. Memory Concurrency Modeling

According to [7], define average hit concurrency $C_H(v_{ijk})$ and ratio of pure miss cycles over miss cycles $\kappa(v_{ijk})$ from a memory-centric perspective. $C_H(v_{ijk})$ and $\kappa(v_{ijk})$ can be modeled by a regression as follows [7].

$$C_H(v_{ijk}) = \sum_{x=1}^{n} f_x \cdot mpki\big(\pi_x(v_{ijk})\big)^{\vartheta_x} + \sum_{x=1}^{n} l_x \cdot ac\big(\pi_x(v_{ijk})\big)^{\iota_x} + g_x \cdot \big(\ln\big(bw(v_{ih_kk})\big)\big)^{\xi_x} \quad (12)$$

where f_x, l_x, and g_x are coefficients, ϑ_x, ι_x, and ξ_x are polynomial orders, $\pi_x(v_{ijk})$ is the x-th node on the reachable path from the core to v_{ijk}, $mpki\big(\pi_x(v_{ijk})\big)$ is the number of memory accesses per thousand instructions of $\pi_x(v_{ijk})$, $ac\big(\pi_x(v_{ijk})\big)$ is the cache access numbers of $\pi_x(v_{ijk})$, and $bw(v_{ihk})$ is DRAM bandwidth.

$$\kappa(v_{ijk}) = \sum_{x=1}^{n} m_x \cdot mpki\big(\pi_x(v_{ijk})\big)^{\varrho_x} + o_x \cdot \left(\ln\big(M(v_{i(h_k-1)k})\big)\right)^{\zeta_x} \quad (13)$$

where m_x and o_x are correlation coefficients, ϱ_x and ζ_x are polynomial orders, and $M(v_{i(h_k-1)k})$ is the miss rate of LLC, which is defined in Eq. (3) in section 3.B.

E. Performance Modeling

Given the memory hierarchy graph G_M and inter-chiplet interconnection network topology graph G_I of the multi-chiplet systems mentioned above, with the miss rate model, delay model, and memory concurrency model, the performance model is computed as follows.

The point-to-point latency between two nodes in the inter- and intra-chiplet interconnection system is calculated using the delay model, and the average delay between adjacent cache levels j and $j-1$ is as follows.

$$\overline{D}(j-1,j) = \frac{\sum_{c=1}^{g}\left(\sum_{k=1}^{n_\alpha}\sum_{i=1}^{|R_k|} D(v_{i(j-1)k}, v_{ijk})\cdot f_{a_c,b_c}\right)}{\sum_{c=1}^{g} f_{a_c,b_c}} + \frac{\sum_{c=1}^{g}\left(\sum_{k=1}^{n_\alpha}\sum_{i=1}^{|R_k|} D(v_{ijk}, v_{i(j-1)k})\cdot f_{a_c,b_c}\right)}{\sum_{c=1}^{g} f_{a_c,b_c}} \quad (14)$$

where f_{a_c,b_c} is the flow traversing $v_{i(j-1)k}$ to v_{ijk} and in reverse, $c = 1, \ldots, g$.

Define $\mathcal{C}(v_{i(j-1)k})$ as the average concurrent memory access time of cache layer $j-1$ within a CPU cycle. $\mathcal{C}(v_{i(j-1)k})$ is closely related to the average concurrent memory access time of the next level cache $\mathcal{C}(v_{ijk})$, and the recursive relationship is as follows [7].

$$\mathcal{C}(v_{i(j-1)k}) = \frac{H(v_{i(j-1)k})}{C_H(v_{i(j-1)k})} + M(v_{i(j-1)k})\cdot\kappa(v_{i(j-1)k})\cdot \big(\mathcal{C}(v_{ijk}) + \overline{D}(j-1,j)\big) \quad (15)$$

The C-AMAT $\sigma(G_M, G_I)$ is as follows [7].

$$\sigma(G_M,G_I) = \sum_{k=1}^{n_\alpha}\sum_{i=1}^{|R_k|}\left(\sum_{j=1}^{h_k-1}\prod_{t=1}^{j} M(v_{itk})\cdot\kappa(v_{itk})\cdot\left(\frac{H(v_{ijk})}{C_H(v_{ijk})} + \overline{D}(j-1,j)\right)\right) \quad (16)$$

where $H(v_{ijk})$ is the hit time of the j-th level cache of the chiplet k, $C_H(v_{ijk})$ is the average hit concurrency of the j-th level cache of the chiplet k, $M(v_{ijk})$ is the miss rate of the j-th level cache of the chiplet k, $\kappa(v_{ijk})$ is the ratio of pure miss cycles over miss cycles, $\overline{D}(j-1,j)$ is the average latency between adjacent $(j-1)$-th level and j-th level caches, n_α is the chiplet number, h_k is the total number of memory hierarchy in chiplet k, $|R_k|$ is the number of cores in chiplet k, and t is the recursive levels in the product up to j.

F. Cost and Power Consumption Modeling

The architecture cost is the cumulative cost of each component (die and DRAM) and the chiplet integration cost c_{int}. The cost of a die $C_{die}(v_{ijk})$ is as follows.

$$C_{die}(v_{ijk}) = \frac{A_i^{die}}{Y^{die}}\cdot c_1 \quad (17)$$

$$Y^{die} = \left(1 + \frac{\mathcal{D}\cdot\sum_{i=1}^{|R_i|} A_i^{die}}{a}\right)^{-a} \quad (18)$$

where A_i^{die} is the die area, c_1 is the cost per silicon area, Y^{die} is the asymptotic yield determined by the die area A_i^{die}, defect density $\mathcal{D}$, and clustering factor a.

The cost of DRAM $C_{dram}(v_{ijk})$ is expressed as:

$$C_{dram}(v_{ijk}) = \frac{bw(v_{ijk})}{U_{bw}}\cdot c_2 \quad (19)$$

where $bw(v_{ijk})$ is DRAM bandwidth, c_2 is the monetary cost of DRAM, and U_{bw} is the bandwidth provided by each unit.

The cost modeling of the multi-chiplet systems is as follows.

$$C(G_M,G_I) = \sum_{k=1}^{|R_k|} C_{die}(v_{ijk}) + C_{dram}(v_{ijk}) + c_{int} \quad (20)$$

The system power consumption is the cumulative power of each component. The power modeling of the multi-chiplet system is as follows.

$$P(G_M,G_I) = \sum_{i=1}^{n_\alpha} P_{core}(v_{ijk}) + P_{dram}(v_{ijk}) + \sum_{k=1}^{|R_k^{|}|} P_{router}(v_{ijk})\cdot r_k \quad (21)$$

where $P_{core}(v_{ijk})$ is the core power consumption, $P_{dram}(v_{ijk})$ is the DRAM power consumption, $P_{router}(v_{ijk})$ is the router power consumption, and r_k is the number of routers on each chiplet or interposer.

G. Problem Definition

With the performance, power, and cost model, under the constraints of cost and power, the optimization problem is defined to achieve the best performance of the multi-chiplet systems to minimize the C-AMAT $\sigma(G_M, G_I)$ with decision variables being G_M and G_I. The cost and power consumption of each chiplet should not exceed the respective threshold C_T and P_T. The optimization problem can be defined as follows.

$$\min\{\sigma(G_M,G_I)\} \quad (22)$$
$$\text{s.t.} \quad C(G_M,G_I) \le C_T, \quad P(G_M,G_I) \le P_T$$

The problem defined above has a huge search space. To efficiently solve this problem, a bilevel optimization algorithm is proposed in the next section.

IV. The Proposed Design Space Exploration Algorithm

A. Bilevel Optimization Algorithm Overview

The bilevel optimization algorithm works as follows. The problem defined in Eqn. (2)-(21) is decomposed into two subproblems: (1) cache subsystem optimization subproblem

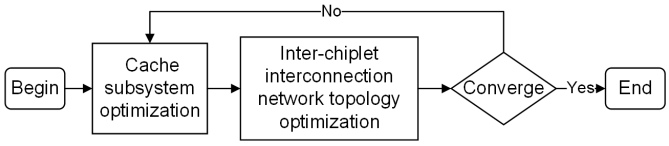


Fig. 4. Workflow of the bilevel optimization.

(P1), and (2) inter-chiplet interconnection network topology optimization subproblem (P2).

The workflow of bilevel optimization is shown in Fig. 4, which is performed iteratively. In each iteration, only one subproblem is solved, with the variables to the other subproblem fixed as constraints.

The optimization alternates between P1 and P2 until the target performance metric σ converges or a predefined maximum number of iterations $n_{\max}$ is reached. The convergence criteria are given by:

$$\sigma_i - \sigma_{i-1} \leq \delta \quad \text{or} \quad i \leq n_{\max} \tag{23}$$

where σ_i is the C-AMAT of the application in the i-th iteration, δ is the convergence condition, and $n_{\max}$ is the iteration limit.

B. Cache Subsystem Optimization (P1)

1) Search space definition: The objective of P1 is to select G_M parameters (cache level h_i, the numbers of cores $|R_i|$ in chiplet i, and parameters for each level of cache $w(v_{ijk})$), and it is defined as follows:

$$\min \left\{\sigma(G_M) = \sigma\big(h_i, |R_i|, w(v_{ijk})\big)\right\} \tag{24}$$
$$\text{s.t. } C(G_M) \leq C_T,\ P(G_M) \leq P_T$$

This subproblem can be solved by a branch-and-bound algorithm. In this algorithm, the decision variables in this subproblem are formed as a tree node g_b^a, which is the a-th node on the b-th layer and has a total of n search quantities $\hat{x_{cb}^a}$. A search tree is formed, where a tree node g_b^a is defined as follows.

$$g_b^a = \langle \hat{x_{1b}^a}, \hat{x_{2b}^a}, \ldots, \hat{x_{nb}^a} \rangle \tag{25}$$

Each node g_b^a is associated with an objective value $\sigma(g_b^a)$, and $C(g_b^a)$ and $P(g_b^a)$ are the cost and power consumption, respectively.

2) Branching rules: Each node g_b^a corresponding to $\hat{x_{cb}^a}$ assigned will branch a maximum of p_c branches, corresponding to $\hat{x_{c+1b}^a}$ assigned to be one of $\{y_1, y_2, \ldots, y_{p_c}\}$, which is the value range of $\hat{x_{c+1b}^a}$.

3) Cut rules: The cut rules are as follows.

- Pruned by optimality: When the currently searched node g_b^a is a leaf node and the feasible solution $\sigma(g_b^a)$ obtained by the current node is still greater than the current global minimum value, it is pruned.
- Pruned by infeasibility: When the currently searched node g_b^a does not meet the constraint conditions mentioned above, i.e., $C(g_b^a) \leq C_T$ or $P(g_b^a) \leq P_T$, where $C(g_b^a)$ and $P(g_b^a)$ return cost and power given the tree node g_b^a, it is pruned.
- Pruned by LPM mismatch: LPM [7] is an indicator of the matching of parameters of adjacent cache levels. If LPM rate is over a threshold of $x\%$, it indicates the match is low and leads to poor performance.

  Based on the performance model mentioned above, we use the LPM model [7] to verify the utilization of each cache layer. $LPMR(v_{ijk})$ is defined as the matching ratio at cache level j as follows.

Algorithm 1: Branch and Bound Algorithm

Input : The cache level h_i, the numbers of cores $|R_i|$ in chiplet i, and the parameters for each level of cache $w(v_{ijk})$.
Output: The cache access time of application σ^*.
Initialization:
- Initialize WQ as a working queue;
- Set g_b^a as the current node, and g_{b+1}^a as the newly branched node;
- Initialize $G^* = \emptyset$ as the best node;
- Set $\sigma^* = +\infty$ (optimal value found so far);
- Push root node into WQ;

while *WQ is not empty* **do**
  Pop the top node g_b^a out of WQ;
  if *g_b^a is not a leaf node* **then**
    Branch new tree nodes;
    for *each newly branched node g_{b+1}^a* **do**
      if *no cutting rules are met for g_{b+1}^a and WQ is not full* **then**
        Push g_{b+1}^a into WQ;
  else
    if $\sigma(g_b^a) \leq \sigma^*$ **then**
      Update $\sigma^* = \sigma(g_b^a)$;
      Set $G^* = g_b^a$;

$$LPMR(v_{ijk}) = \frac{f_{mem} \cdot \mathcal{C}(v_{i1k}) \cdot \prod_{j=1}^{h_k} \mu(v_{ijk})}{cpi(v_{ijk})} \tag{26}$$

where f_{mem} is the average number of memory accesses per instruction of the core, and $\mu(v_{ijk})$ is the ratio of miss cycles over memory active cycles, both of which can be obtained through the polynomial regression models.

Define a threshold of $x\%$ as the optimization target for the memory system, starting with matching at the L1 cache. Using the LPM model efficiently eliminates suboptimal branches. It ensures that only configurations meeting the performance threshold are explored, thus reducing search overhead.

- Pruned by Length of WQ: When the length of WQ is greater than W, it is pruned.

4) Workflow: Algorithm 1 outlines how the search process works. The algorithm works as follows.

- Initialization: Initialize σ^* to $+\infty$, which records the best objective function so far. Initialize g^* to an empty state ($\emptyset$), which will record the best node found so far in the search tree. The root node is enqueued into WQ.
- Search process: In each iteration of the search process, check if the work queue WQ is not empty. If WQ is not empty, g_b^a is dequeued from WQ. If g_b^a is not a leaf node, branch new tree nodes according to the branch rules. New nodes are enqueued into WQ if they don't meet the cut rules. If g_b^a is a leaf node, check if it is the best solution. If so, $\sigma^* = \sigma(g_b^a)$.
- Termination: If at any point the working queue WQ becomes empty, the search process terminates.

The time complexity of the BNB algorithm depends on the size of the search tree, determined by the branching factor p_c and the depth h_d. In the worst case, it explores $\mathrm{O}(p_c^{h_d})$ nodes, with each node involving $\mathrm{O}(1)$ operations for branching, pruning, and comparison. Pruning strategies (optimality, infeasibility, LPM, WQ length) significantly reduce the effective number of nodes, making the practical complexity

$O(\max\{|WQ|, p_c^{h_d}\})$.

C. Inter-chiplet Interconnection Network Topology Optimization (P2)

The objective of P2 is to select an inter-chiplet interconnection topology for G_I, and flit size f_i, and it is defined as follows.

$$\min\{\sigma(G_I)\} \quad \text{s.t. } C(G_I) \leq C_T,\ P(G_I) \leq P_T \tag{27}$$

To solve this problem, a few predefined topologies $t_i = \langle t_1, t_2, \ldots, t_j \rangle$, including Mesh, Ring, Torus, Crossbar, Clos, and Fat tree, and the flit size $f_i = \{32, 64, \ldots, 1024\}$, will be selected. An exhaustive search is performed to find the optimal topology and flit size.

V. Experimental Results

A. Experiment Setup

Experiments were performed on the sniper simulator with McPAT as the power simulator.

Table 2 lists the configurations of the multi-chiplet systems simulator. The benchmarks are from PARSEC, SPLASH-2, neural network, and database applications. The area and power model of routers are from DSENT, the area and power model of SRAM are from CACTI, and the area and power model of processor cores are from McPAT. The transmission energy consumption between adjacent chiplet is 1.17 PJ/bit [5], and the transmission latency between adjacent chiplets is composed of the following three parts: 1) packetization and depacketization overheads in the D2D interface, adopted from [13], 2) transceiver transmission latency, adopted from [14], and 3) interposer wire delay and power, adopted from [15].

As shown in Fig. 5, the selection of the $x\%$ threshold for the LPM model was obtained through experiments. Fig. 5 shows that $x = 65\%$ strikes a good balance of average application performance and algorithm search time, and thus is used in the experiments.

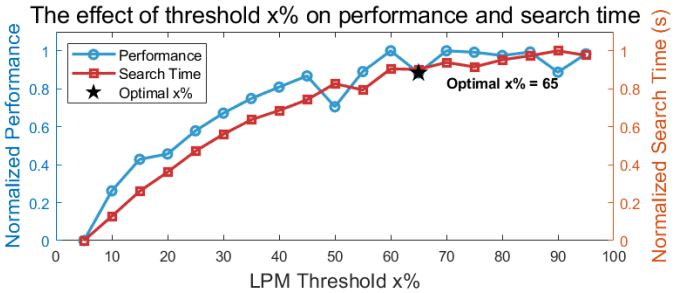


Fig. 5. The effect of a threshold of $x\%$ on the average application performance and the algorithm search time on the above benchmarks.

To evaluate the algorithm proposed in this paper, IntLP [4] is chosen for comparison. IntLP uses an integer linear programming model to determine the optimal partitioning of functional components and the selection of chiplets. In addition, Zen 4 and SPR are simulated in sniper to consider for comparison.

B. Performance Evaluation

In the subsequent experiments, the performance of the proposed method, Zen 4, SPR, and IntLP [4] were compared.

Fig. 6 (a) and (b) show the execution time comparison with power budgets of 200W and 160W respectively. From Fig. 6 (a), one can see that the execution time of the proposed method reduces 39.7%, 39.2%, and 25.91% over Zen 4, SPR, and IntLP, respectively. From Fig. 6 (b), one can see that at low power budget, the execution time of the proposed method reduces 40.61%, 39.49%, and 18.01% over Zen 4, SPR, and IntLP, respectively. The execution time of the four methods is normalized to that of Zen 4. The reason is that the proposed method can effectively explore the design space of cache subsystems in multi-chiplet systems, while Zen 4 and SPR are fixed designs, and IntLP can only optimize size and associativity for L1 and L2 caches, which fails to explore the design space well.

TABLE II
The Parallel Multi-Chiplet X86-64 Simulator Configuration for Design Space Exploration

Configuration	
Core architecture	x86-64
Main memory size	2GB
Get/Decode/Submit Size	4/4/4
Baseline frequency	3GHz
ROB size	64
L1 data cache (private)	16 KB, 2-way, 32B line, 2 cycles, 2 ports
L1 instruction cache (private)	32 KB, 2-way, 64B line, 2 cycles, 2 ports
Level 2 cache (shared)	64KB slices /core, 64 line, 6 cycles, 2 ports
Flit size	32, 64, 128, 256, 512, 1024 bits
Intra-chiplet communication latency	Router: 2 cycles; link: 1 cycle
Inter-chiplet communication latency	Router: 2 cycles
Buffer depth	4 flits
Benchmarks	
barnes, cholesky, fft, lu.cont, radiosity, radix, raytrace, streamcluster, swaptions, water.nsq, CNN, GCN, ResNet50, TPC_H	

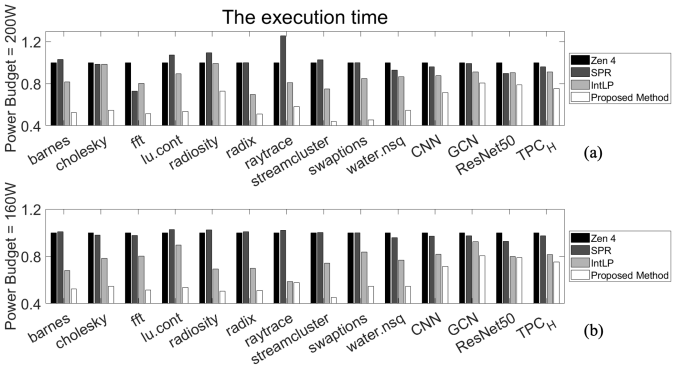


Fig. 6. The execution time of the different methods with (a) 200W and (b) 160W power budgets.

VI. Conclusion

In this paper, the problem of optimizing cache hierarchy in multi-chiplet based many-core systems was studied. Graph models were used to model the performance with respect to cache and inter-chiplet interconnection parameters. An optimization problem was formulated to minimize the cache access time of applications under cost and power constraints. A bilevel optimization algorithm was proposed to efficiently solve the above problem by iteratively solving two subproblems: (1) cache subsystem optimization and (2) inter-chiplet interconnection optimization. Experimental results confirmed the superiority of the proposed method over existing ones. Thus, it is suitable for cache hierarchy optimization for future large scale multi-chiplet systems.

REFERENCES

[1] J. Kim et al., "Architecture, Chip, and Package Co-design Flow for 2.5D IC Design Enabling Heterogeneous IP Reuse," in *DAC*, 2019, pp. 1–6.

[2] R. Bhargava and K. Troester, "AMD Next-Generation 'Zen 4' Core and 4th Gen AMD EPYC Server CPUs," *IEEE Micro*, vol. 44, no. 3, 2024, pp. 8–17.

[3] N. Nassif et al., "Sapphire Rapids: The Next-Generation Intel Xeon Scalable Processor," in *ISSCC*, 2022, pp. 44–46.

[4] S. Pal, D. Petrisko, R. Kumar, and P. Gupta, "Design Space Exploration for Chiplet-Assembly-Based Processors," *IEEE Trans. VLSI*, vol. 28, no. 4, 2020, pp. 1062–1073.

[5] Z. Tan, H. Cai, R. Dong, and K. Ma, "NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators," in *ISCA*, 2021, pp. 1013–1026.

[6] X. Hao, Z. Ding, J. Yin, Y. Wang, and Y. Liang, "Monad: Towards Cost-Effective Specialization for Chiplet-Based Spatial Accelerators," in *ICCAD*, 2023, pp. 1–9.

[7] J. Liu, P. Espina, and X.-H. Sun, "A Study on Modeling and Optimization of Memory Systems," *JCST*, vol. 36, no. 1, 2021, pp. 71–89.

[8] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. S. Jr, and J. Emer, "Adaptive Insertion Policies for High Performance Caching," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, 2007, pp. 381–391.

[9] A. Barai, G. Chennupati, N. Santhi, A.-H. Badawy, Y. Arafa, and S. Eidenbenz, "PPT-SASMM: Scalable Analytical Shared Memory Model: Predicting the Performance of Multicore Caches from a Single-Threaded Execution Trace," in *MEMSYS*, 2021, pp. 341–351.

[10] S. A. Przybylski, *Cache and memory hierarchy design: a performance directed approach*. Morgan Kaufmann, 1990.

[11] R. Al-Dujaily, N. Dahir, T. Mak, F. Xia, and A. Yakovlev, "Dynamic programming-based runtime thermal management (DPRTM): An online thermal control strategy for 3D-NoC systems," *ACM TODAES*, vol. 19, no. 1, 2013, pp. 1–27.

[12] A. E. Kiasari, Z. Lu, and A. Jantsch, "An Analytical Latency Model for Networks-on-Chip," *IEEE Trans. VLSI*, vol. 21, no. 1, 2013, pp. 113–123.

[13] A. Traber, F. Zaruba, and S. Stucki, "PULPino: A small single-core RISC-V SoC. In 3rd RISCV Workshop," in *RISCV Workshop*, 2016.

[14] D. Das Sharma, G. Pasdast, Z. Qian, and K. Aygun, "Universal Chiplet Interconnect Express (UCIe): An Open Industry Standard for Innovations With Chiplets at Package Level," *IEEE Trans. CPMT*, vol. 12, no. 9, 2022, pp. 1423–1431.

[15] M. A. Kabir, D. Petranovic, and Y. Peng, "Coupling extraction and optimization for heterogeneous 2.5D chiplet-package co-design," in *ICCAD*, 2020, pp. 1–8.

Harrow: Synthesis of Optical Logic Circuits via Harmonic Mean and Integer Partition

Jun-Wei Liang, Iris Hui-Ru Jiang, and Kai-Hsiang Chiu
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei 106319, Taiwan
email: {asfhiol, huiru.jiang, khchiu168}@gmail.com

***Abstract*—With the advancement of high-speed and energy-efficient optical interconnect and computation, photonic integrated circuits (PICs) have become a promising alternative to traditional CMOS circuits. A PIC can be synthesized by mapping the binary decision diagram (BDD) of target functions to optical switches and combiners. However, excessive signal attenuation along the light propagation may require extra optical-electrical signal conversion, thus introducing unwanted delays. In this paper, we aim to overcome this deficiency during logic synthesis: First, we optimize the signal efficiency factor by applying the concept of harmonic means to optimize DC combiners. Second, we eliminate redundant combiners by integer partition. Moreover, we properly arrange these proposed techniques in an optimal sequence of operations to form our main framework. Experimental results show that our framework outperforms the state of the art in terms of efficiency factor.**

***Index Terms*—Photonic integrated circuit, logic synthesis, efficiency factor, harmonic mean, integer partition**

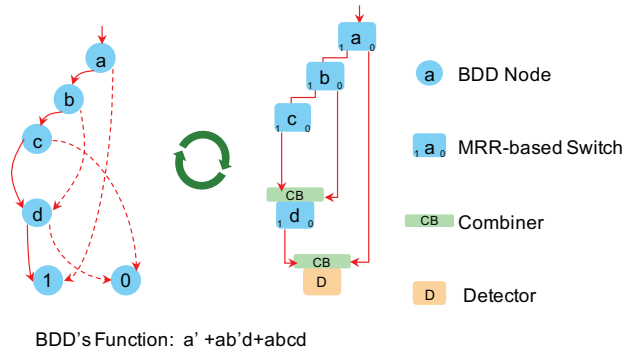


Fig. 1: Conversion between a binary decision diagram (BDD) and a synthesized photonic integrated circuit (PIC).

I. Introduction

Signal delay and energy consumption in integrated circuits have become the bottleneck of semiconductor technology advancement. Optical interconnect and computation, leveraging light-based transmission, is viewed as a promising alternative to conventional CMOS circuits, offering benefits in speed and energy efficiency. The automation of logic circuit design for photonic integrated circuits (PICs) has recently garnered considerable research attention [1]–[3]. The design flow of PICs can be divided into logic synthesis, optical component placement, and optical routing considering Wavelength Division Multiplexing (WDM) sharing. As the initial and foundational stage, logic synthesis critically impacts both placement and routing quality in the PIC design process. For instance, the power loss, timing delay, and hardware cost could be primarily determined in the logic synthesis stage [4]–[6].

Condrat *et al.* [4] pioneered the virtual gate concept in optical circuit synthesis, implementing Boolean functions through fundamental logical operations and Binary Decision Diagram (BDD) structures. Furthermore, to prevent significant light splitting loss incurred by [4], Wille *et al.* proposed a reverse BDD-based synthesis approach without using any splitters, which has become the backbone of optical circuit synthesis [5]. As depicted in Fig. 1, a target function is realized by implementing all of the paths from the root node of its BDD to the 1-terminal. Each BDD decision node is mapped to an optical switch controlled by its corresponding variable value, a node with multiple incoming signal paths requires a combiner, and the 1-terminal is replaced by a photo detector. The light source feeds to the top switch (the root node), traverses each switch (BDD node) and edge on the 1-paths, and finally arrives at the photo detector as the function output. Excessive signal attenuation during light propagation, however, can render the final output undetectable, necessitating solutions for effective signal management. Consequently, restoring the signal via additional optical-electrical (OE/EO) conversions becomes essential, involving either the insertion of OE/EO converters along BDD edges [7] or interconnecting partitioned subcircuits with these converters [8], albeit at the cost of added delays. (Meanwhile, BDD variable ordering can be optimized for enhancing the signal efficiency [9].) To further improve the synthesized circuit, Zhao *et al.* explored the use of combiners and directional couplers (DCs) as WDM, while Matsuo *et al.* leveraged WDMs to reduce hardware costs [10], [11].

In order to avoid the timing delays caused by additional OE/EO converters, the signal magnitude should be maximized, and thus, the efficiency improvement methodology was proposed. The efficiency improvement methodology is defined by the engineering approaches in a PIC to modify its optical components without changing functionality, including switches, combiners, and detectors. In Fig. 2(a), *Efficiency factor* is used to measure loss on a reversed BDD described in Section II-B. In Fig. 2(b), a directional coupler (DC) combiner can replace a Y-branch combiner to improve loss described in Section II-A. Based on this information, Zhao *et al.* proposed two algorithms in the efficiency improvement methodology [6], which were performed on node i in Fig. 2. In Fig. 2(c), the Combiner Elimination algorithm duplicates a sub-BDD multiple times to eliminate that sub-BDD's sourcing combiner. In Fig. 2(d), the Coupler Assignment algorithm takes advantage of DC usage on two selected combiners whose parameters are obtained from quadratically constrained programming. There is still a room for further efficiency improvement. Therefore, in this paper, we design two novel algorithms in the efficiency improvement methodology based on the theory of harmonic mean and integer partition. In Fig. 2(e), the Efficiency Adjustment algorithm can achieve optimal efficiency by harmonic mean on the parameters of a DC combiner. In Fig. 2(f), the Switch Duplication algorithm

This work was supported in part by the National Science and Technology Council, Taiwan.

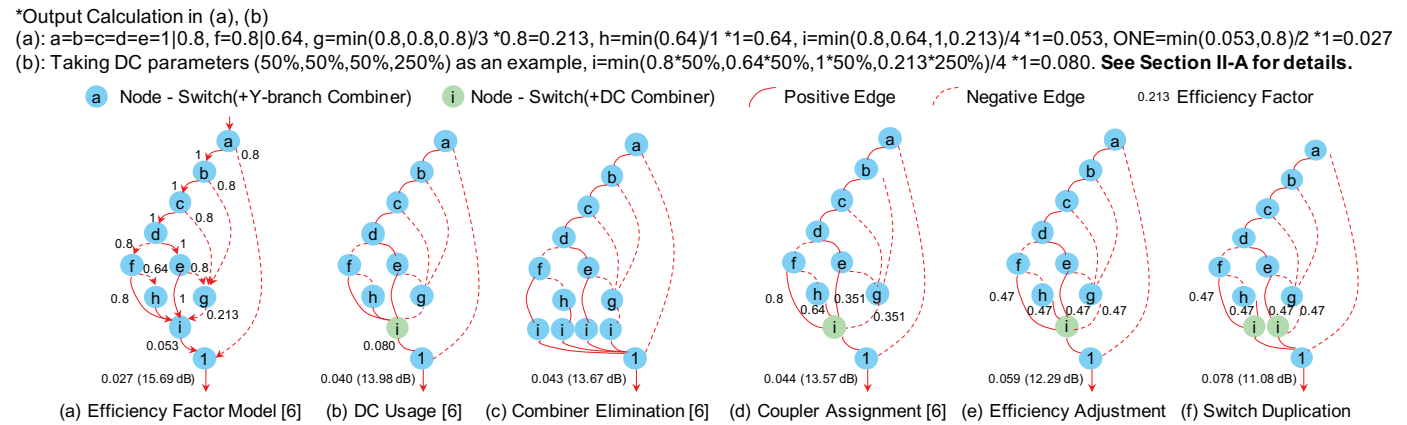


Fig. 2: Optical circuit synthesis. (a) Reverse BDD for the Boolean function $a' + abcd'fi + abcd'f'hi + abcdei + (ab' + abc' + abcde')g'i$. Proper switch output assignment results in an efficiency factor of 0.027. (b) A directional coupler combiner can replace the Y-branch combiner at node i to enhance BDD's efficiency (0.040). (c,d) [6] performs either Combiner Elimination or Coupler Assignment to improve efficiency. (e,f) Our algorithm achieves twofold improvement in efficiency.

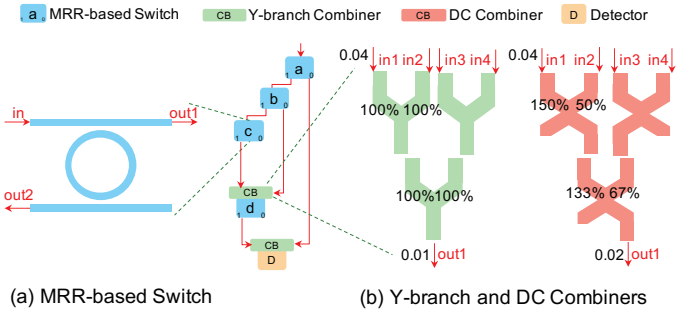


Fig. 3: Optical components of BDD synthesis. (a) An MRR-based switch with one input and two output directions. (b) A combiner is composed of multiple 2×1 combiners, either Y-branch combiners or directional coupler (DC) combiners.

traverses all switch duplication possibilities via integer partition and its factor permutations.

The remainder of this paper is organized as follows. Section II introduces the optical components, efficiency/loss model, and the addressed PIC logic synthesis problem formulation. Section III introduces the mathematical theories to design two algorithms and discusses their arrangement sequence. Section IV outlines our integrated framework. Section V demonstrates experimental results. Finally, Section VI concludes this work.

II. Preliminaries and Problem Formulation

This section introduces the optical components, details the efficiency factor computation, and states the problem formulation.

A. Optical Components

Fig. 3 illustrates the three key optical components used for implementing a BDD.

1) Detector: The detector, an essential element in every PIC, transforms the signal back into an electronic signal. It can detect light above a certain threshold. As signal falls below this threshold, an OE/EO converter will be added to restore power, which induces additional timing delay, hardware cost, and design complexity [7], [8]. Therefore, we should avoid OE/EO converters by improving the signal magnitude of optical circuits as much as possible.

2) Switch: Research in [6], [10] demonstrates that microring-resonator-based (MRR) optical switches are capable of synthesizing abstract BDD nodes. An optical terminator effectively resolves garbage output issues. Within the MRR-based switch architecture, resonance is utilized to direct light towards the second output, causing a certain transmission loss, as depicted in Fig. 3(a). By default, the first output (out1) is designated to the BDD node's positive edge, while the second output (out2) is for the BDD node's negative edge (else-edge). The signal magnitude ratio between out2 and out1 is 0.8, corresponding to a transmission loss of approximately $1dB$. Proper flipping the use of these two outputs may improve the signal efficiency, e.g., node g in Fig. 2. Other previous studies have incorporated different components. For instance, works by [4], [5] employed Mach-Zehnder interferometer (MZI), while [8] applied a DC for the BDD node.

3) Combiner: To aggregate incoming edges, combiners are designed as either Y-branch or directional coupler (DC) types. In Fig. 3(b), both N-input Y-branch combiners and DC combiners are in an array structure, and their efficiency degradation due to the mode mismatch ratio of 50% is calculated by $50\%^{\lg(N)} = N^{-1}$ [6]. Using a DC combiner allows adjustments to the efficiencies of incoming edges, optimizing overall performance. Consider a set of input efficiencies, $in_1, ..., in_N$, to an N-input DC combiner. Based on its functionality, we could adjust the input efficiencies by a scalar set, $k_1, ..., k_N$:

$$(in_1, ..., in_N) \rightarrow (k_1 in_1, ..., k_N in_N) \text{ w.r.t.} \sum_{j=1}^{N} k_j = N,$$

with an additional area of $N-1$ switches as hardware cost. For example, the scalar set of DC combiner at node i in Fig. 2(b) is $k_1, k_2, k_3, k_4 = 0.5, 0.5, 0.5, 2.5$; $0.5 + 0.5 + 0.5 + 2.5 = 4$. Interestingly, all of the aforementioned combiners can function as splitters via the reciprocal and the scatter matrix symmetry.

Based on the knowledge of these optical components above, we now introduce the efficiency factor model to estimate the loss in our work.

B. Efficiency Factor and Loss Model

Power loss in the synthesized PIC is estimated by calculating the efficiency factor of its BDD conversion, illustrated in Fig. 2(a) [6].

For each BDD, we compute the efficiency factor of each edge and node in topological order, denoted by γ: For edges,

the respective efficiency factor of a positive and negative edges is set to 1.0 and 0.8 due to the actual switch structure. For nodes, we consider the worst-case efficiency of their incoming edges. And, the efficiency $\gamma(in_1, ..., in_N)$ can be formulated as below:

$$\text{Y-branch Combiner: } \gamma = \frac{\min_{j=1}^{N}\{in_j\}}{N},$$

$$\text{DC Combiner: } \gamma = \frac{\min_{j=1}^{N}\{k_j in_j\}}{N} \text{ w.r.t. } \sum_{j=1}^{N} k_j = N.$$

For example, a Y-branch combiner is applied to node i in Fig. 2(a), while a DC combiner is used in Fig. 2(b).

Overall, the efficiency of a BDD is defined by the efficiency factor of its 1-terminal, γ_{Term}; the loss of a BDD is defined from its efficiency factor, $-10 * \log_{10}(\gamma_{\text{Term}})$. Also, the critical path within this BDD, from its source node to terminal one node exhibiting the worst-case (lowest) efficiency factor, is identified. Finally, a synthesized PIC's efficiency, loss, and critical path is the worst-case of efficiency, loss, and critical path to all BDDs.

C. Problem Formulation

To reduce power loss and elevate efficiency factors through logic synthesis, we address the following efficiency optimization problem:

The PIC Efficiency Optimization Problem: Given a circuit of Boolean functions and optical components, including switches, combiners, and detectors, the primary objective is to generate a logic circuit that maximizes the worst-case (lowest) efficiency factor, and the secondary objective is to minimize the number of additional OE/EO converters for power restoration.

III. Theoretical Foundations

In this section, we first introduce the harmonic mean and integer partition optimization to combiners, followed by an analysis of their optimization order and the design of our main algorithm.

A. Harmonic Mean Optimization

Sections II-A and II-B introduce an efficiency factor model for an N-input DC combiner, defined as $\gamma = \frac{\min_{j=1}^{N}\{k_j in_j\}}{N}$, subject to the condition $\sum_{j=1}^{N} k_j = N$. However, this model presents significant challenges. The objective is to determine the optimal scalar set, $k_1, ..., k_N$, which maximizes the efficiency factor of an N-input DC combiner, γ, called the **Efficiency Adjustment Problem**. Therefore, the harmonic mean is introduced as a solution method.

Theorem 1. *Let us first define the the harmonic mean of N-inputs, Harm$(in_1, ..., in_N) := \frac{N}{\frac{1}{in_1}+...+\frac{1}{in_N}}$. For an N-input DC combiner, the maximum value of the efficiency factor is equal to Harm$(in_1, ..., in_N) * \frac{1}{N}$.*

Proof. For a simpler representation, the numerator of the efficiency factor term is designated by f. Thus, $\gamma = \frac{f}{N}$ and $f(in_1, ..., in_N) = \min_{j=1}^{N}\{k_j in_j\}$ w.r.t. $\sum_{j=1}^{N} k_j = N$. Our objective is to prove that the harmonic mean represents the maximum value of f:

$$f \le k_j in_j \Leftrightarrow f * \frac{1}{in_j} \le k_j, \quad \forall j = 1, ..., N, \text{ w.r.t. } \sum_{j=1}^{N} k_j = N,$$

$$f * (\frac{1}{in_1} + ... + \frac{1}{in_N}) \le \sum_{j=1}^{N} k_j = N,$$

$$f \le \frac{N}{\frac{1}{in_1} + ... + \frac{1}{in_N}} = \text{Harm}(in_1, ..., in_N),$$

the last equality holds if and only if $k_j = \frac{\text{Harm}(in_1,...,in_N)}{in_j}, \forall j = 1, 2, ..., N$. Therefore, the maximum value of the efficiency factor is exactly Harm$(in_1, ..., in_N) * \frac{1}{N}$. □

Consequently, given an N-input DC combiner, we can calculate its maximum efficiency factor and corresponding scalar set in linear time complexity, called '**Efficiency Adjustment** by Harmonic Mean.' Now, we discuss another optimization issue.

B. Integer Partition Optimization

Expanding on prior research [6], the 'Combiner Elimination' algorithm eliminates an N-input combiner by applying an N-piece partitioning method, as shown in Fig. 2(c).

This action can be viewed as N-1 times duplication of a BDD switch node. An alternative approach is to duplicate the switch once, twice, thrice, etc. As a result, another problem arises. The objective is to enumerate all possible ways to duplicate switch nodes first and then arrange the incoming edges to the duplicated switches in order to optimize the efficiency factor, called the **Switch Duplication Problem**.

For the first objective, we propose a refined method using integer partition theory, which studies all possibilities for partitioning an integer number. For example, when $N = 4$ for node i in Fig. 2, the possible partitions include '4', '$3 + 1$', '$2 + 2$', '$2 + 1 + 1$', and '$1 + 1 + 1 + 1$'. Notably, the partition of '$1 + 1 + 1 + 1$' is exactly the special case corresponding to 'combiner elimination.' Typically, partition case enumeration can be executed using Depth-First Search (DFS), although this results in an exponentially large number of instances: $\frac{1}{4n\sqrt{3}} \exp(\pi\sqrt{\frac{2n}{3}})$ as $n \to \infty$, [12].

For the second objective, we introduce a greedy method for the optimal arrangement of input factors across all duplication possibilities. It can be formulated as follows:

The Factor Arrangement Subproblem: Suppose an N-input combiner, its input factors $in_1 \le ... \le in_N$, and an integer partition of N, $N = p_1 + ... + p_k$, where p_i is the number of factors required for the i-th switch. We aim to maximize the worst-case efficiency factor by arranging N-many incoming factors to k-many duplicated switches, that is:

$$\text{Through } \{in_1, ..., in_N\} \Leftrightarrow IN_{p_1} \sqcup IN_{p_2} \sqcup ... \sqcup IN_{p_k},$$

$$\text{To maximize } \min_{t=1}^{k} \frac{\min\{IN_{p_t}\}}{p_t},$$

where IN_{p_i} is the i-th partition of the set of input factors.

Theorem 2. *In the **Factor Arrangement Subproblem**, arranging the lowest input factor to the smallest partition is always optimal.*

Proof. Let us relabel $p_1 \le ... \le p_k$ for easier description. It is always true that the smallest partition must contain the lowest input factor in the optimal arrangement solution.

Suppose not, we may assign in_1 to some IN_{p_t} $(t > 1)$ to afford a larger partitioning efficiency factor. However, this assignment will break the lower bound to the worst-case efficiency factor:

$$\text{Suppose not: } \frac{\min\{in_1 \cup IN_{p_t}\}}{p_t} = \frac{in_1}{p_t},$$

$$\text{However: } \frac{\min\{IN_{p_1}\}}{p_1} = \frac{in_1}{p_1} \ge \frac{in_1}{p_t}, \frac{\min\{IN_{p_t}\}}{p_t} \ge \frac{in_1}{p_t}. \text{ ↯}$$

Leveraging the greedy property above, we can iteratively arrange the lowest input factor to the smallest partition to derive an optimal solution to the **Factor Arrangement Subproblem**. □

Combining insights from integer partitioning, DFS enumeration, and optimal assignment solutions, we theoretically propose

the 'Switch Duplication by Integer Partition' algorithm. The subsequent section shows the optimization order to implement the 'Efficiency Adjustment' and 'Switch Duplication' algorithms.

C. Optimization Order

The optimal sequence for performing operations warrants investigation. Potential approaches include: (A) Efficiency Adjustment only, (B) Switch Duplication only, (C) Efficiency Adjustment followed by Switch Duplication, and (D) Switch Duplication followed by Efficiency Adjustment. Again, we can answer this question through a theoretical examination.

Theorem 3. *To enhance the worst-case efficiency factor of an N-input combiner, the optimal operational sequence is ranked as follows: option (C) supersedes option (A), which in turn outperforms option (D), expressed as '(C) $\succeq$ (A) $\succeq$ (D).'*

Proof. First, (C) $\succeq$ (A) trivially holds since the possibility of not duplicating the switch (not partitioning integer) is also considered in the Switch Duplication algorithm. This means that option (A) is considered within option (C).

Secondly, we prove (A) $\succeq$ (D) now. Suppose an N-input combiner, its input factors $in_1 \leq ... \leq in_N$, and an integer partition of N, $N = p_1 + ... + p_N$ w.r.t. $p_1 \leq ... \leq p_N$, are given. Let us compare the numerators of the worst-case efficiency factors of (A) and (D), denoted by f and g respectively:
(D) - Duplicate Combiner first:

$$\{in_1, ..., in_N\} \xrightarrow{\text{Greedy Arrangement}} IN_{p_1} \sqcup IN_{p_2} \sqcup ... \sqcup IN_{p_k},$$
$$\Leftrightarrow \{in_1, in_2, ..., in_{p_1}\} \sqcup \{in_{p_1+1}, in_{p_1+2}, ..., in_{p_1+p_2}\} \sqcup ...$$
$$\Leftrightarrow IN_{p_t} = \{in_{p_1+...+p_{t-1}+1}, ..., in_{p_1+...+p_{t-1}+p_t}\}, \forall t = 1, ..., k.$$

(D) - Followed by Adjust Efficiency: $(\forall t = 1, ..., k)$
Efficiency factor of IN_{p_t}: $\text{Harm}(IN_{p_t})$,
Worst-case efficiency factor: $g \leq \text{Harm}(IN_{p_t})$,

$$\Leftrightarrow g \leq \frac{p_t}{\frac{1}{in_{p_1+...+p_{t-1}+1}} + ... + \frac{1}{in_{p_1+...+p_{t-1}+p_t}}},$$
$$\Leftrightarrow \frac{p_t}{g} \geq \frac{1}{in_{p_1+...+p_{t-1}+1}} + ... + \frac{1}{in_{p_1+...+p_{t-1}+p_t}}. \quad (*)$$

(A) - Efficiency Adjustment only:

Worst-case efficiency factor: $f = \frac{N}{\frac{1}{in_1} + ... + \frac{1}{in_N}}$,

$$\Rightarrow f = \frac{N}{\frac{1}{in_1} + ... + \frac{1}{in_N}} \overset{(*)}{\geq} \frac{N}{\frac{p_1}{g} + ... + \frac{p_k}{g}} = \frac{N}{\frac{p_1+...+p_k}{g}} = \frac{N}{\frac{N}{g}} = g.$$

Hence, we derive that the best sequence of operations is (C) $\succeq$ (A) $\succeq$ (D). □

Following this, the Switch Duplication algorithm could be simplified using the insights gained from above.

D. Simplified Switch Duplication

Adopting the strategy of 'Efficiency Adjustment followed by Switch Duplication,' it becomes possible to simplify the Switch Duplication algorithm using the information derived from the harmonic mean. For an adjusted N-input DC combiner, the **Factor Arrangement Subproblem** is no longer a difficulty since all input factors are $\text{Harm}(in_1, ..., in_N)$. Furthermore, it can be proved that only evenly dispersed partitions can be the optimal partition. For $N = 6$, feasible partitions include '6', '$3+3$', '$2+2+2$', '$2+2+1+1$', '$2+1+1+1+1$', and '$1+1+1+1+1+1$', excluding configurations such as '$5+1$' or '$3+2+1$'. This breakthrough reduces the partition instances from exponentially to linear and allows for replacing DFS enumeration with simple linear traversal.

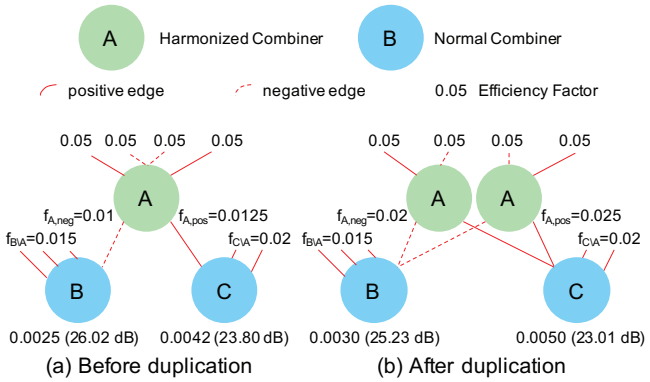


Fig. 4: In (a) is an adjusted 4-input DC combiner A, and its children switch nodes B and C. In (b), the Switch Duplication with respect to the $2+2$ partition is considered. Moreover, the hardware costs before and after duplication are the same.

Fig. 4 illustrates the operation of a Simplified Switch Duplication corresponding to a '$4 = 2+2$' partition. In Fig. 4(a), we observe that the input factors for the adjusted DC combiner A are uniform at 0.05, which acts as a lower bound on the efficiency of combiners B and C. In Fig. 4(b), through applying a '$2+2$' partition on the adjusted DC combiner, the efficiency at combiners B and C is seen to increase. Moreover, the hardware costs before and after duplication are $4+1+1$ and $2+2+1+1$ switches, which are maintained. This case study demonstrates the significant impact of partitioning on not just the adjusted DC combiner but also on children combiners.

To formulate this tradeoff, the efficiency factors of combiner A and its two derivatives, combiner B and C, are considered. The efficiency factor before partition is noted as f_A, f_B, and f_C, respectively. After duplication with respect to the partition of $N_A = \frac{N_A}{k} + ... + \frac{N_A}{k}$, we have:

$$f_A = \frac{\text{Harm}(in_1, ..., in_{N_A})}{N_A} \Rightarrow f'_A = \frac{\text{Harm}(in_1, ..., in_{N_A})}{\frac{N_A}{k}},$$
$$f_B = \frac{\min(f_A, f_{B-A})}{N_B} \quad \Rightarrow \quad f'_B = \frac{\min(f'_A, f_{B-A})}{N_B - 1 + k},$$
$$f_C = \frac{\min(f_A, f_{C-A})}{N_C} \quad \Rightarrow \quad f'_C = \frac{\min(f'_A, f_{C-A})}{N_C - 1 + k},$$

Objective function: $\log(\frac{f'_B}{f_B}) + \log(\frac{f'_C}{f_C})$,

where N_A, N_B, and N_C are the number of incoming edges of A, B, and C, respectively. To leverage the benefits of Switch Duplication effectively, we introduce normalization as a core concept to establish an objective function: $\log(\frac{f'_B}{f_B}) + \log(\frac{f'_C}{f_C})$. The aim is to select the partition that maximizes the objective function output in a Switch Duplication scenario. It is worth mentioning that in the base case, when $k = 1$, the function's output is precisely 1. The following work calls this algorithm 'Simplified Switch Duplication.'

IV. Our Synthesis Framework

In this section, we present our PIC logic synthesis framework, named Harrow.

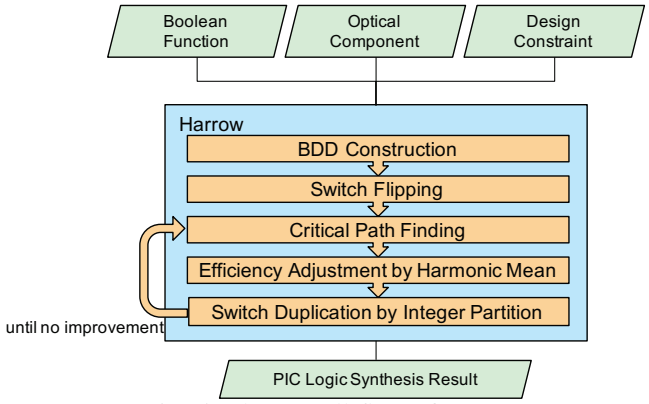


Fig. 5: The overall flow of Harrow.

Algorithm 1: Harrow

while *efficiency has not converged* **do**
  $(f, n_S, n_T) \leftarrow$ the BDD, its source, and terminal-one
  for *each node* n_i *in* f*, from* n_T *to* n_S **do**
    swap(n_i.left_child, n_i.right_child)
    $n_i \leftarrow n_i$.argmin$(in_1, \ldots, in_N)$
while *ans* $==$ *True* **do**
  $(f, n_S, n_T) \leftarrow$ the BDD, its source, and terminal-one
  ans = Combiner-Optimization(f, n_S, n_T)

Algorithm 2: Combiner-Optimization

$(f, n_S, n_T) \leftarrow$ the chosen BDD, its source, and terminal-one
for *each node* n_i *in* f*, from* n_T *to* n_S **do**
  if n_i*.inputs* < 2 **then**
    continue
  if n_i*.Adjusted and* n_i*.Duplicated* **then**
    continue
  if *not* n_i*.Adjusted* **then**
    Efficiency_Adjustment(n_i)
    n_i.Adjusted = True
  else if *not* n_i*.Duplicated* **then**
    $(A, B, C) \leftarrow (n_i, n_i$.left_child, n_i.right_child$)$
    for *each evenly dispersed partition of* n_i **do**
      Compute f'_A, f'_B, f'_C and Obj(A, B, C)
    Switch_Duplication(n_i)
    // Through the partition with largest Obj value
    n_i.Duplicated = True
  return *True*
  $n_i \leftarrow n_i$.argmin$(in_1, \ldots, in_N)$
return *False*

A. Overview

Fig. 5 depicts the overall flow of Harrow. First, we construct an abstract BDD from the given Boolean function. Subsequently, we properly flip the switch output assignment of positive and negative edges on the generated BDD to enhance loss. The framework proceeds with iterative identification of the critical path, wherein Efficiency Adjustment and Switch Duplication algorithms are applied to the respective BDD nodes, repeating the process until convergence is achieved with no further optimization.

TABLE I: Performance Scalability Comparison of net efficiency improvement (Imprv) against limited/actual switch ratio cost (Limit/#SW) for [5], [6], Harrow with Efficiency Adjustment only (Harrow-EA), and proposed Harrow (Harrow-All).

Limit	[5]		[6]		Harrow-EA		Harrow-All	
	#SW	Imprv	#SW	Imprv	#SW	Imprv	#SW	Imprv
1	1.000	1.000	1.000	8.965	1.000	7.853	1.000	7.853
1.05	1.000	1.000	1.015	14.702	1.025	9.070	1.024	9.092
1.1	1.000	1.000	1.046	18.457	1.057	11.209	1.057	11.206
1.15	1.000	1.000	1.069	18.646	1.091	16.913	1.091	16.652
1.2	1.000	1.000	1.089	20.976	1.123	29.855	1.126	24.897
1.25	1.000	1.000	1.111	18.678	1.158	42.992	1.158	33.330
1.3	1.000	1.000	1.125	20.372	1.187	79.171	1.191	76.921
1.35	1.000	1.000	1.138	18.746	1.208	99.493	1.215	92.678
1.4	1.000	1.000	1.153	19.171	1.235	121.999	1.242	133.731
1.45	1.000	1.000	1.172	20.935	1.258	143.358	1.263	147.941
1.5	1.000	1.000	1.199	20.894	1.280	164.992	1.288	170.084
1.55	1.000	1.000	1.216	21.940	1.295	167.468	1.302	171.198
1.6	1.000	1.000	1.230	20.519	1.316	169.129	1.328	174.732
1.65	1.000	1.000	1.242	21.252	1.339	174.116	1.354	178.601
1.7	1.000	1.000	1.261	22.667	1.361	176.312	1.378	182.183
2	1.000	1.000	1.345	24.132	1.391	187.084	1.411	192.875

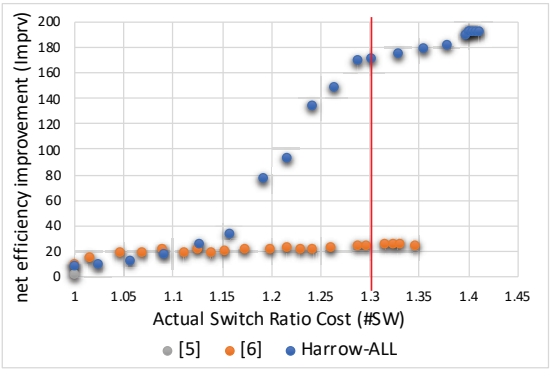


Fig. 6: Visual representation of Table I. Performance Scalability Comparison of net efficiency improvement (Imprv) against actual switch ratio cost (#SW) for [5], [6], and proposed Harrow (Harrow-All).

B. Switch Flipping

Let us consider a BDD generated from the given single-bit Boolean function. As mentioned in Section II-A2, we can improve the efficiency by flipping the positive and negative edge output of a BDD switch node, corresponding to a (1.0, 0.8) exchange of the efficiency factor. The Switch Flipping is to flip BDD nodes located along the critical path. Empirical evidence indicates that the BDD's efficiency typically stabilizes after approximately a dozen iterations of switch flipping.

C. Critical Path Finding

To identify the critical path, one observation is the critical path could be dynamically changed during the BDD efficiency optimization process. Therefore, it is necessary to identify the critical path for each time. Critical Path Finding involves traversing the BDD from the terminal node back to the source node, iteratively identifying and backtracking through parent nodes with the minimum incoming efficiency, denoted as $\arg\min(in_1, ..., in_N)$. To analyze the time complexity of this algorithm, the maximum height equates to the variable count for any BDD.

D. Combiner Optimization

The Combiner Optimization includes Efficiency Adjustment by Harmonic Mean and Switch Duplication by Integer Partition, as depicted in Algorithm 2. In lines 1–18, we may leverage

TABLE II: Comparison of net efficiency improvement (Imprv), loss (Loss) (dB), and CPU time (Time) (s) for [5], [6], Harrow-EA, and Harrow-All at actual switch ratio cost (#SW) of 1.3.

Benchmark	#PI	#PO	[5]				[6]				Harrow-EA				Harrow-All			
			#SW	Imprv	Loss	Time	#SW	Imprv	Loss	Time	#SW	Imprv	Loss	Time	#SW	Imprv	Loss	Time
cps	24	109	1654	1.00	32.35	0.10	2219	23.10	23.82	0.29	2079	**91.87**	**18.10**	0.10	2082	**91.87**	**18.10**	0.11
alu4	14	8	653	1.00	45.56	0.06	814	11.11	37.08	0.12	976	**49.70**	**32.50**	0.06	961	27.22	33.87	0.06
table3	14	14	1634	1.00	42.29	0.06	2036	11.51	32.68	0.08	2083	93.26	24.55	0.07	2085	**93.65**	**24.53**	0.07
apex5	117	88	629	1.00	27.96	0.11	724	2.33	25.46	0.14	868	4.07	22.62	0.11	834	**4.89**	**21.63**	0.11
table5	17	15	1757	1.00	48.19	0.06	2276	65.15	33.89	0.09	2315	513.25	25.85	0.07	2347	**545.03**	**25.71**	0.13
vda	17	39	1017	1.00	28.76	0.07	1347	3.77	23.89	0.09	1437	12.85	18.79	0.07	1450	**13.02**	**18.75**	0.08
k2	45	45	2009	1.00	41.98	0.09	2652	66.59	29.85	0.11	2655	933.57	22.26	0.09	2679	**958.94**	**22.15**	0.12
seq	41	35	2192	1.00	44.09	0.09	2818	46.31	31.94	0.15	2671	382.58	24.69	0.10	2718	**399.76**	**24.54**	0.13
count	35	16	195	1.00	28.27	0.06	312	**35.97**	**14.54**	0.06	220	13.72	17.55	0.06	220	13.72	17.55	0.11
dalu	75	16	1482	1.00	44.58	0.10	1817	8.62	37.63	0.12	1771	50.87	30.30	0.10	1859	**53.44**	**30.12**	0.11
i2c	147	142	947	1.00	30.46	0.25	1181	38.08	20.60	0.29	1150	**100.09**	**15.60**	0.26	1154	98.85	15.64	0.30
i8	133	81	433	1.00	32.49	0.12	564	5.16	26.74	0.14	581	50.98	19.08	0.13	599	**52.08**	**19.01**	0.13
simple_spi	148	144	763	1.00	27.13	0.26	961	8.42	21.56	0.29	939	**22.52**	**17.27**	0.26	942	22.10	17.31	0.29
frg2	143	139	708	1.00	37.27	0.16	933	7.10	30.70	0.19	948	**25.22**	**24.98**	0.16	947	22.22	25.30	0.18
Comparison	-	-	1.000	1.000	-	-	1.297	23.873	-	-	1.295	167.468	-	-	1.302	171.198	-	-

the Critical Path Finding to traverse the BDD in search of nodes to apply the Efficiency Adjustment or Switch Duplication algorithms. Lines 10–16 integrate the methodology described in Section III-D to implement the **Simplified Switch Duplication algorithm** effectively.

Now, we can describe our Harrow as listed in Algorithm 1. Lines 1–5 employ the Switch Flipping algorithm to enhance efficiency, while lines 6–8 iteratively apply the Combiner Optimization until the process indicates no further potential for improvement.

V. Experimental Results

The proposed algorithm and the methods from [5], [6] were implemented in C++ programming language and integrated with the ABC engine [13]. All experiments were performed on an Apple M1 3.2GHz MacBook Pro with 16GB RAM. Consistent with previous works [6], [7], [10], we used the benchmark [14] in our experiment. In the experiment, the execution commands involved: 'read benchmark,' 'collapse,' 'Extra_ReorderCudd,' and a custom command of experimental algorithms. Certain benchmarks could not be tested: 'stpmotor (steppermotordrive)', 'spla,' 'alu4,' and 'misex3c' cases.

Table I provides a comparative analysis of the scalability of our proposed algorithms against [5], [6], highlighting significant improvements in net efficiency across various switch ratio limits. Since the detection threshold value was set to 15dB, 20dB, 25dB, and 40dB in previous works [7], [8], we only measured these efficiency improvement algorithms on BDDs whose terminal loss exceeded 15dB. The first line is the upper bound of the switch ratio cost (Limit) to every measured BDD. For these four methods, columns 2, 4, 6, 8 are the exact switch ratio cost, and columns 3, 5, 7, 9 are the average of measured BDDs' net efficiency improvement on these 14 benchmarks. For example, if the switch ratio cost is limited to no more than 1.5 on every BDD of benchmarks, the actual switch ratio cost of these four methods overall is 1.000, 1.199, 1.280, 1.288. Fig. 6 visualizes the scalability of net efficiency improvements with increasing switch ratio costs, demonstrating the superior performance of our proposed approach. The switch ratio cost of 1.3 represents an optimal balance between performance improvement and switch count cost, serving as a sweet point for algorithm scalability.

Table II indicates the effectiveness of our designed algorithms, benchmarking our results against [5], [6] when the switch ratio cost is restricted to 1.3. The initial three columns list the benchmark information, including the number of primary inputs (#PI) and outputs (#PO). For these four methods, columns 4, 8, 12, 16 are the exact switch ratio cost (#SW) on each benchmark; columns 5, 9, 13, 17 are the average of net efficiency improvement (Imprv) on each benchmark; columns 6, 10, 14, 18 outline the loss, which is calculated by $-10 * \log_{10}(\gamma_{\text{Term}})$, see Section II-B; columns 7, 11, 15, 18 list the CPU times in seconds. All experimental algorithms were performed independently on the measured BDDs of benchmarks in the entire experiment. For [6], iterations of combiner elimination and coupler assignment were set to 5 and 37, respectively. For our designed methods, Harrow-EA and Harrow-All, the iterations of switch flipping were set to 50, and the maximum partition count (k) was 1 and 6, respectively.

First of all, we compared [6] and Harrow-All. Our Harrow-All method achieved a double-digit net efficiency improvement over [6], with a seven times improvement and better loss on 13 of 14 benchmarks. However, when the switch ratio cost was limited to 1.05 or 1.1, Harrow-All underperformed due to insufficient switch quota for the 'Efficiency Adjustment' algorithm on small BDDs. Secondly, we compared Harrow-EA and Harrow-All. While Harrow-EA and Harrow-All performed better on 5 and 9 benchmarks, respectively, with average improvements tied for both. Our explanation is that despite the theoretical guarantee of no extra switch cost in a 'Switch Duplication' operation, the duplicated switches contribute additional edges, ultimately increasing the switch count and diminishing Harrow-All's performance. In summary, all three methods demonstrated the potential to enhance net efficiency, and our Harrow-All method indicated a substantial advancement in optimizing PIC logic synthesis.

VI. Conclusions

To mitigate the signal attenuation issue in PICs, this paper presents three novel techniques for signal efficiency improvement, including harmonic mean, integer partition, and optimal operational sequence. The harmonic mean provides an optimal method for adjusting combiner efficiency, the integer partition offers a systematic approach to exploring all switch-duplication possibilities, and the optimal operational sequence ensures the efficient arrangement of these techniques. Together, these techniques form the basis for the Efficiency Adjustment and Simplified Switch Duplication algorithms, advancing the field of PIC logic synthesis. Experimental results demonstrate that Harrow achieves a double-digit improvement in efficiency compared to state-of-the-art methods, while maintaining comparable hardware overhead.

REFERENCES

[1] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avizienis, S. Lin *et al.*, "Single-chip microprocessor that communicates directly using light," *Nature*, vol. 528, no. 7583, pp. 534–538, 2015.

[2] A. Mirza, S. M. Avari, E. Taheri, S. Pasricha, and M. Nikdast, "Opportunities for cross-layer design in high-performance computing systems with integrated silicon photonic networks," in *Design, Automation and Test in Europe Conference (DATE)*, 2020, pp. 1622–1627.

[3] P. Wesling, *Heterogeneous Integration Roadmap 2023*, 2023, Chapter 9: Integrated Photonics. [Online]. Available: https://eps.ieee.org/images/files/HIR_2023/ch09_photonics.pdf

[4] C. Condrat, P. Kalla, and S. Blair, "Logic synthesis for integrated optics," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2011, pp. 13–18.

[5] R. Wille, O. Keszocze, C. Hopfmuller, and R. Drechsler, "Reverse BDD-based synthesis for splitter-free optical circuits," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015, pp. 172–177.

[6] Z. Zhao, Z. Wang, Z. Ying, S. Dhar, R. T. Chen, and D. Z. Pan, "Logic synthesis for energy-efficient photonic integrated circuits," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 355–360.

[7] Z. Zhao, J. Gu, Z. Ying, C. Feng, R. T. Chen, and D. Z. Pan, "Design technology for scalable and robust photonic integrated circuits," in *International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[8] R. Matsuo, J. Shiomi, T. Ishihara, H. Onodera, A. Shinya, and M. Notomi, "A synthesis method for power-efficient integrated optical logic circuits towards light speed processing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 488–493.

[9] R. Matsuo and S.-i. Minato, "BDD variable ordering for minimizing power consumption of optical logic circuits," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 96–101.

[10] Z. Zhao, D. Liu, Z. Ying, B. Xu, C. Feng, R. T. Chen, and D. Z. Pan, "Exploiting wavelength division multiplexing for optical logic synthesis," in *Design, Automation and Test in Europe Conference (DATE)*, 2019, pp. 1567–1570.

[11] R. Matsuo, J. Shiomi, T. Ishihara, H. Onodera, A. Shinya, and M. Notomi, "BDD-based synthesis of optical logic circuits exploiting wavelength division multiplexing," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 203–209.

[12] G. E. Andrews, *The theory of partitions*. Cambridge university press, 1998, no. 2.

[13] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification (CAV)*, 2010, pp. 24–40.

[14] A. Mishchenko, "IWLS 2005 benchmarks," Accessed: 2024-06-16. [Online]. Available: https://people.eecs.berkeley.edu/ãlanmi/benchmarks/iwls2005/

UniCAIM: A Unified CAM/CIM Architecture with Static-Dynamic KV Cache Pruning for Efficient Long-Context LLM Inference

Weikai Xu[1#], Wenxuan Zeng[42#], Qianqian Huang[13], Meng Li[213*], and Ru Huang[13*]
[1]*School of Integrated Circuits, Peking University, China;* [2]*Institute for Artificial Intelligence, Peking University, China;*
[3]*Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China;*
[4]*School of Software and Microelectronics, Peking University, China.* [#]*Equal contribution.*

***Abstract*—Transformer-based large language models (LLMs) have achieved impressive performance in various natural language processing (NLP) applications. However, the high memory and computation cost induced by the KV cache limits the inference efficiency, especially for long input sequences. Compute-in-memory (CIM)-based accelerators have been proposed for LLM acceleration with KV cache pruning. However, as existing accelerators only support static pruning with a fixed pattern or dynamic pruning with primitive implementations, they suffer from either high accuracy degradation or low efficiency. In this paper, we propose a ferroelectric FET (FeFET)-based unified content addressable memory (CAM) and CIM architecture, dubbed as UniCAIM. UniCAIM features simultaneous support for static and dynamic pruning with 3 computation modes: 1) in the CAM mode, UniCAIM enables approximate similarity measurement in $\mathcal{O}(1)$ time for dynamic KV cache pruning with high energy efficiency; 2) in the charge-domain CIM mode, static pruning can be supported based on accumulative similarity score, which is much more flexible compared to fixed patterns; 3) in the current-domain mode, exact attention computation can be conducted with a subset of selected KV cache. We further propose a novel CAM/CIM cell design that leverages the multi-level characteristics of FeFETs for signed multi-bit storage of the KV cache and in-place attention computation. With extensive experimental results, we demonstrate UniCAIM can reduce the area-energy-delay product (AEDP) by 8.2∼831× over the state-of-the-art CIM-based LLM accelerators at the circuit level, along with high accuracy comparable with dense attention at the application level, showing its great potential for efficient long-context LLM inference.**

I. Introduction

Large language models (LLMs) have recently demonstrated remarkable performance in a wide range of applications such as question answering, code completion, and dialogue systems [1]–[3]. The context length supported by LLMs is also growing progressively to support more applications like multi-turn chat, text summarization, etc. However, with the increase in sequence lengths, the KV cache size gradually exceeds the LLM parameter size and the attention computation latency is becoming increasingly dominant, both of which emerge as prominent bottlenecks in long-context LLM inference [4], as shown in Fig. 1.

Recently, various solutions have been proposed to reduce the KV cache overhead from the algorithm perspective, utilizing the highly sparse nature of attention [5], [6]. There are static and dynamic KV cache pruning policies, aiming to either reduce memory footprint by permanently evicting unnecessary tokens or reduce the computation load by only fetching the KV pairs with high attention scores, respectively [5]–[8]. Meanwhile, from the hardware perspective, the computing-in-memory (CIM) architecture which can perform the general matrix-vector multiplication (GEMV) operations within the memory array, has been proven to compute attention efficiently by reducing the data movement [9]–[12]. Besides, some works specifically adopt KV cache pruning policies with CIM architecture to further support sparse attention computation [13]–[18].

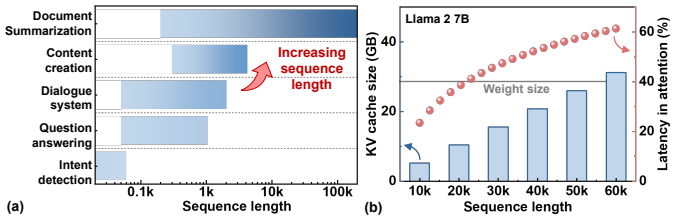


Fig. 1. (a) Various NLP tasks with increasing sequence length. (b) The impact of sequence length on KV cache size and attention latency in Llama-2-7B, which is a typical LLM, indicating the memory and computation challenges faced by long-context LLMs.

However, the current CIM-based LLM accelerators only focus on either static or dynamic KV cache pruning algorithms, hindering the simultaneous optimization of memory usage and computation overhead. Moreover, the existing hardware implementations are relatively primitive, suffering from the following challenges. On the one hand, the current CIM designs adopting static KV cache pruning only support fixed sparse attention pattern [13], [14], leading to accuracy degradation, especially for long sequences [19], [20]. Although there are advanced token-aware static KV pruning policies with better accuracy, they require the computation of accumulative attention scores, which will lead to a significant amount of extra computation and power consumption in current CIM architectures. On the other hand, to further support dynamic KV cache pruning, additional expensive operators such as top-k selection, are required, suffering from high hardware cost, latency, and energy consumption [15], [16]. Therefore, it is challenging for the current CIM architectures to efficiently support both dynamic and static KV cache pruning simultaneously, restricting the practical deployment of LLMs with increasing sequence length.

In this work, a unified content addressable memory (CAM) and CIM architecture called UniCAIM, featuring static-dynamic KV cache pruning for long-context LLM inference, is proposed through algorithm-hardware co-optimization. UniCAIM addresses the challenges mentioned above, and the main contributions can be summarized as follows:

- A hardware-friendly static-dynamic KV cache pruning framework is proposed, which can significantly improve the inference efficiency of LLMs without degrading the model. Firstly, the unimportant tokens are dropped permanently in the prefill stage, reducing the overall memory overhead. Moreover, during each decoding step, only the most relevant (i.e., top-k) tokens are selected for exact attention computation to reduce the computation load, and one token is evicted based on accumulative attention scores to enhance memory utilization and management when the generated length exceeds the reserved KV cache size.
- A UniCAIM architecture with CAM and CIM modes is proposed for efficiently implementing the static-dynamic KV cache pruning and sparse attention computing. Based on the CAM mode

This work was supported in part by NSFC under Grant 62495102 and Grant 92464104, in part by the National Key Research and Development Program under Grant 2024YFB4505004, in part by Beijing Municipal Science and Technology Program under Grant Z241100004224015, and in part by 111 Project under Grant B18001.

*Corresponding author: ruhuang@pku.edu.cn; meng.li@pku.edu.cn

of UniCAIM, attention scores can be approximately measured and compared to select the top-k tokens in $\mathcal{O}(1)$ time without the exact computations, and the attention scores can also be accumulated for static eviction by further leveraging the charge-domain CIM within the same operation cycle, leading to fast and energy-efficient static-dynamic KV cache pruning. Moreover, the current-based CIM is utilized for the exact attention score computation of selected tokens with high precision.

- The proposed UniCAIM is implemented based on emerging compact ferroelectric FET (FeFET) devices with carefully designed circuits in hardware for further area-energy-delay product (AEDP) reduction. Experiments and evaluations show that the proposed FeFET-based UniCAIM can achieve 8.2∼831× AEDP reduction compared with the state-of-the-art CIM-based LLM accelerators, showing its great potential for high speed, area- and energy-efficient long-context LLM acceleration.

II. Background

In this section, we review the sparse attention in transformer-based LLMs and exiting KV cache pruning policies from the algorithm and hardware perspectives, as well as the basics of FeFET and its advantages for CIM and CAM.

A. Sparse attention and KV cache pruning

*1) **Sparse attention and existing KV cache pruning algorithms**:* In transformers, the quadratic computational complexity of attention is one of the major bottlenecks [21]. Many recent research efforts have been devoted to exploiting the intrinsic sparsity in attention [22]. For long-context generation tasks, KV cache pruning becomes a promising solution for efficient sparse attention computation. Existing KV cache pruning methods can be roughly categorized into static pruning and dynamic pruning. Some static pruning policies predefine the positions of important tokens and remain consistent across decoding steps, such as StreamingLLM [19], while such fixed patterns lack flexibility for different LLMs and contexts. Other static pruning policies permanently drop some tokens and the dropped ones cannot be used in the subsequent decoding steps, such as SnapKV [8] and H2O [7]. However, only tokens that remain throughout the entire decoding process can be pruned, otherwise suffering significant accuracy degradation. On the other hand, dynamic pruning policies select unimportant tokens to drop at for different decoding steps, such as InfLLM [23] and LongCache [24]. Though dynamic pruning is more flexible than static pruning, it involves more expensive computations, such as attention score ranking and top-k selection, which pose greater challenges for efficient hardware implementation.

*2) **Existing hardware implementations for KV cache pruning**:* KV cache pruning in LLMs offers significant advantages by reducing memory usage and redundant computation. Recently, CIM-based transformer accelerators have attracted widespread attention for LLMs due to the largely reduced data movement overhead, with some studies exploring the implementation of KV cache pruning in hardware. On the one hand, some works implement fix-pattern KV cache pruning which is well-suited for CIM architecture, such as TranCIM [13] adopting the pruning algorithm proposed in StreamingLLM [19]. However, this approach performs computations on neighboring tokens within a predetermined fixed attention range defined in the algorithm, which lacks flexibility and fails to consider the varying contributions of different tokes to the final prediction. Consequently, the CIM with fix-pattern KV cache pruning designs suffers from suboptimal accuracy and efficiency, especially when processing different LLMs and contexts. On the other hand, some works have implemented

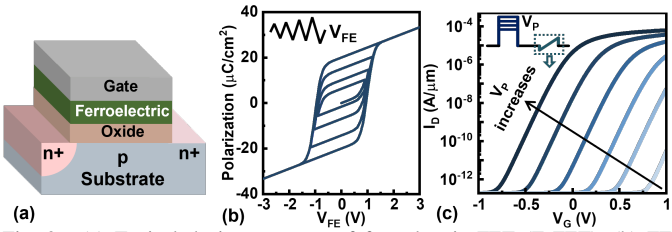


Fig. 2. (a) Typical device structure of ferroelectric FET (FeFET). (b) FE polarization-voltage loops with multilevel FE polarizations. (c) Gradually modulated I_D-V_G curves of FeFET for the multilevel storage capability.

Table I. Qualitative comparison of proposed FeFET-based UniCAIM with the state-of-the-art CIM-based LLM accelerators.

LLM accelerator	TranCIM [13]	CIMFormer [15]	Sprint [17]	UniCAIM (This work)
Technology	SRAM CIM	SRAM CIM	RRAM CIM	FeFET CAM/CIM
Static pruning	Yes (Fixed pattern)	No	No	Yes (Accumulate attention)
Dynamic pruning	No	Yes (Top-k selection)	Yes (Approximate attention)	Yes (Top-k selection)
Memory footprint reduction	Good	Bad	Medium	Excellent
Energy-efficiency improvement	Medium	Medium	Good	Excellent
Accuracy	Bad	Good	Medium	Good

dynamic KV cache pruning, primarily involving attention score ranking and selection, such as CIMFormer [15] adopting the top-k selection. However, it requires additional hardware overhead to achieve top-k selection with complex time complexity of $\mathcal{O}(nlogn)$ [25] or $\mathcal{O}(logn)$ with additional circuits for token gathering [15], resulting in high latency and power consumption. Besides, there are emerging non-volatile memories (NVMs)-based CIM designs for dynamic pruning by utilizing approximate attention scores [17], [18], but suffering the trade-off between energy efficiency and accuracy.

Therefore, current CIM-based LLM accelerators support either static or dynamic KV cache pruning policies, along with relatively primitive implementations, facing substantial penalties in terms of area, latency, and power consumption (Table I). Thus, while CIM-based KV cache optimization techniques bring new possibilities for efficient inference of long-context LLMs, there are still several challenges that need to be addressed.

B. FeFET for CIM and CAM

In recent years, various emerging NVMs, such as resistive random-access memory (RRAM), magnetic tunnel junction (MTJ) and FeFET, have triggered lots of attention for CIM, due to the high storage density and efficient GEMV operation via analog computing within the memory array [26]–[28]. On the other hand, CAM is a specialized memory architecture, where an input query is compared with entire stored entries simultaneously within the memory array [29]–[31]. Different from conventional CIM, CAM only identifies the matching entry or measures the matching degree without precise quantification, which is more energy efficient. Among the NVMs, the three-terminal FeFET has the advantages of low write energy due to the electric-field-driven write mechanism and high I_{ON}/I_{OFF} ratio due to the transistor structure, along with good CMOS compatibility [32]. Fig. 2(a) shows the structure of an FeFET device, with a HfO_2-based ferroelectric (FE) layer integrated into the gate stack of a MOSFET. Applying different program voltages (V_P) to the gate of FeFET will switch the FE polarization states (Fig. 2(b)), and thus gradually modulate the threshold voltage (V_{TH}) of FeFET, indicating the multilevel storage capability (Fig. 2(c)). Besides, by applying a relatively small read voltage (V_R), the channel conductance state can be non-destructively readout without FE polarization switching. Therefore, FeFET can achieve both storage and computation, making it a promising candidate for CIM and CAM designs [27], [31].

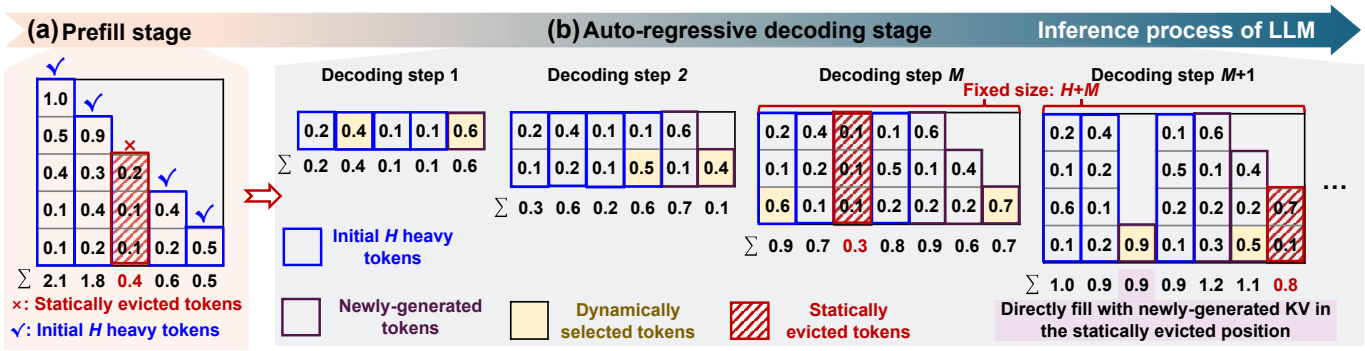


Fig. 3. Framework of the proposed hybrid static-dynamic KV cache pruning algorithm. (a) During the prefill stage, static pruning evicts unimportant tokens for the subsequent generation. (b) During the decoding stage, dynamic pruning preserves a subset of tokens for sparse attention computation, while static pruning evicts one token at each step when the generated length exceeds the reserved size for a fixed KV cache size.

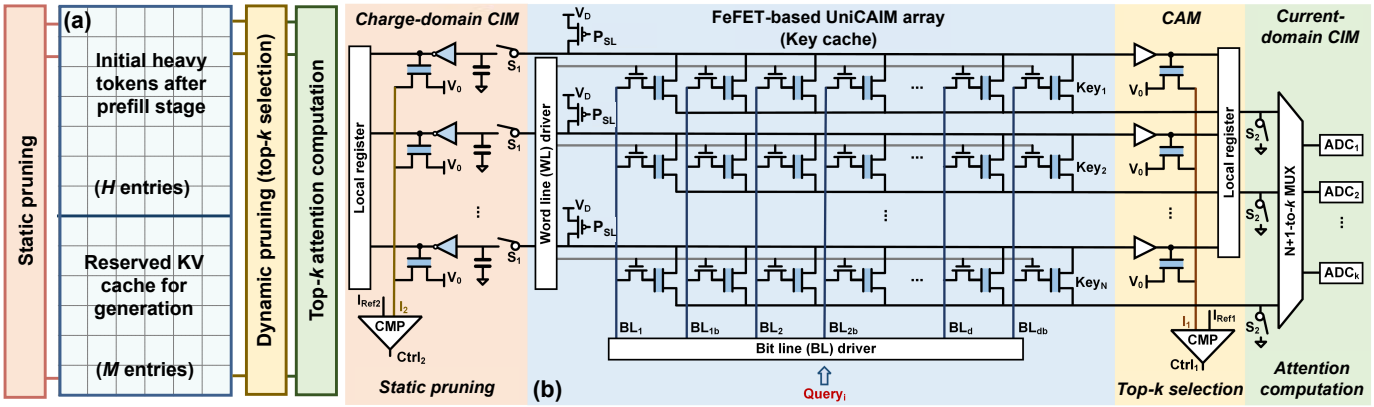


Fig. 4. (a) The proposed UniCAIM architecture for static-dynamic KV cache pruning and sparse attention computing. (b) The hardware design of UniCAIM based on FeFET, including FeFET-based UniCAIM array and carefully designed peripheral circuits for CAM, charge-domain and current-domain CAM.

III. UniCAIM Architecture with Static-Dynamic KV Cache Pruning

A. Hybrid Static-Dynamic KV Cache Pruning Algorithm

The framework of the proposed hybrid pruning algorithm is illustrated in Fig. 3, which combines static pruning during the prefill stage for an overall memory footprint reduction and static-dynamic pruning during the decoding stage for further efficient sparse attention.

*1) **Prefill stage with one-shot static pruning**:* During the prefill stage, we leverage static pruning to drop a part of tokens permanently which are almost unimportant throughout the entire decoding process, and remain the heavy tokens. Following [5], [7], [8], we determine which tokens to statically drop based on the accumulative attention scores as shown in Fig. 3(a). Specifically, we drop the tokens with the lowest accumulated attention scores, reducing the overhead of the entire inference process.

*2) **Decoding stage with step-wise static-dynamic pruning**:* Since the LLM attention exhibits high sparsity [5] and different queries can attend to different tokens [6], we propose to further selectively choose a subset of important tokens for efficient attention computation at different decoding steps (Fig. 3(b)). We adopt the most commonly used Cosine similarity as the attention score ($Attn$) to evaluate the importance of the KV cache for previous tokens as follows:

$$Attn(q, K) = q \cdot K^T, \tag{1}$$

where $q \in \mathbb{R}^{h\times 1\times d}$ denotes the query at the current step, $K \in \mathbb{R}^{h\times N\times d}$ denotes the key cache, and h, N, d represent the number of heads, the number of tokens in key cache, and hidden dimension, respectively. After the similarity measure, we leverage the top-k selection strategy to pick up the most important tokens for the subsequent sparse attention.

Moreover, when considering the memory limitation and hardware constraints, the consistently increasing KV cache size during the decoding process is not friendly. To address this problem, we always keep a table of the accumulated attention scores and statically drop one token with the lowest accumulated attention score when the generated length exceeds the reserved size for decoding, and directly fill with the newly-generated token in the statically evicted position. As a result, the KV cache size is fixed with enhanced memory utilization and management.

B. FeFET-based UniCAIM Architecture

*1) **Overview of FeFET-based UniCAIM**:* Fig. 4(a) shows the proposed UniCAIM architecture for static-dynamic KV cache pruning, which is specially optimized for the auto-regressive decoding stage, due to the memory-bound nature of the decoding stage. After one-shot static pruning at the prefill stage, H heavy tokens with the highest accumulated attention scores are retained and stored in the UniCAIM array, and M entries are reserved for newly generated tokens at the decoding stage. During each encoding step, the approximate attention scores are evaluated for dynamically selecting the top-k tokens with the highest scores, which will be accurately computed. Additionally, the accumulated attention scores are also evaluated for statically evicted the token with the lowest scores, when the number of generated tokens exceeds M. The step-wise static pruning during the decoding stage can maintain a fixed-size KV cache, resulting in higher area efficiency and better memory utilization. In this work, the proposed UniCAIM architecture is implemented using emerging FeFET in hardware (Fig. 4(b)), incorporating an FeFET-based UniCAIM array which is shared by CAM and CIM, along with

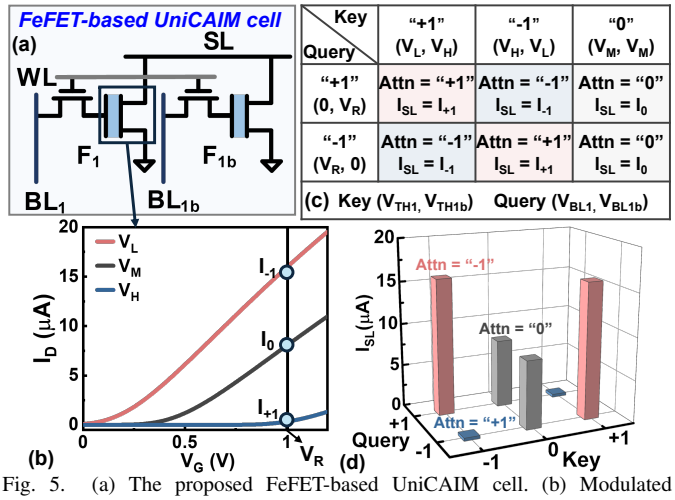
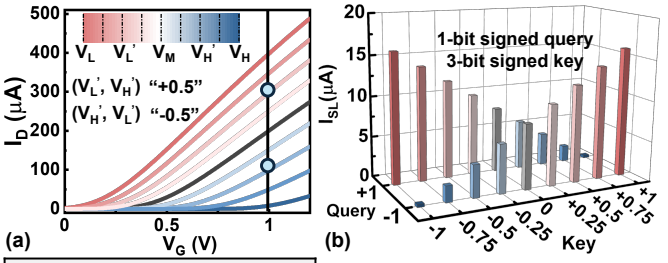


Fig. 5. (a) The proposed FeFET-based UniCAIM cell. (b) Modulated threshold voltages (V_{TH}) of FeFET. (c) The truth table of signed key and query. (d) The computing results of local signed multiplication.



(c) Multilevel query expansion

Query	$Cell_1$	$Cell_2$	$Cell_3$	$Cell_4$
"+1"	$(0, V_Q)$	$(0, V_Q)$	$(0, V_Q)$	$(0, V_Q)$
"+0.5"	$(V_Q, 0)$	$(0, V_Q)$	$(0, V_Q)$	$(0, V_Q)$
"0"	$(V_Q, 0)$	$(V_Q, 0)$	$(0, V_Q)$	$(0, V_Q)$
"-0.5"	$(V_Q, 0)$	$(V_Q, 0)$	$(V_Q, 0)$	$(0, V_Q)$
"-1"	$(V_Q, 0)$	$(V_Q, 0)$	$(V_Q, 0)$	$(V_Q, 0)$

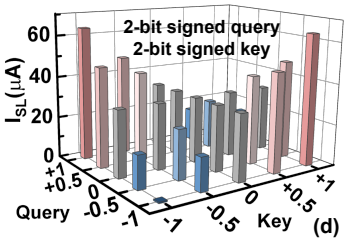


Fig. 6. (a) Gradually modulated V_{TH} of FeFET for in-place multilevel signed key expansion. (b) The computing results of local signed multiplication between 1-bit signed query and 3-bit signed key. (c) The proposed encoding method for multilevel query expansion. (d) The computing results of local signed multiplication between 2-bit signed query and 2-bit signed key.

carefully designed peripheral circuits dedicated to dynamic pruning, static pruning, and attention computation.

*2) **FeFET-based UniCAIM cell**:* The proposed FeFET-based UniCAIM cell is composed of two 1-transistor-1-FeFET (1T1F) units (Fig. 5(a)), which can store the signed key and implement the local signed multiplication (i.e., attention score) between signed key and query. In the write stage, the signed key of "-1"/"+1" is programmed in two FeFETs (F_1 and F_{1b}) with complementary V_{TH} states (V_{TH1} and V_{TH1b}) by applying relatively large V_P for FE polarization switching (Fig. 5(b) and (c)). Then the signed query of "-1"/"+1" is represented by complementary non-destructive read voltages at bit-lines (BL_1 and BL_{1b}) with relatively small amplitudes. The signed multiplication result can be expressed through sense-line current (I_{SL}), where the low/high I_{SL} represents the result of "+1"/"-1", which is meticulously designed for efficient KV-cache pruning and attention computation which will be discussed in the following sections. Moreover, the Key of "0" can be represented by programming both FeFETs to the medium V_{TH} states, which will result in medium I_{SL} with both query inputs (Fig. 5(d)).

Furthermore, by utilizing the multilevel storage capability of

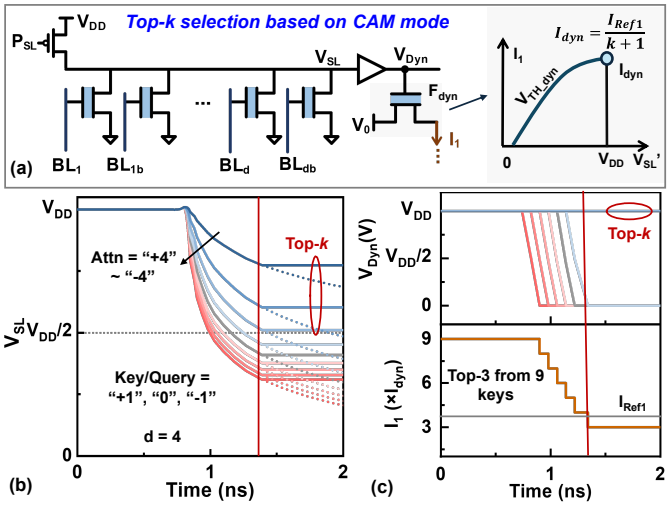


Fig. 7. (a) The main circuit of CAM mode for top-k selection. (b) The discharge speed of the sense-line (SL) is correlated with the attention scores, where the SL with higher attention discharges more slowly. (c) The sequence diagrams of top-k selection take the example of top-3 selection from 9 keys.

FeFET, the proposed FeFET-based UniCAIM cell supports signed multi-bit storage and in-place attention computation for higher area-efficiency. With multiple V_P applied to the gate of FeFET, the FE polarization can be gradually switched in succession, resulting in the continuous modulation of V_{TH}, and the multiple V_{TH} pairs (V_{TH1}, V_{TH1b}) in complementary form represent multilevel signed key states (Fig. 6a). For example, the (V_L', V_H') and (V_H', V_L') can represent keys of "+0.5" and "-0.5", respectively. The signed multiplication results with 3-bit signed keys are shown in Fig. 6b. Besides, the multilevel signed query expansion, including "0", can be implemented through bitwise expansion with the proposed mapping method as shown in Fig. 6(c), enabling signed multiplication with multilevel signed queries and keys (Fig. 6(d)).

*3) **CAM mode of UniCAIM for dynamic pruning**:* To implement the dynamic KV cache pruning, the CAM mode of UniCAIM is designed for fast and energy-efficient top-k selection. As shown in Fig. 7a, each row shares one precharge p-type transistor for precharging the SL and detecting circuits including one buffer and one FeFET (F_{dyn}) with properly programmed V_{TH} for top-k selection. In the dynamic pruning phase, all SLs are precharged to high voltage (i.e., V_{DD}) by setting the P_{SL} to ground, and then all BLs are prechraged to the corresponding voltages according to the proposed mapping method of the input query. Based on the proposed UniCAIM cell, the higher similarity between the query and the key (i.e., the higher attention score) will lead to the smaller I_{SL}, and thus result in a slower discharge speed of the SL. For a key/query with a dimension of 4, where each state is "+1", "-1" or "0", there are 9 possible attention scores ranging from "-4" to "+4", and the corresponding SL discharge processes for different scores are shown in Fig. 7b. The SLs with lower similarity will discharge to $V_{DD}/2$ more quickly, resulting in V_{Dyn} switching to ground, which turns off the corresponding F_{dyn}. In contrast, the SLs with higher similarity maintain V_{Dyn} at V_{DD}, which turns on the F_{dyn} with I_{dyn}. By setting the I_{Ref1} to $(k+1)I_{dyn}$, top-k selection can be performed, as shown in the example of top-3 selection from 9 keys (Fig. 7(c)). When k SLs with higher similarity maintain high voltage, the accumulated I_1 will be less than I_{Ref1}, causing the control signal ($Ctrl_1$) to switch (Fig. 4), which will disable the discharge process of SLs. The addresses of the top-k selected keys are stored in the local register for exact similarity computation in subsequent steps.

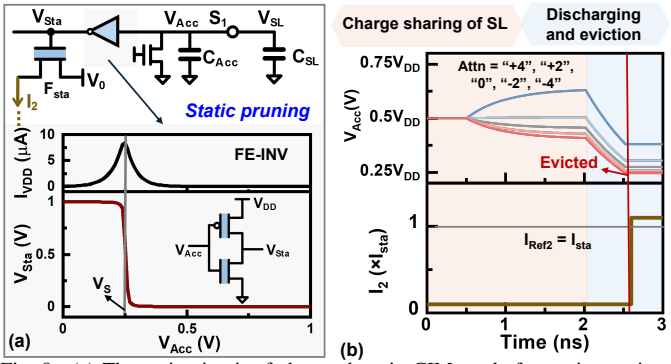


Fig. 8. (a) The main circuit of charge-domain CIM mode for static pruning. (b) The process and sequence diagrams of static pruning, including charge sharing of SL for accumulating attention scores and static eviction.

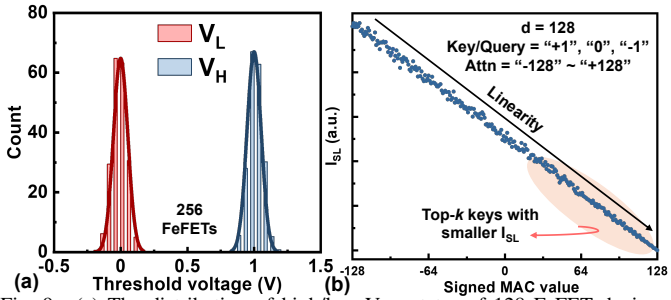


Fig. 9. (a) The distribution of high/low V_{TH} states of 128 FeFET devices. (b) The I_{SL} is linear to the singed multiply-accumulate (MAC) value.

The proposed FeFET-based CAM can achieve top-k selection with $\mathcal{O}(1)$ time complexity by comparing the similarity without the need for actual computation, which improves the speed and energy-efficiency compared with conventional designs. Moreover, k can be easily configured by programming the F_{dyn} without the additional hardware overhead, indicating the enhanced versatility for different LLMs and contexts.

*4) **Charge-domain CIM mode of UniCAIM for static pruning**:* During the decoding stage, different tasks will generate varying lengths of tokens, which poses significant challenges to memory overhead and management. Therefore, in this work, when the number of generated tokens exceeds the reserved KV cache size (i.e., M), the static KV cache pruning is applied. The charge-domain CIM is further designed for accumulating the similarity and evicting the smallest one. After implementing top-k selection based on CAM mode, the switch S_1 is closed and the charge sharing occurs between SL capacitor (C_{SL}) and accumulate capacitor (C_{Acc}) which has accumulated the previous similarity scores (Fig. 8(a)). To implement static pruning within the same operation cycle along with dynamic pruning, without the need to recharge SLs, an FeFET-based inverter (FE-INV) is designed with programmed switching voltage (V_S). The SL with the smallest accumulated similarity will discharge to V_S firstly through the discharge transistor, which will turn on the corresponding F_{sta} with I_{sta}. Consequently, the accumulated I_2 is larger than I_{Ref2} which is equal to I_{sta}, causing the control signal ($Ctrl_2$) to switch, which will turn off the discharge transistor (Fig. 8(b)). The address of the statically selected key is stored in the local register, which will be evicted after exact similarity computation at the current generation step. Moreover, to avoid swapping memory when the stored key is evicted, the newly-added key is directly overwritten by enabling the word-line (WL) of the corresponding row and applying the appropriate V_P on BLs with a single write cycle.

*5) **Current-domain CIM mode of UniCAIM for attention computation**:* To implement the exact attention computation for the top-k selected tokens after dynamic-static KV cache pruning, the FeFET-based current-domain CIM is proposed, where the I_{SL} is precisely quantized via the analog-to-digital converter (ADC) to calculate the accurate attention score. Fig. 9 shows the I_{SL} with 128 activated FeFET-based UniCAIM cells, which shows robust linearity with respect to the singed multiply-accumulate (MAC) value, when considering the device-to-device variation of FeFET devices with a standard deviation of 54mV [33]. Benefiting from the dynamic pruning, only the I_{SL} of top-k most similar keys need to be quantized by the ADCs, which can be selected through a Multiplexer (MUX). It can significantly reduce the computational overhead and the power consumption of the ADC, which is the primary source of power usage in analog CIM systems. Moreover, due to the meticulous design of FeFET-based UniCAIM cell, the I_{SL} is smaller when the attention score is larger, and thus the dynamically selected top-k tokens which need to be precision computed have relatively small I_{SL}, leading to lower energy consumption for attention computation.

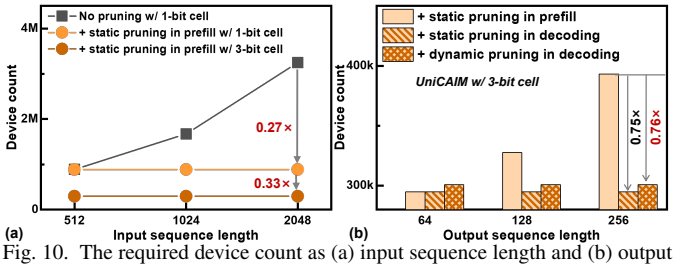


Fig. 10. The required device count as (a) input sequence length and (b) output sequence length increases with different pruning conditions.

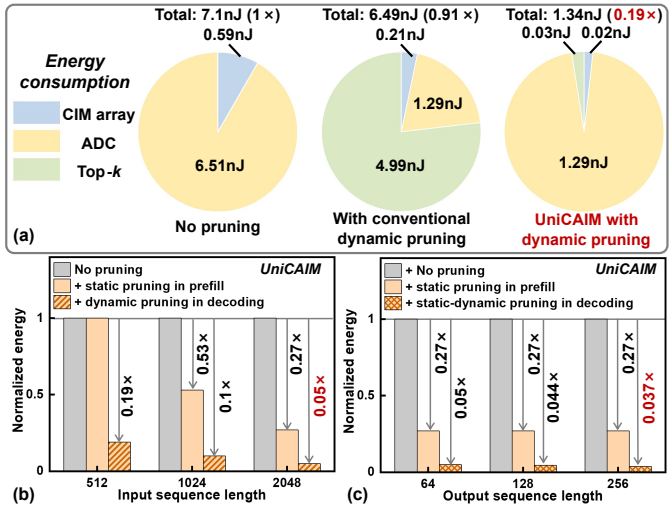


Fig. 11. (a) The impact of dynamic pruning on power consumption, with a 20% pruning ratio. The energy consumption as (b) input sequence length with an output sequence length of 64 and (c) output sequence length with an input sequence length of 2048 increases with different pruning conditions.

IV. Experimental Results

In this section, we validate and evaluate the proposed FeFET-based UniCAIM architecture for LLM acceleration. Firstly, the key performance metrics for attention including area, energy, and delay are evaluated and compared with the state-of-the-art CIM-based designs at the circuit level. Subsequently, the impact of the proposed dynamic-static KV cache pruning algorithm on the accuracy of LLM applications is evaluated.

A. Circuit-level Evaluations

The circuit-level experiments and evaluations of proposed FeFET-based UniCAIM are carried out based on typical emerging FeFET devices with circuit simulation in HSPICE. The 45nm BSIM model

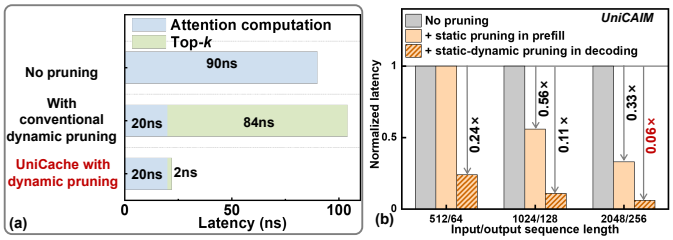


Fig. 12. (a) The impact of dynamic pruning on latency, with 20% pruning ratio, 512 input sequence length, and 64 output sequence length. (b) The comparison of latency as input and output sequence length increase with different pruning conditions.

Table II. Quantitative comparison of proposed FeFET-based UniCAIM with the state-of-the-art CIM-based LLM accelerators.

LLM accelerator		CIMFormer [15]	TranCIM [13]	Sprint [17]	UniCAIM (This work)
Technology		SRAM CIM	SRAM CIM	RRAM CIM	FeFET CAM/CIM
AEDP reduction	No pruning	1×	1×	2.5×	3~9×
	50% pruning	1×	8.9×	15×	124~372×
	80% pruning	1×	14.6×	24×	277~831×

[34] is used for all MOSFETs, and the Preisach ferroelectric switching model [35] is used for FeFETs. The parasitic wire capacitance is extracted according to [36]. The KV cache size contains 576 tokens, with 512 initial heavy tokens and 64 reserved tokens for decoding. Each token has a hidden dimension (i.e., d) of 128 with a 3-bit UniCAIM cell, and the range of attention scores is "-512" to "512", and thus a 10-bit SAR ADC is used for quantization [37].

*1) **Area**:* The static KV cache pruning during the prefill/decoding stage can reduce the KV cache size corresponding to the input/output sequence, leading to improved area efficiency, and the improvement is more significant as the input/sequence length increases due to the higher compression ratio (Fig. 10). It indicates that static KV cache pruning is the key technique for improving area efficiency. Moreover, benefiting from the proposed FeFET-based UniCAIM cell with *in-situ* multilevel expansion capability, the area efficiency can be further improved. Besides, the proposed CAM-based dynamic pruning circuit is highly compact, resulting in only a slight decrease in the improvement of area efficiency from 15× to 14.7× (Fig. 10(b)).

*2) **Energy**:* To implement dynamic pruning, conventional designs involve approximate attention computation, followed by selecting the top-k highest similar tokens with additional circuits, limiting the energy efficiency improvement. In this work, the FeFET-based CAM mode is designed for dynamic pruning, which evaluates the relative attention and selects the top-k tokens through a single SL charge-discharge process, without the need for actually calculating attention scores and ranking them. It significantly improves energy efficiency, due to reducing the energy consumption overhead associated with ADCs, and avoiding the additional power needed for top-k selection circuits (Fig. 11(a)). On the other hand, the static KV cache pruning reduces the number of tokens, thereby naturally reducing energy consumption. Moreover, the improvement of energy-efficiency is more significant from 5.3× to 27× as the input and output sequence length increase, benefiting from the static pruning during both prefill and decoding stages (Fig. 11(b) and (c)).

*3) **Delay**:* Although CIM architecture enables parallel computation in theory, the output parallelism is constrained by the maximum number of ADCs that can be accommodated within the area and power budget of the system, which further increases the compute delay. In this work, 64 SLs are sensed in parallel with 64 ADCs. As shown in Fig. 12(a), although the conventional dynamic KV cache pruning reduces the number of attention scores that need precision

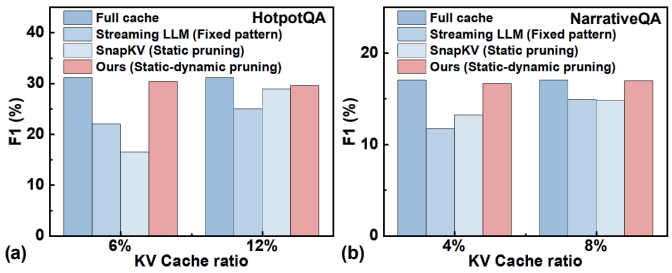


Fig. 13. Accuracy evaluation of KV cache pruning on different datasets of (a) HotpotQA and (b) NarrativeQA, where F1 is a widely adopted evaluation metric that measures the similarity of the outputs to the ground-truth answers for QA and summarization tasks.

quantization, the approximate attention computation is still limited by the number of ADCs. Additionally, the top-k selection process involves complex $\mathcal{O}(n \cdot \log n)$ time complexity, which actually increases the overall computation latency. The proposed FeFET-based UniCAIM does not require ADC quantization during dynamic pruning and can implement the top-k selection with $\mathcal{O}(1)$ time complexity, thereby further reducing computation latency. Moreover, the static KV cache pruning reduces the number of attention scores that require ADC quantization, thereby reducing computation latency. Moreover, the speed up is more significant from 4.2× to 16.7× as the input and output sequence length increase, benefiting from the static KV cache pruning during the prefill stage and static-dynamic KV cache pruning decoding stages (Fig. 12(b)).

*4) **Comparison with state-of-the-art designs**:* We use the recently reported CIM-based LLM accelerators CIMFormer [15], TranCIM [13], and Sprint [17] as the baselines for quantitative comparison, and we apply the same static/dynamic KV cache pruning ratio across different designs for a fair comparison. The proposed FeFET-based UniCAIM with static-dynamic KV cache pruning achieves a significant reduction in the AEDP (Table II). With 1-bit FeFET-based UniCAIM cell, the AEDP is reduced by 8.2×/13.9×/124× and 11.5×/19×/277× with the pruning ratio of 50% and 80% compared with Sprint/TranCIM/CIMFormer, respectively. Moreover, When further utilizing a 3-bit FeFET-based UniCAIM cell, the AEDP reduction is even more pronounced, which achieves 24.8×/41.7×/372× and 34.6×/56.9×/831×, indicating its great potential for long-context LLM acceleration at the edge.

B. Application-level Evaluation

We evaluate our proposed static-dynamic KV cache pruning algorithm on the LongChat-v1.5-7B-32k [38] model on widely used long-context tasks from LongBench [39], including HotpotQA and NarrativeQA. The prompt length of HotpotQA is 1.5k and the prompt length of NarrativeQA is 2.5k. As can be observed in Fig. 13, our proposed static-dynamic KV cache pruning algorithm does not significantly affect the accuracy of LLM and achieves comparable accuracy with full-cache attention even under low KV cache ratios. When compared with the recent KV cache pruning algorithms SnapKV [8] and StreamingLLM [19], our algorithm consistently achieves higher accuracy, indicating the effectiveness of our algorithm.

V. Conclusion

In this work, a novel FeFET-based UniCAIM architecture featuring dynamic-static KV cache pruning is proposed for sparse attention. The UniCAIM architecture can efficiently implement dynamic pruning, static pruning, and attention computation, with different CAM and CIM modes. Evaluation results at circuit and application levels show significantly reduced AEDP and high accuracy, indicating its great potential for long-context LLM acceleration at the edge.

REFERENCES

[1] Z. Yi, J. Ouyang, Y. Liu, T. Liao, Z. Xu, and Y. Shen, "A survey on recent advances in llm-based multi-turn dialogue systems," *arXiv preprint arXiv:2402.18013*, 2024.

[2] S. Sudhakaran, M. González-Duque, M. Freiberger, C. Glanois, E. Najarro, and S. Risi, "Mariogpt: Open-ended text2level generation through large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[3] D. Van Veen, C. Van Uden, L. Blankemeier, J.-B. Delbrouck, A. Aali, C. Bluethgen, A. Pareek, M. Polacin, E. P. Reis, A. Seehofnerová *et al.*, "Adapted large language models can outperform medical experts in clinical text summarization," *Nature medicine*, vol. 30, no. 4, pp. 1134–1142, 2024.

[4] S. Luohe, Z. Hongyi, Y. Yao, L. Zuchao, and Z. Hai, "Keep the cost down: A review on methods to optimize llm's kv-cache consumption," *arXiv preprint arXiv:2407.18003*, 2024.

[5] Y. Zhao, D. Wu, and J. Wang, "Alisa: Accelerating large language model inference via sparsity-aware kv caching," *arXiv preprint arXiv:2403.17312*, 2024.

[6] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," *arXiv preprint arXiv:2406.10774*, 2024.

[7] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[8] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "Snapkv: Llm knows what you are looking for before generation," *arXiv preprint arXiv:2404.14469*, 2024.

[9] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[10] L. Xu, S. Yuan, D. Wang, Y. Chen, X. Li, and Y. Sun, "Heirs: Hybrid three-dimension rram-and sram-cim architecture for multi-task transformer acceleration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[11] Z. Lu, X. Wang, M. T. Arafin, H. Yang, Z. Liu, J. Zhang, and G. Qu, "An rram-based computing-in-memory architecture and its application in accelerating transformer inference," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[12] X. Yang, B. Yan, H. Li, and Y. Chen, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[13] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "Trancim: Full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 6, pp. 1798–1809, 2022.

[14] F. Tu, Y. Wang, Z. Wu, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "16.4 tensorcim: A 28nm 3.7 nj/gather and 8.3 tflops/w fp32 digital-cim tensor processor for mcm-cim-based beyond-nn acceleration," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 254–256.

[15] R. Guo, X. Chen, L. Wang, Y. Wang, H. Sun, J. Wei, H. Han, L. Liu, S. Wei, Y. Hu *et al.*, "Cimformer: A systolic cim-array-based transformer accelerator with token-pruning-aware attention reformulating and principal possibility gathering," *IEEE Journal of Solid-State Circuits*, 2024.

[16] F. Tu, Z. Wu, Y. Wang, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "16.1 muitcim: A 28nm 2.24 μj/token attention-token-bit hybrid sparse digital cim-based accelerator for multimodal transformers," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 248–250.

[17] A. Yazdanbakhsh, A. Moradifirouzabadi, Z. Li, and M. Kang, "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 744–762.

[18] Q. Zheng, S. Li, Y. Wang, Z. Li, Y. Chen, and H. H. Li, "Accelerating sparse attention with a reconfigurable non-volatile processing-in-memory architecture," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[19] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[20] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[21] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[22] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin *et al.*, "Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention," *arXiv preprint arXiv:2407.02490*, 2024.

[23] C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, Z. Liu, S. Han, and M. Sun, "Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory," *arXiv preprint arXiv:2402.04617*, 2024.

[24] X. Liu, Q. Guo, Y. Song, Z. Liu, K. Lv, H. Yan, L. Li, Q. Liu, and X. Qiu, "Farewell to length extrapolation, a training-free infinite context with finite attention scope," *arXiv preprint arXiv:2407.15176*, 2024.

[25] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.

[26] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[27] G. Yin, Y. Cai, J. Wu, Z. Duan, Z. Zhu, Y. Liu, Y. Wang, H. Yang, and X. Li, "Enabling lower-power charge-domain nonvolatile in-memory computing with ferroelectric fets," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 7, pp. 2262–2266, 2021.

[28] M. M. Frank, N. Li, M. J. Rasch, S. Jain, C.-T. Chen, R. Muralidhar, J.-P. Han, V. Narayanan, T. M. Philip, K. Brew *et al.*, "Impact of phase-change memory drift on energy efficiency and accuracy of analog compute-in-memory deep learning inference," in *2023 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2023, pp. 1–10.

[29] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE journal of solid-state circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[30] W. Xu, J. Luo, Q. Huang, and R. Huang, "Compact and efficient cam architecture through combinatorial encoding and self-terminating searching for in-memory-searching accelerator," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[31] X. S. Hu, M. Niemier, A. Kazemi, A. F. Laguna, K. Ni, R. Rajaei, M. M. Sharifi, and X. Yin, "In-memory computing with associative memories: A cross-layer perspective," in *2021 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2021, pp. 25–2.

[32] S. Dutta, H. Ye, A. A. Khandker, S. G. Kirtania, A. Khanna, K. Ni, and S. Datta, "Logic compatible high-performance ferroelectric transistor memory," *IEEE Electron Device Letters*, vol. 43, no. 3, pp. 382–385, 2022.

[33] J. Cai, M. Imani, K. Ni, G. L. Zhang, B. Li, U. Schlichtmann, C. Zhuo, and X. Yin, "Energy efficient data search design and optimization based on a compact ferroelectric fet content addressable memory," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 751–756.

[34] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[35] K. Ni, M. Jerry, J. A. Smith, and S. Datta, "A circuit compatible accurate compact model for ferroelectric-fets," in *2018 IEEE symposium on VLSI technology*. IEEE, 2018, pp. 131–132.

[36] H. Bhardwaj, S. Jain, and H. Sohal, "Power optimization using current-mode signalling technique for iot applications," *Measurement: Sensors*, vol. 24, p. 100494, 2022.

[37] C.-C. Liu, S.-J. Chang, G.-Y. Huang, Y.-Z. Lin, C.-M. Huang, C.-H. Huang, L. Bu, and C.-C. Tsai, "A 10b 100ms/s 1.13 mw sar adc with binary-scaled error compensation," in *2010 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2010, pp. 386–387.

[38] D. Li, R. Shao, A. Xie, Y. Sheng, L. Zheng, J. Gonzalez, I. Stoica, X. Ma, and H. Zhang, "How long can context length of open-source llms truly promise?" in *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

[39] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou *et al.*, "Longbench: A bilingual, multitask benchmark for long context understanding," *arXiv preprint arXiv:2308.14508*, 2023.

CMFuzz: Parallel Fuzzing of IoT Protocols by Configuration Model Identification and Scheduling

Qi Xu[†], Fuchen Ma[†] ✉, Yuanliang Chen[†], Wanli Chen[‡], Feifan Wu[†], Yanyang Zhao[†] ✉, Heyuan Shi[‡], Yu Jiang[†]
[†] KLISS, BNRist, Software of Software, Tsinghua University, Beijing, China
[‡] School of Electronic Information, Central South University, Changsha, Hunan, China

***Abstract*—IoT protocols are essential for the communication among diverse devices. In real-world scenarios, IoT protocols utilize flexible configurations to meet various use cases. These configurations can significantly impact the protocols' execution paths, with many bugs emerging only under specific configurations. Fuzzing has become a prominent technique for uncovering vulnerabilities in IoT protocol implementations. However, traditional fuzzing approaches are typically conducted using fixed or default configurations, overlooking potential issues that might arise in different settings. This limitation can lead to missing critical bugs that appear only under alternative configurations.**

In this paper, we propose CMFuzz, a parallel fuzzing framework designed to improve fuzzing effectiveness of IoT protocols through configuration identification and scheduling. CMFuzz first constructs a generalized protocol configuration model by systematically extracting configuration items from protocol implementations. Then, based on this model, CMFuzz defines the relations among configuration items and introduces a relation-aware allocation mechanism to distribute them across parallel fuzzing instances. For evaluation, We implement CMFuzz on top of the widely-used protocol fuzzer Peach and conduct experiments on six popular IoT protocols. Compared to the original parallel mode of Peach and state-of-the-art parallel protocol fuzzer SPFuzz, CMFuzz covers an average of 34.4% and 28.5% more branches within 24 hours. Additionally, CMFuzz has detected 14 previously-unknown bugs in these real-world IoT protocols.

***Index Terms*—IoT Protocols, Parallel Fuzzing, Configuration Model Identification and Scheduling.**

I. Introduction

The Internet of Things (IoT) refers to a network of interconnected devices that communicate and exchange data using various protocols to provide advanced services. These protocols enable essential functionalities across smart homes, healthcare, and industrial systems, highlighting their significance in modern technology [1]–[4]. However, IoT protocols are also vulnerable to cyber threats, particularly in critical infrastructure systems, where disruptions can impair operations, cause financial loss, or jeopardize safety [5], [6]. Ensuring the security of IoT protocols is therefore crucial for maintaining uninterrupted services and protecting public safety.

In IoT systems, protocols are inherently configuration-sensitive, as they are designed to operate across diverse scenarios and meet a variety of operational requirements. These protocols often offer extensive and diverse configuration options, where different combinations can influence the system's execution paths and behaviors. Consequently, certain vulnerabilities may only surface under specific configurations. For example, CoAP [7] supports block-wise transfers to handle large payloads by dividing them into smaller, sequential blocks. This feature is an optional configuration that significantly impacts the protocol's execution. When enabled, the protocol introduces additional logic to manage stateful interactions by tracking individual block transfers across multiple requests and responses. This added complexity increases the likelihood of issues such as memory leaks or resource exhaustion.

✉Fuchen Ma and Yanyang Zhao are the corresponding authors.

Fuzzing is a testing technique used to detect vulnerabilities in software by feeding volumes of inputs and observing the system's behavior. It plays a crucial role in identifying security flaws in IoT protocol implementations. Existing protocol fuzzers often build data models and state models to start the fuzzing process. A data model defines the structure and format of protocol inputs. While a state model describes the sequential flow of states that the protocol follows. Protocol fuzzers like Peach [8] and SPFuzz [9] leverage these two models to guide the generation of inputs and manage transitions between protocol states. As a result, they can systematically explore protocol behaviors and uncover a range of vulnerabilities.

However, traditional fuzzers neglect the configuration model of protocols, which sets the basic environment of the protocol instance. Thus, they overlook potential behaviors under alternative configurations in IoT systems. This limitation hinders the effective exploration of deeper protocol vulnerabilities that may arise in non-default settings. Therefore, it is essential to acknowledge the multi-configuration nature of IoT systems and incorporate multiple configurations during fuzzing under the guidance of configuration models. In fact, to explore protocol execution paths under various configurations, a straightforward approach is to run parallel fuzzing instances with different configurations. However, this presents two key challenges:

The first challenge is how to generate high quality configurations for fuzzing. IoT protocols are designed with rich and complex configurations to support a wide variety of use cases and devices. The configurations span multiple domains, including security settings, network parameters, resource management, application-specific options, and so on. Each of these configurations can affect the protocol's behavior and execution path, potentially revealing vulnerabilities. Automatically constructing realistic and comprehensive configurations that reflect real-world scenarios is a difficult task. **The second challenge** is how to divide configuration item combinations between different parallel instances. Configuration items often have dependencies, synergies, or conflicts. Arbitrary assignment across instances can lead to inefficiencies or initialization failures. The challenge, therefore, is to quantify the relations among configuration items and leverage them to group the items into cohesive groups.

To address the above mentioned challenges, we propose CMFuzz. First, in addition to the traditional data model and state model, CMFuzz introduces a generalized configuration model for protocol fuzzing. Given that IoT protocols may define configurations in different forms, such as configuration files and Command-Line Interface (CLI) options, CMFuzz systematically extracts and standardizes these configuration items to build a unified model. This abstraction accommodates different configuration formats and paradigms, ensuring adaptability across diverse IoT protocols. By decomposing protocol configurations into individual items, the model enables precise exploration of parameter spaces. Then, based on

the generalized configuration model, CMFUZZ defines the pairwise relations among configuration items and introduces a relation-aware allocation mechanism to distribute them across parallel fuzzing instances. In this way, CMFUZZ can effectively explore diverse and meaningful configuration spaces with minimal redundancy.

For evaluation, we implement CMFUZZ on top of the widely-used parallel protocol fuzzer Peach. We evaluate it on six real-world IoT protocol implementations. The results show that CMFUZZ improves the total number of branches covered by 34.4% on Peach and 28.5% on SPFuzz. It also achieves a significant speedup in reaching identical code coverage compared to baseline fuzzers. Furthermore, CMFUZZ has detected 14 previously unknown bugs in real-world IoT protocols. These critical vulnerabilities could lead to security breaches, crashes, or denial of service, posing significant risks to devices operating with these protocols in real-world environments.

Our main contributions are as follows:

- We introduce the configuration model to protocol fuzzing and propose a method to construct it, enabling exploration beyond default setups to uncover additional vulnerabilities.
- We consider dependencies between configurations and design a relation-aware allocation mechanism to effectively partition configuration spaces across parallel fuzzing instances.
- We implement and evaluate CMFUZZ on six widely used IoT protocol implementations. The results demonstrate that CMFUZZ outperforms the state-of-the-art fuzzers and has exposed 14 security-critical vulnerabilities.

II. BACKGROUND

A. IoT Protocols

The Internet of Things (IoT) has rapidly expanded, connecting billions of devices across various domains, from home automation and healthcare to industrial control systems. IoT protocols are essential for facilitating communication and interoperability among these devices, with each protocol adapted to meet the distinct requirements of IoT environments. Commonly used IoT protocols, such as MQTT, CoAP, and DDS, are designed to operate efficiently in constrained environments, supporting lightweight messaging, real-time data handling, and flexible Quality of Service (QoS) controls. These protocols typically provide mechanisms to ensure data reliability, adaptability to diverse network conditions, and interoperability across devices with varying capabilities.

Generally, IoT protocols are designed with a high degree of configurability to address the diverse and dynamic nature of IoT environments. Configurations, typically applied through configuration files or command-line interface (CLI) options, allow protocols to be tailored to match specific application requirements, device capabilities, and network conditions. For instance, security settings in these protocols may include options for different authentication methods, encryption algorithms, and key management strategies to ensure data integrity and confidentiality across the network. Quality of Service (QoS) parameters may specify message delivery guarantees, balancing factors such as latency and reliability. Network configurations define addressing schemes and routing protocols that support scalability and efficient communication. Resource management configurations enable protocols to optimize power consumption, bandwidth usage, and processing load to suit devices with limited resources.

B. Protocol Fuzzing

Fuzzing is a widely used technique to identify vulnerabilities in network protocol implementations by generating and injecting unexpected or malformed packets into the protocol to observe how it responds. Based on how packets are generated, fuzzers can be categorized into two types: mutation-based and generation-based. Mutation-based fuzzers generate test cases by randomly mutating existing valid inputs from a predefined corpus, applying various transformations such as bit flipping, field truncation, or inserting unexpected values. This approach simplifies fuzzing setup, as it does not rely on protocol specifications. In contrast, generation-based fuzzers construct inputs according to the protocol's structure and rules, producing well-formed messages that can effectively target specific states or fields. This allows them to thoroughly exercise the protocol's logic and identify vulnerabilities in various scenarios, making them particularly effective for complex protocols with structured interactions.

Traditional protocol fuzzers are typically based on two models: the data model and the state model. The data model defines the structure and format of packet fields, such as field lengths, types, and value ranges, allowing the fuzzer to craft near-valid protocol messages. The state model captures interaction sequences and transitions within the protocol, simulating how states change in response to different inputs. By adhering to protocol data formats, these two models help fuzzers produce highly tailored, protocol-compliant inputs, reaching deeper, less frequently accessed protocol states. This structured approach is especially advantageous for protocols with intricate dependencies or stateful interactions, as it ensures comprehensive coverage of both common and edge-case scenarios.

III. SYSTEM DESIGN

We present an overview of CMFUZZ's design in Figure 1. CMFUZZ starts with a set of inputs comprising the protocol under test and its configurations, defined through various sources such as command-line interface (CLI) options and configuration files. The objective of CMFUZZ is to systematically extract and organize these configurations to enable efficient and diverse parallel fuzzing. The framework outputs a set of cohesive configuration groups, which are then distributed to different parallel fuzzing instances.

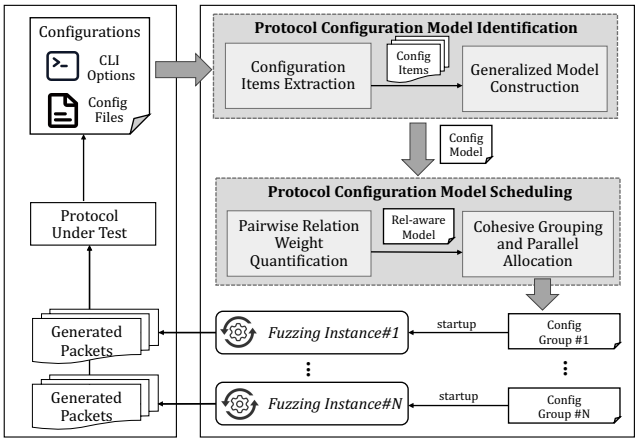


Fig. 1: CMFUZZ System Overview. It mainly consists of two components: (i) a configuration model identification module for extracting configuration items and constructing generalized model; (ii) a configuration model scheduling module for distributing configurations across parallel fuzzing instances.

CMFUZZ introduces a configuration model to explore a broader range of execution paths, supplementing the traditional data and state models used in existing protocol fuzzers. Starting with various protocol configuration formats, CMFUZZ first extracts a comprehensive list of configuration items to capture all adjustable parameters. These items are then organized into a generalized configuration

model, consisting of entities that represent each configuration item. Each entity includes the item's name, type, mutability flag, and a set of typical mutation values. With this configuration model in place, CMFUZZ then quantifies the pairwise relation weights between configuration entities by assessing their impact on startup coverage. Using these weights, CMFUZZ finally employs a cohesive grouping and allocation strategy, grouping interdependent entities together and assigning them to the same parallel fuzzing instance. This relation-aware allocation ensures that each instance explores a distinct and meaningful subset of configurations, thereby enhancing overall coverage and effectiveness in parallel protocol fuzzing.

A. Protocol Configuration Model Identification

1) Configuration Items Extraction: IoT protocol configurations are primarily found in two formats: command-line interface (CLI) options and configuration files. CLI options often follow predictable patterns, such as `--option=value` or `-flag`. To identify and extract these options, CMFUZZ uses a pattern-matching parser. For configuration files, formats vary widely, including key-value pairs (e.g., .ini files), structured hierarchies (e.g., JSON, XML), or unstandardized formats. CMFUZZ applies format-specific static analysis to handle each file structure: (i) For key-value formats, it directly parses each line to extract keys and values. (ii) For hierarchical formats, it recursively parses the structure to retrieve keys and default values based on the file's nested organization. (iii) For custom formats, CMFUZZ uses heuristics and configurable parsing rules to identify adjustable parameters based on keywords and contextual clues.

Algorithm 1: Configuration Items Extraction

Input: $\mathcal{C}_{options}$: CLI Options Configurations
Input: $\mathcal{C}_{files}$: Configuration Files
Output: Set_{CI}: The set of configuration items

1 **Algorithm**
2 $Set_{CI} \leftarrow \emptyset$
3 $Options \leftarrow$ GETCLIOPTIONS($\mathcal{C}_{options}$)
4 $Files \leftarrow$ GETCONFIGFILES($\mathcal{C}_{files}$)
5 `Extracion(` $Options, Files$)
6 **Procedure** `Extracion(` $Set_{Options}, Set_{Files}$)
7 **if** $Set_{Options} \neg \emptyset$ **then**
8 **for** $\mathcal{O}_\mathcal{C} \in Set_{Options}$ **do**
9 $Item \leftarrow$ EXTRACTCLIOPTIONS($\mathcal{O}_\mathcal{C}$)
10 $Set_{CI} \leftarrow Set_{CI} \cup Item$
11 **if** $Set_{Files} \neg \emptyset$ **then**
12 **for** $\mathcal{F}_\mathcal{C} \in Set_{Files}$ **do**
13 $Format \leftarrow$ DETECTFILEFORMAT($\mathcal{F}_\mathcal{C}$)
14 **switch** $Format$ **do**
15 **case** *KeyValue* **do**
16 $Set_{FI} \leftarrow$ EXTRACTKEYVALUE($\mathcal{F}_\mathcal{C}$)
17 **case** *Hierarchical* **do**
18 $Set_{FI} \leftarrow$ EXTRACTHIERARCHICAL($\mathcal{F}_\mathcal{C}$)
19 **otherwise do**
20 $Set_{FI} \leftarrow$ EXTRACTCUSTOMFORMAT($\mathcal{F}_\mathcal{C}$)
21 $Set_{CI} \leftarrow Set_{CI} \cup Set_{FI}$
22 **return**

Algorithm 1 illustrates the detailed workflow of CMFUZZ for extracting configuration items from various formats of IoT protocol configurations. The algorithm begins with two primary inputs: CLI Options Configurations and Configuration Files. First, we initialize an empty set Set_{CI} to store all configuration items extracted from these sources (line 2). Next, we retrieve CLI options and configuration files as structured data from the inputs (lines 3–4). Following this, we extract detailed configuration items according to their specific formats using the `Extraction` procedure. For each CLI option in $Set_{Options}$, we extract relevant details, including the configuration name, value and ranges, and add each item to Set_{CI} (lines 8–10). For each file in $Files$, we determine the format (line 13) and use format-specific extraction functions to parse the file's structure and content accordingly (lines 15–20). Items extracted from each file are then added to Set_{CI}, creating a consolidated set of configuration items for model construction (line 21).

2) Generalized Model Construction: Following the extraction of individual configuration items, CMFUZZ organizes these items into a generalized configuration model, which serves as the core structure within our framework.

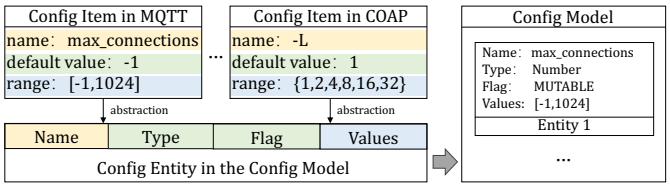


Fig. 2: Entities in the config model derived from the config item

As shown in Figure 2, this model is composed of 4-tuple entities, with each entity encapsulating essential attributes corresponding to a configuration item extracted from the protocol. The $Name$ attribute is inherited directly from the configuration item. The $Type$ attribute is inferred from the item's value patterns, for instance, numeric values are labeled as *Number*, boolean-like values as *Boolean*, and values resembling file paths or URLs as *String*. The $Flag$ attribute indicates whether a value is likely to change during typical protocol operations. Static values, such as paths or system directories, are marked as *IMMUTABLE*, while adjustable values, like numeric ranges or mode settings, are marked as *MUTABLE*. The $Values$ attribute, representing the typical set of values for each configuration, is derived from the item's defined ranges or common values using heuristic-based rules. This standardized configuration model provides a structured foundation for subsequent processes.

B. Protocol Configuration Model Scheduling

1) Pairwise Relation Weight Quantification: To leverage interdependencies among configurations and distribute the configuration model across parallel fuzzing instances, CMFUZZ enhances the original configuration model into a relation-aware configuration model, as illustrated in Figure 3. This transformation involves constructing a weighted relation graph, where each configuration entity is represented as a node, and edges with quantified weights connect pairs of entities. These weights reflect the strength of relationships between entities.

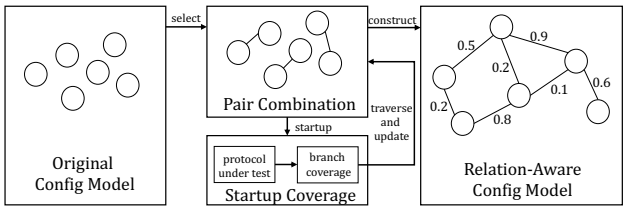


Fig. 3: Enhance an original config model into a relation-aware config model. CMFuzz achieves this by calculating the startup coverage.

When quantifying relations between configuration entities, CMFUZZ utilizes code coverage as a primary metric. Coverage serves as an effective indicator of configuration dependencies, as configurations

with synergistic relations often unlock new execution paths when used together. Conversely, configurations with conflicting relations, such as an encryption mode incompatible with a compression setting, may cause startup failures, resulting in zero code coverage. To improve efficiency, CMFuzz uses startup coverage as a lightweight proxy for overall code coverage. Because configurations are loaded and initialized during the startup phase, initial coverage provides a strong indication of the configuration's impact on later stages of execution.

By analyzing the startup coverage for each pair of configuration entities, CMFuzz then calculates the relation weight between them. Since each configuration entity may contain multiple typical values, CMFuzz explores all possible value combinations for each pair of entities and records the coverage achieved for each combination. To capture the peak interaction effect, CMFuzz selects the highest coverage across all combinations as the final relation weight between the two entities. This strategy ensures that the weight reflects the strongest potential interaction between entities, capturing critical synergies that might only emerge under specific values. Specifically, if the coverage for a pair of entities is zero across all combinations, CMFuzz does not create an edge between these entities. Finally, to ensure consistency and comparability across all relation weights, CMFuzz normalizes the calculated weights, scaling them to a standard range of [0, 1]. This normalization facilitates the grouping and clustering of configuration entities in subsequent phases.

2) Cohesive Grouping and Parallel Allocation: With the Relation-Aware Configuration Model constructed, CMFuzz proceeds to divide the graph into cohesive groups of configuration entities and allocate each group to a distinct parallel fuzzing instance.

Algorithm 2: Allocation Algorithm

Input: $RACModel$: Relation-Aware Configuration Model
Input: $\mathcal{N}_f$: The number of fuzzing instances
Output: $Groups$: The array of configuration groups allocated to each instance

1 **Algorithm**
2 $Groups \leftarrow \emptyset$, $GroupCnt \leftarrow 0$
3 $Set_{Edges} \leftarrow$ SortByWeight($RACModel$)
4 **for** $\mathcal{E}_s \in Set_{Edges}$ **do**
5 GroupNextEdge($\mathcal{E}_s$,$\mathcal{N}_f$)
6 **return** $Groups$
7 **Procedure** GroupNextEdge($\mathcal{E}$,$\mathcal{N}$)
8 $\mathcal{C}_1 \leftarrow \mathcal{E}.C_1$, $\mathcal{C}_2 \leftarrow \mathcal{E}.C_2$
9 **if** $\neg$IsSet($\mathcal{C}_1$) $\wedge$ $\neg$IsSet($\mathcal{C}_2$) **then**
10 **if** $GroupCnt < \mathcal{N}$ **then**
11 $NewGroup \leftarrow \{\mathcal{C}_1, \mathcal{C}_2\}$
12 $Groups \leftarrow Groups \cup \{NewGroup\}$
13 $GroupCnt \leftarrow GroupCnt + 1$
14 **else**
15 **for** $\mathcal{C} \in \{\mathcal{C}_1, \mathcal{C}_2\}$ **do**
16 $BestGroup \leftarrow$ FindBest($\mathcal{C}, Groups$)
17 $BestGroup \leftarrow BestGroup \cup \mathcal{C}$
18 **else if** IsSet($\mathcal{C}_1$) $\oplus$ IsSet($\mathcal{C}_2$) **then**
19 $CurGroup \leftarrow$ IsSet($\mathcal{C}_1$) ? Find($\mathcal{C}_1$) : Find($\mathcal{C}_2$)
20 $CurGroup \leftarrow CurGroup \cup$ (IsSet($\mathcal{C}_1$) ? $\mathcal{C}_2$: $\mathcal{C}_1$)
21 **return**

Algorithm 2 illustrates our allocation strategy, which aims to maximize the relation weights within groups and minimize the relation weights between groups. The algorithm starts with two inputs: the Relation-Aware Configuration Model $RACModel$ and the number of fuzzing instances N_f. First, we initialize an empty set $Groups$ to store the divided configuration groups and a counter $GroupCnt$ to track the number of groups created (line 2). Next, we sort all edges in $RACModel$ by weight in descending order, storing the sorted edges in Set_{Edges} (line 3). This sorting prioritizes edges with higher weights, promoting the formation of tightly connected groups by processing the strongest relationships first. For each edge in Set_{Edges}, we call the `GroupNextEdge` procedure, which determines how the two connected configuration entities (nodes) should be grouped (lines 4-5). In this procedure, if neither of the entities is assigned to a group (line 9), we verify whether the current number of groups $GroupCnt$ has reached the target number of instances $\mathcal{N}$ (line 10). If not, we create a new group with these two entities, add it to $Groups$, and increment $GroupCnt$ by one (lines 11-13). Once the maximum number of groups $\mathcal{N}$ has been reached, each new entity is placed into an existing group that best preserves its relations using the FindBest function (lines 15-17). Alternatively, if one entity is already part of a group, we assign another unassigned entity to the same group, thereby preserving their connection (lines 18-20).

Specifically, to identify the most appropriate existing group for an entity, the FindBest function computes a suitability score for each candidate group G_i based on the strength of the entity's connections to members of the group. The score is defined by:

$$Score(G_i, \mathcal{C}) = \frac{\left(\sum_{\mathcal{C}' \in G_i} w(\mathcal{C}, \mathcal{C}')\right)^2}{|G_i|}$$

where $w(\mathcal{C}, \mathcal{C}')$ represents the relation weight between the new entity $\mathcal{C}$ and each entity $\mathcal{C}'$ in group G_i, and $|G_i|$ denotes the number of entities in group G_i. In this formula, the numerator represents the strength of the relationship between the new node and existing nodes in G_i, while the denominator $|G_i|$ accounts for the balance in the number of entities across groups. The numerator is squared to amplify the effect of stronger connections.

Once cohesive groups have been formed, each group is assigned to a separate parallel fuzzing instance. Each instance then reassembles the configuration entities within its assigned group back into runtime-ready forms, such as configuration files or CLI options. During execution, each instance evaluates the $Flag$ attribute of each configuration entity to determine if the configuration value should be mutated. If mutation is required, it is applied based on the $Values$ attribute, which guides how the mutation should be performed. Mutations are introduced adaptively and are only applied if the current instance's coverage has reached saturation, meaning coverage has not increased over a set duration. Additionally, to prevent cross-contamination between instances, each fuzzing instance operates in an isolated network namespace. Through this structured parallel approach, each fuzzing instance is empowered to fully explore its assigned configuration subset, maximizing the chances of uncovering configuration-specific vulnerabilities.

IV. Evaluation

We have implemented CMFuzz on the state-of-the-art parallel protocol fuzzer Peach [8]. In the *Configuration Model Identification* module, we employ the Python Configuration File Parser [10] along with other specific libraries (such as `json`, `yaml`, `pyparsing`, and others) to convert configuration files into configuration items. For CLI options, we use Python Regular Expression Operations [11] to flexibly identify configuration items. In the *Configuration Model Scheduling* module, we leverage the `networkx` library to construct a weighted graph, where configuration entities serve as nodes and relations as weighted edges. For parallel fuzzing, we use Linux Network Namespaces (netns) via the `ip` networking tool to iso-

late network environments. Additionally, we use Clang to insert `trace-pc-guard` instrumentation, a feature in LLVM SanitizerCoverage [12], to collect branch coverage for pairwise relation weight quantification and parallel execution. For evaluation, we answer the following two research questions:

- **RQ1** Is CMFuzz more efficient than traditional parallel protocol fuzzers when applied to IoT protocols?
- **RQ2** Is CMFuzz effective in exposing previously unknown vulnerabilities in real-world IoT protocols?

A. Experiment Setup

Subjects. We selected six widely used IoT protocols: MQTT [13], CoAP [7], DDS [14], DTLS [15], AMQP [16] and DNS [17]. We use their corresponding popular open-source implementations as the test subjects, including Mosquitto [18], libcoap [19], CycloneDDS [20], OpenSSL [21], Qpid [22] and Dnsmasq [23].

Metrics. We employed three metrics for our evaluation: (i) branch coverage achieved, (ii) speedup in reaching the same coverage level as the compared fuzzers, and (iii) the number of bugs detected. The first metric is commonly used to measure the effectiveness of fuzzers, the second metric assesses the speed of parallel fuzzing, and the third metric indicates vulnerability detection capabilities.

Experiment Settings. Since the fuzzing performance fluctuates to a certain degree due to the inherent randomness, we ran each fuzzing tool on each selected project with a 24-hour time budget and repeated each 24-hour experiment five times to establish statistical significance of results [24]. For fairness, we use the same Pit files that specify the data and state models for each protocol. The experiment is executed in a Docker container with 4 CPU cores and 8 GB of RAM.

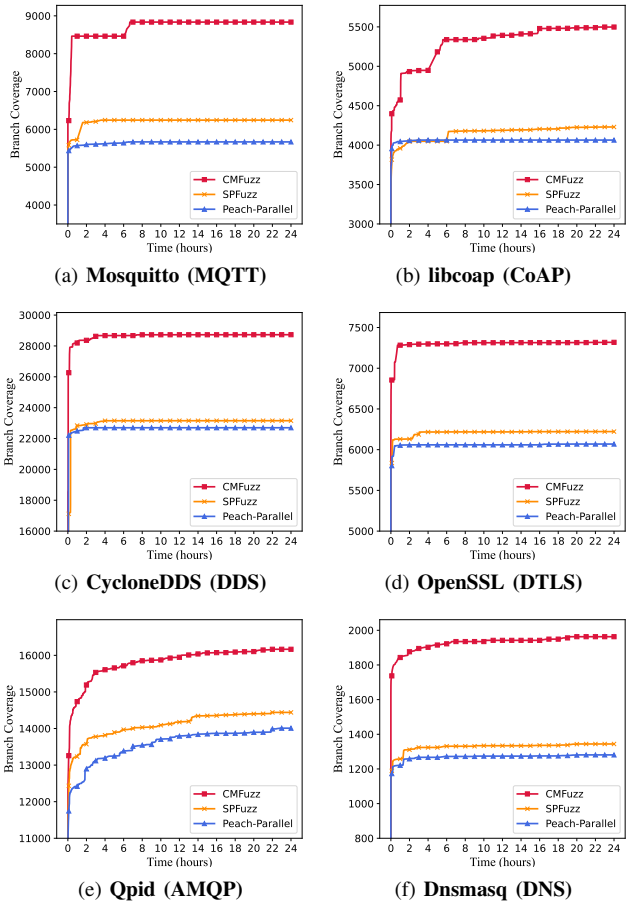
B. Coverage Analysis

To assess the efficiency of CMFuzz, we compared it to the state-of-the-art parallel-mode protocol fuzzers Peach [8] and SPFuzz [9]. Each fuzzer was run with four instances per project. We tracked and analyzed the number of branches covered by each fuzzer over 24 hours and calculated the speedup achieved by CMFuzz in reaching the same coverage level as the other fuzzers. Detailed results and improvements are summarized in Table I. The *Speedup* metric is defined as the baseline fuzzer's time to reach its final coverage divided by the time CMFuzz requires to achieve the same coverage. On average, CMFuzz improved branch coverage by 34.4% over Peach and 28.5% over SPFuzz across all tested protocol implementations. In terms of speedup, CMFuzz achieved an average acceleration of 3,544× compared to Peach and 2,746× compared to SPFuzz. These results reflect CMFuzz's significant enhancements in fuzzing efficiency for IoT protocols.

TABLE I: Average number of branches covered by each fuzzer running with four parallel instances over a 24-hour period.

Subject	CMFuzz	Peach	Improv	Speedup	SPFuzz	Improv	Speedup
Mosquitto	8,835	5,668	+55.9%	2,195×	6,244	+41.5%	30×
libcoap	5,498	4,064	+35.1%	450×	4,230	+30.0%	269×
cyclonedds	28,729	22,698	+26.6%	79×	23,155	+24.1%	146×
OpenSSL	7,318	6,069	+20.6%	8,820×	6,222	+17.6%	7,351×
Qpid	16,166	14,009	+15.4%	117×	14,440	+12.0%	41×
Dnsmasq	1,963	1,284	+52.9%	9,600×	1,348	+45.6%	8,640×
AVERAGE			+34.4%	3,544×		+28.5%	2,746×

Figure 4 illustrates that CMFuzz consistently achieves the highest branch coverage across all six projects, significantly outperforming the baseline tools Peach and SPFuzz. This underscores its effectiveness in exploring diverse execution paths. At the start of each session, all fuzzers exhibit an initial boost in coverage. CMFuzz achieves a considerable early lead because many of its extracted configuration items are loaded at startup, allowing rapid exploration of initial coverage. As time progresses, Peach and SPFuzz reach a saturation point, while CMFuzz continues to increase coverage by dynamically adjusting typical values from $Values$ field in configuration entities. This mechanism is particularly effective in protocols like CoAP, where varying block mode values allow exploration of alternative handling methods. Overall, in projects like Mosquitto and Dnsmasq, CMFuzz shows substantial improvement due to the extensive configurations in these protocols. For instance, Mosquitto supports varied QoS levels, authentication methods, and unique features like bridge connections, enabling CMFuzz to activate diverse paths by simulating different settings. In contrast, CMFuzz 's improvements on OpenSSL, CycloneDDS, and Qpid are modest due to limitations in flexibility. For instance, DTLS relies on fixed cryptographic settings, DDS's structured management restricts configuration diversity, and AMQP's predefined structure limits exploration.



(a) **Mosquitto (MQTT)** (b) **libcoap (CoAP)**

(c) **CycloneDDS (DDS)** (d) **OpenSSL (DTLS)**

(e) **Qpid (AMQP)** (f) **Dnsmasq (DNS)**

Fig. 4: Average number of branches achieved by CMFuzz and baseline parallel fuzzers within 24 hours for 5 repetitions on each IoT protocol implementation. All fuzzers are run with 4 instances.

C. Bug Detection Capability

In addition to the fuzzing efficiency improvement, CMFuzz has also uncovered 14 serious previously unknown vulnerabilities in these target protocols. These vulnerabilities pose significant security risks to devices operating with these protocols, potentially leading to device crashes, unauthorized access, or even complete system compromise. Table II summarizes the vulnerabilities discovered by CMFuzz. Each of them presents critical threats to IoT systems. Specifically, heap-use-after-free allows attackers to reuse freed memory, leading to

unpredictable behavior or remote code execution. SEGV is triggered by invalid memory accesses, and may expose sensitive information or allow code injection. Memory leaks, which cause memory to be exhausted over time, are particularly dangerous in resource-constrained IoT devices, potentially leading to system unresponsiveness and disrupted services. Allocation-size-too-big occurs when an abnormally large memory is allocated, potentially exhausting system resources or leading to denial-of-service attacks. Stack-buffer-overflow and heap-buffer-overflow involve writing data beyond allocated boundaries, potentially overwriting critical memory areas, leading to crashes or enabling attackers to execute malicious code.

TABLE II: Summary of vulnerabilities detected by CMFuzz

No.	Protocol	Vulnerability Type	Affected Function
1	MQTT	heap-use-after-free	Connection::newMessage
2		heap-use-after-free	neu_node_manager_get_addrs_all
3		heap-use-after-free	mqtt_packet_destroy
4		SEGV	loop_accepted
5		memory leaks	multiple functions
6	CoAP	SEGV	coap_clean_options
7		stack-buffer-overflow	CoapPDU::getOptionDelta
8		SEGV	coap_handle_request_put_block
9	AMQP	stack-buffer-overflow	pthread_create
10	DNS	stack-buffer-overflow	get16bits
11		heap-buffer-overflow	dns_question_parse, dns_request_parse
12		allocation-size-too-big	dns_request_parse
13		heap-buffer-overflow	printf_common
14		heap-buffer-overflow	config_parse

Case Study: Bug #8 in libcoap (CoAP). Figure 5 illustrates a SEGV vulnerability exposed by CMFuzz in the CoAP protocol. The bug occurs within `coap_handle_request_put_block` function, where `lg_srcv->body_data` is explicitly set to NULL (line 6) if `lg_srcv` is not found in the current session (line 3). However, if the block transfer process is incomplete or certain expected blocks fail to arrive, `lg_srcv->body_data` may remain NULL without being updated with valid data.

```
1  int coap_handle_request_put_block(...) {
2    ...
3    if (!lg_srcv) {
4      ...
5      // Initialize lg_srcv->body_data to NULL
6      lg_srcv->body_data = NULL;
7      LL_PREPEND(session->lg_srcv, lg_srcv);
8    }
9    ...
10 #if COAP_Q_BLOCK_SUPPORT
11   if (block_option == COAP_OPTION_Q_BLOCK1) {
12     if (check_all_blocks_in(...)) {
13       goto give_app_data;
14     }
15   }
16   ...
17 give_app_data:
18   ...
19   // Potential null dereference
20   pdu->body_data = lg_srcv->body_data->s;
21   ...
22 }
```

Fig. 5: The simplified code snippets related to the Bug#8.

When the non-default Q-Block1 configuration, which enables blockwise transfers, is activated (lines 10-11), the function checks if all blocks have been received (line 12). If they have, the code proceeds to the `give_app_data` label to reassemble and process the complete message body (line 13). At this label, `pdu->body_data` is assigned `lg_srcv->body_data->s` directly (line 20), assuming it is valid. However, if `lg_srcv->body_data` is still NULL, this assignment will cause a null pointer dereference, leading to a segmentation fault. This bug is critical as it potentially leads to server crashes and denial of service when handling blockwise transfers under specific configurations. Notably, it cannot be triggered under the default configuration, as it requires specific blockwise transfer settings to activate the vulnerable code path.

V. Related Work

Protocol Fuzzing. Numerous studies emphasize the extensive use of generation-based fuzzing for protocol implementation testing [8], [25]–[28]. Predominantly, existing research concentrates on enhancing algorithms that generate test packets. For example, Bleem [26] proposes a generation strategy focusing on packet sequences. Conversely, Logos [27] utilizes protocol server logs to conduct deep and efficient fuzzing within black-box environments. PAVFuzz [28] adopts a dynamic method to iteratively learn relationships between packet fields, aiming to improve packet generation. All these approaches focus on refining packet or sequence generation, and their effectiveness can be further bolstered by integrating CMFuzz. Our approach, CMFuzz, can autonomously determine and schedule configurations for parallel protocol fuzzing tasks and improve the effectiveness of the vulnerability detection. Thus, CMFuzz can be integrated with these existing methodologies to significantly boost fuzzing efficiency across multi-core computational resources.

Parallel Fuzzing. The enhancement of fuzzing efficiency is significantly driven by augmenting computational resources and leveraging parallel processing [29], [30]. Current research efforts concentrate on optimizing mutation-based fuzzers through information synchronization mechanisms and the strategic orchestration of fuzzing path scheduling. For example, tools such as AFL [31] and its protocol-oriented variant AFLNet [32] enable parallel fuzzing through synchronized seed sharing; AFLTeam [33] employs a dynamic task allocation model that refines task distribution, while PAFL [34] integrates a real-time synchronization framework for fuzzing status. Furthermore, SPFuzz [9] adopts a state-aware path-based method to enhance autonomous vehicle system scenarios. However, current fuzzing methodologies predominantly depend on standard configurations, neglecting the exploration of potential system behaviors under diverse configurations within IoT environments. This narrow focus restricts the ability to uncover deeper, configuration-specific protocol vulnerabilities that could emerge beyond default settings. Thus, recognizing the inherent multi-configuration complexity of IoT systems and integrating varied configurations into the fuzzing process becomes crucial for comprehensive vulnerability analysis.

VI. Conclusion

In this paper, we propose CMFuzz, a fuzzing framework that enhances parallel fuzzing for IoT protocols by dynamically identifying and scheduling configuration models. CMFuzz constructs a generalized configuration model by extracting protocol configuration items and grouping them based on inter-item relations. This approach enables parallel instances to explore diverse configuration combinations, significantly increasing coverage and the likelihood of discovering configuration-dependent bugs. Experimental results show that CMFuzz improves coverage by an average of 34.4% and 28.5% over Peach and SPFuzz, with significant speedups achieved for both baseline fuzzers. Furthermore, CMFuzz has uncovered 14 previously unknown vulnerabilities in real-world IoT protocols.

VII. Acknowledgment

This research is sponsored in part by the National Key Research and Development Project (No. 2022YFB3104000), and NSFC Program (No. U2441238, 62021002).

REFERENCES

[1] C. Paul, A. Ganesh, and C. Sunitha, "An overview of iot based smart homes," in *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, 2018, pp. 43–46.

[2] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of things (iot): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 474–10 498, 2021.

[3] W. Z. Khan, M. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, "Industrial internet of things: Recent advances, enabling technologies and open challenges," *Computers & electrical engineering*, vol. 81, p. 106522, 2020.

[4] A. Triantafyllou, P. Sarigiannidis, and T. D. Lagkas, "Network protocols, schemes, and mechanisms for internet of things (iot): Features, open challenges, and trends," *Wireless communications and mobile computing*, vol. 2018, no. 1, p. 5349894, 2018.

[5] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3453–3495, 2018.

[6] K. Tsiknas, D. Taketzis, K. Demertzis, and C. Skianis, "Cyber threats to industrial iot: a survey on attacks and countermeasures," *IoT*, vol. 2, no. 1, pp. 163–186, 2021.

[7] C. Bormann and K. Hartke, "Constrained Application Protocol (CoAP)," RFC 7252, June 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252

[8] M. Eddington, "Peach fuzzing platform." https://gitlab.com/gitlab-org/security-products/protocol-fuzzer-ce.

[9] J. Yu, Z. Luo, F. Xia, Y. Zhao, H. Shi, and Y. Jiang, "SPFuzz: Stateful path based parallel fuzzing for protocols in autonomous vehicles," 2024.

[10] Python, "Configuration file parser," https://docs.python.org/3/library/configparser.html.

[11] ——, "Regular expression operations," https://docs.python.org/3/library/re.html.

[12] LLVM, "Sanitizer coverage," https://clang.llvm.org/docs/SanitizerCoverage.html.

[13] A. Banks, E. Briggs, R. Coppen, and K. Borgendale, "MQTT Version 5.0," OASIS Standard, November 2018. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[14] O. M. Group, "Data Distribution Service (DDS) Version 1.4," OMG Specification, April 2015. [Online]. Available: https://www.omg.org/spec/DDS/1.4

[15] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, January 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6347

[16] OASIS, "Advanced message queuing protocol (amqp)," OASIS Standard, October 2012. [Online]. Available: https://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

[17] P. Mockapetris, "Domain Names - Implementation and Specification," RFC 1035, November 1987. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1035

[18] Eclipse, "Mosquitto," https://github.com/eclipse/mosquitto.

[19] O. Bergmann, "libcoap," https://github.com/obgm/libcoap.

[20] Eclipse, "Cyclonedds," https://github.com/eclipse-cyclonedds/cyclonedds.

[21] OpenSSL, "Openssl," https://www.openssl.org/.

[22] Apache, "Qpid," https://qpid.apache.org/components/cpp-broker/.

[23] S. Kelley, "Dnsmasq," http://www.thekelleys.org.uk/dnsmasq/doc.html.

[24] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. W. Hicks, "Evaluating fuzz testing," *2018 ACM SIGSAC CCS*.

[25] jtpereyda, "BooFuzz: Network protocol fuzzing for humans," https://github.com/jtpereyda/boofuzz.

[26] Z. Luo, J. Yu, F. Zuo, J. Liu, Y. Jiang, T. Chen, A. Roychoudhury, and J. Sun, "Bleem: packet sequence oriented fuzzing for protocol implementations," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4481–4498.

[27] F. Wu, Z. Luo, Y. Zhao, Q. Du, J. Yu, R. Peng, H. Shi, and Y. Jiang, "Logos: Log guided fuzzing for protocol implementations," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 1720–1732. [Online]. Available: https://doi.org/10.1145/3650212.3680394

[28] F. Zuo, Z. Luo, J. Yu, Z. Liu, and Y. Jiang, "PAVFuzz: State-sensitive fuzz testing of protocols in autonomous vehicles," *ACM/IEEE Design Automation Conference (DAC)*, 2021.

[29] W. Xu, S. Kashyap, C. Min, and T. Kim, "Designing new operating primitives to improve fuzzing performance," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2313–2328. [Online]. Available: https://doi.org/10.1145/3133956.3134046

[30] V.-T. Pham, M.-D. Nguyen, Q.-T. Ta, T. Murray, and B. I. Rubinstein, "Towards systematic and dynamic task allocation for collaborative parallel fuzzing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1337–1341.

[31] M. Zalewski, "american fuzzy lop - a security-oriented fuzzer," *https://github.com/google/AFL*, 2015.

[32] V.-T. Pham, M. Böhme, and A. Roychoudhury, "Aflnet: a greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 460–465.

[33] V. Pham, M. Nguyen, Q. Ta, T. Murray, and B. I. Rubinstein, "Towards systematic and dynamic task allocation for collaborative parallel fuzzing," in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering : NIER Track*, 2021.

[34] J. Liang, Y. Jiang, Y. Chen, M. Wang, C. Zhou, and J. Sun, "Pafl: extend fuzzing optimizations of single mode to industrial parallel mode," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 809–814. [Online]. Available: https://doi.org/10.1145/3236024.3275525

SeIM: In-Memory Acceleration for Approximate Nearest Neighbor Search

Chaoqiang Liu*, Dan Chen†, Yu Huang*, Wenjing Xiao‡, Haifeng Liu*, Yi Zhang*,
Huize Li†, Xiaofei Liao*, and Hai Jin*
*National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, China
†School of Computing, National University of Singapore, Singapore
‡ Guangxi Key Laboratory of Multimedia Communications and Network Technology, Guangxi University, Nanning, China
chqliu@hust.edu.cn, danchen@nus.edu.sg, yuh@hust.edu.cn, wenjingx@gxu.edu.cn,
{hfliu, yizh}@hust.edu.cn, huizeli@nus.edu.sg, {xfliao, hjin}@hust.edu.cn

***Abstract*—*Approximate nearest neighbor search* (ANNS) is crucial in many applications to find semantically similar matches for user queries. Especially with the development of *large language models* (LLMs), ANNS is becoming increasingly important in *retrieval-augmented generation* (RAG). An in-depth analysis of ANNS reveals that its diverse operations, from extensive memory access to intensive sorting, are key performance bottlenecks, imposing significant strain on both the memory system and computing resources. Based on these observations, we present SeIM, a hierarchical in-memory architecture to accelerate ANNS. SeIM is designed to accommodate the diverse operational characteristics of ANNS. Specifically, SeIM offloads highly parallel memory-bound operations to the memory bank level and introduces a unified *execution model* to reuse hardware units, requiring only lightweight modifications to standard DRAM architecture. Additionally, SeIM places compute-bound sorting operations, which require cross-unit data access, at the memory controller level and employs an *adaptive transmission filtering* technique to reduce unnecessary data transfers and processing during sorting. Our evaluation shows that SeIM achieves 268×, 22×, and 5× higher throughput, 306×, 59×, and 4× lower latency, and 3081×, 287×, and 2× higher power efficiency than state-of-the-art CPU-, GPU-, and ASIC-based ANNS solutions.**

***Index Terms*—processing-in-memory, DIMM, ANNS.**

I. Introduction

Approximate nearest neighbor search (ANNS) is a fundamental technique for retrieving vectors similar to a query vector in vector space, widely used in deep learning [1] and information retrieval [2] systems. Traditional *large language models* (LLMs), which rely on static training data, often struggle with real-time updates, leading to contextual misunderstandings and inaccurate responses. The *retrieval-augmented generation* (RAG) [3], [4] addresses this limitation by combining ANNS-enabled vector databases with LLMs, incorporating external knowledge to improve the performance of generative models. Additionally, recommender systems like YouTube [5] and search engines like Google [6] use ANNS to improve suggestions and search results.

This work is supported by the National Key Research and Development Program of China (No. 2023YFB4503400), the National Natural Science Foundation of China and Research Grants Council (NSFC/RGC) Joint Research Scheme (No. 62461160333), and the National Research Foundation, Singapore, under its Competitive Research Programme Award NRF-CRP23-2019-0003. Dan Chen is the corresponding author.

ANNS can be classified into four main categories: quantization-based [7], graph-based [8], hash-based [9], and tree-based [10]. Among these, quantization-based ANNS has emerged as the industry's first choice [11] for large-scale search due to their high performance and low memory footprint. In quantization-based ANNS, the search process begins by filtering out the cluster centroids closest to the query vector. Next, a distance table is constructed using the codebook, and the compressed vectors within the filtered clusters are scanned. The distance to the query vector can be quickly obtained for each compressed vector by looking up the distance table. This process mainly involves three types of operation: *vector computation*, *distance sorting*, and *lookup-based computation*.

We conduct an in-depth analysis of the search process using the mainstream ANNS library FAISS [12] and identify three key findings. First, search performance is constrained by multiple phases collectively, rather than a single phase. Therefore, optimizing search performance requires comprehensive improvements across the entire process, rather than focusing on a specific phase. Second, there are significant differences in the operational intensity of various operations: *vector computation* and *lookup-based computation* are highly memory-bound and susceptible to frequent data movement between memory and processing units, whereas *distance sorting* is primarily limited by computational capacity. Lastly, these three operations exhibit a mixed computation pattern: *vector computation* and *lookup-based computation* are highly parallelizable but differ in their processing for each dimension, *distance sorting* requires sequential aggregation of the results from all subtasks, complicating parallelization efforts.

Previous studies [13]–[15] have demonstrated that *processing-in-memory* (PIM) architectures hold significant potential for accelerating memory-bound applications. However, applying existing PIM solutions to quantization-based ANNS presents several challenges. First, the search process involves both memory-bound and compute-bound operations, posing challenges for existing PIM architectures to fully support the search process. Second, while current PIM accelerators [16], [17] optimized for *general matrix-vector multiplication*

(GEMV) excel at *vector computation*, they lack optimization for *lookup-based computation*, which demands distinct hardware designs and may introduce additional overhead. Therefore, new designs are needed to accelerate quantization-based ANNS.

In this work, we present SeIM, a high-performance and energy-efficient in-memory accelerator for quantization-based ANNS. SeIM's innovation lies in its hierarchical acceleration design, which addresses the distinct characteristics of search operations and delivers full functionality at a low cost. On the one hand, we integrate computation logic into memory to support both *vector computation* and *lookup-based computation*, requiring only lightweight modifications to DIMMs based on standard DRAM technology. On the other hand, we aggregate computation results in the memory controller and efficiently handle sorting operations. With higher power and area budgets compared to DIMMs, the memory controller supports the integration of more sufficient computational capabilities. To the best of our knowledge, SeIM is the first solution to leverage PIM technique for accelerating quantization-based ANNS.

In particular, SeIM first deploys custom processing units within each DRAM bank, specifically designed to support *vector computation* and *lookup-based computation*. By employing all-bank mode, it maximizes internal bandwidth utilization and parallelism. Additionally, SeIM introduces a unified *execution model* for these offloaded-to-memory operations, enabling logical reuse and minimizing hardware resource overhead. Second, SeIM incorporates hardware priority queues within the memory controller to efficiently handle *distance sorting*. SeIM further employs an *adaptive transmission filtering* technique to reduce memory transfers caused by invalid distance values. Finally, SeIM adopts a heterogeneous computing programming model, similar to CUDA/SYCL [18], to enhance development efficiency and improve the user experience.

We list our contributions as follows:

- We perform an in-depth characterization of quantization-based ANNS, identifying key performance bottlenecks and motivating the design of in-memory acceleration.
- We introduce SeIM, the *first* in-memory accelerator for quantization-based ANNS, which addresses the diverse requirements of various operations in the search process through a hierarchical PIM architecture.
- We further propose two key optimizations: (I) a unified *execution model* to reduce hardware resource overhead for in-memory computations, and (II) an *adaptive transmission filtering* technique to decrease unnecessary data transfers for sorting.
- Experimental results show that SeIM achieves significant improvements in both performance and energy efficiency compared to existing software and hardware solutions.

II. Background and Motivation

A. Quantization-based ANNS

***Product Quantization* (PQ)**. PQ is a widely employed technique for compressing high-dimensional vectors [7]. To mitigate compression errors, this algorithm begins by splitting

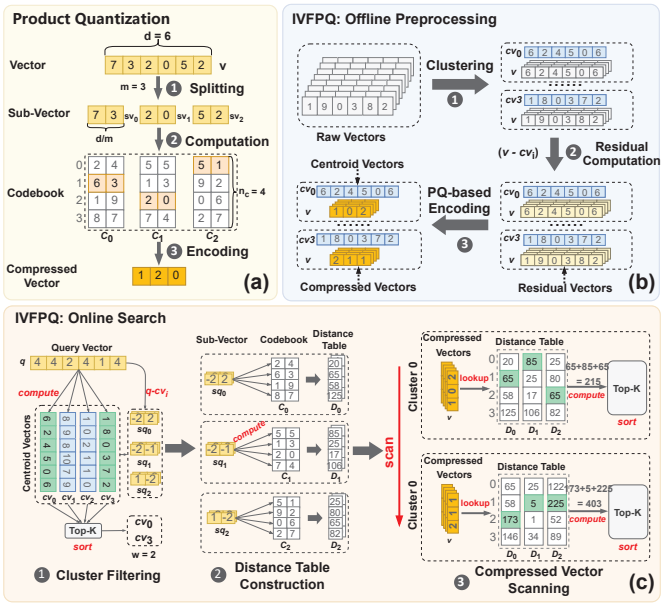


Fig. 1. An illustrative example of quantization-based ANNS (L2 distance)

the input vector v, of dimension d, into m distinct sub-vectors sv_i (where $0 \leq i < m$), each with a dimension of d/m (❶). For each sub-vector sv_i, the algorithm identifies the index of its nearest vector within the corresponding codebook C_i (❷). Consequently, the final vector v is encoded as a concatenation of m indices (❸). Fig. 1(a) illustrates an example of encoding a six-dimensional vector v with PQ. After encoding, v can be represented by three indices $\{1, 2, 0\}$.

PQ-based ANNS. Many popular ANNS libraries typically integrate the *inverted file index* (IVF) with PQ to reduce the number of compressed vectors that need to be scanned [7], a technique known as IVF-PQ. Fig. 1(b) illustrates the offline preprocessing process of IVF-PQ for database vectors. First, it clusters the database vectors (❶). Then, it computes the residuals of each vector relative to its respective cluster centroid (❷). Finally, it encodes all residual vectors utilizing product quantization (❸). After encoding is complete, the vector database will store the trained centroid vectors and compressed vectors.

Since preprocessing is an one-time operation, our primary focus is on the online search process. In ANNS, commonly used distance metrics between vectors include *euclidean distance* (L2) and *inner product distance* (IP). Fig. 1(c) shows the process of online search in IVF-PQ using the L2 distance. The initial phase is *cluster filtering* (CF). It selects the w nearest centroid vectors to the query vector q (❶). The subsequent phase is *distance table construction* (DTC). The algorithm computes the residual between the query vector q and the centroid vector of the selected cluster, then splits it into m sub-vectors and calculates the distance between each sub-vector and its corresponding codebook. Note that the computation process of IP distance is different from that of L2 distance (❷) [11]. The final phase is *compressed vector scanning* (CVS). It processes the compressed vectors within the selected w clusters. This is executed by using each dimension of a compression vector as an index to retrieve and accumulate

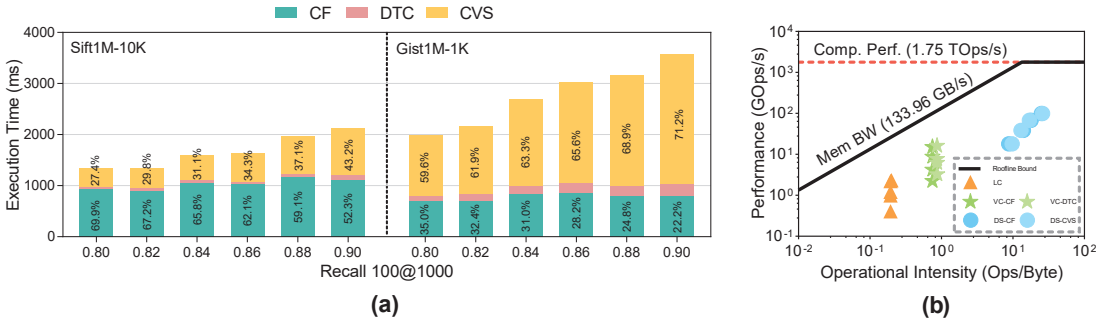


Fig. 2. (a) Search breakdown for different phases across datasets with varying recall (0.84, 0.86, 0.88, 0.90) and (b) roofline model analysis of ANNS, sweeping batch size

data from the corresponding distance table. After obtaining the distance values, the algorithm must also sort them to identify the top-k most similar vectors (❸).

B. Characterizations of Quantization-based ANNS

We characterize, FAISS [12], the popular ANNS library built by Facebook, on the CPU to identify performance bottlenecks and optimization opportunities. Experimental details are in §IV-A.

Performance Analysis. Fig. 2(a) delineates the execution time distribution across the three phases of the search process. Notably, *cluster filtering* and *compressed vector scanning* account for over 90% of the total time, indicating that optimizing these two phases is crucial to increase performance. We further analyze the key operations involved within these three phases. There are three key operations including: *vector computation* (VC), *distance sorting* (DS), and *lookup-based computation* (LC). During the *cluster filtering* phase, operations include *vector computation* and *distance sorting*; *Distance table construction* is primarily focused on *vector computation*; and *compressed vector scanning* consists of *lookup-based computation* and *distance sorting*.

Roofline Analysis. Fig. 2(b) depicts the roofline model data points for three key operations under various query conditions. The model indicates that *vector computation* and *lookup-based computation* are memory-bound, whereas *distance sorting* operation is compute-bound. Taking SIFT1M [19] as an example, the original vectors are 128-dimensional (4 bytes per element), compressed to 64-dimensional (1 byte per element), with a search for the top-100 nearest neighbors. For *vector computation*, this entails a 512-Byte DRAM read and 383 arithmetic operations (128 SUB, 128 MUL, 127 ADD), translating to an operational intensity of 0.75 operations per Byte. *Lookup-based computation* operation reads the 64-Byte compressed vector, retrieves 64 4-Byte distance values, and finally performs 63 additions with an intensity of 0.19 operations per Byte. *Distance sorting* operation, receiving a 4-Byte distance, involves updating a priority queue with up to 100 comparisons, reflecting an intensity of 25 operations per Byte.

Computation Pattern Analysis. *Vector computation* and *lookup-based computation* share similarities and key differences. Both lack dependencies between tasks, allowing them to be divided into independent subtasks for parallel processing. However, *vector computation* performs element-wise operations between two vectors followed by accumulation, while *lookup-based computation* uses vector dimensions as indices to retrieve and accumulate results from a lookup table. In contrast, *distance sorting* has a completely different pattern. It aggregates results from the other two operations to update a priority queue. Due to the dependencies in updates, distance sorting cannot be split into independent parallel subtasks.

C. Processing-in-memory Acceleration for Quantization-based ANNS

Why Choose PIM? PIM is a key technique proposed to accelerate the memory-bound operation. By integrating the compute unit in memory, PIM technique can significantly shorten the data transfer path between memory and processing units [13]–[15]. Quantization-based ANNS involves extensive memory access, making it suitable for PIM with high bandwidth and low latency capabilities. However, achieving this goal remains challenging. First, the characteristics of search operations vary, requiring more specific designs than existing PIM solutions. *Vector computation* and *lookup-based computation* are memory-bound and highly parallel, making them ideal for lower memory levels like the bank level. In contrast, *distance sorting* is compute-bound and requires aggregating results from the other two operations, making it better suited for higher memory levels. Second, while current PIM solutions [16], [17] for matrix-vector multiplications (GEMV) focus on *vector computation* at the memory bank level, they lack optimizations for *lookup-based computation*. Supporting *lookup-based computation* would require new hardware designs, potentially increasing overhead.

How to Design? To address these challenges, we propose a hierarchical PIM design. For *vector computation* and *lookup-based computation*, we deploy processing units at the memory bank level, maximizing the utilization of internal memory bandwidth. Additionally, we design a unified execution model and computation logic for these two operations, enabling resource sharing and minimizing hardware modifications, while maintaining compatibility with commercial DIMM architectures. For *distance sorting*, which requires aggregating results from the other two operations and involves computation-intensive sorting tasks, we extend the memory controller with dedicated hardware for sorting acceleration. Furthermore,

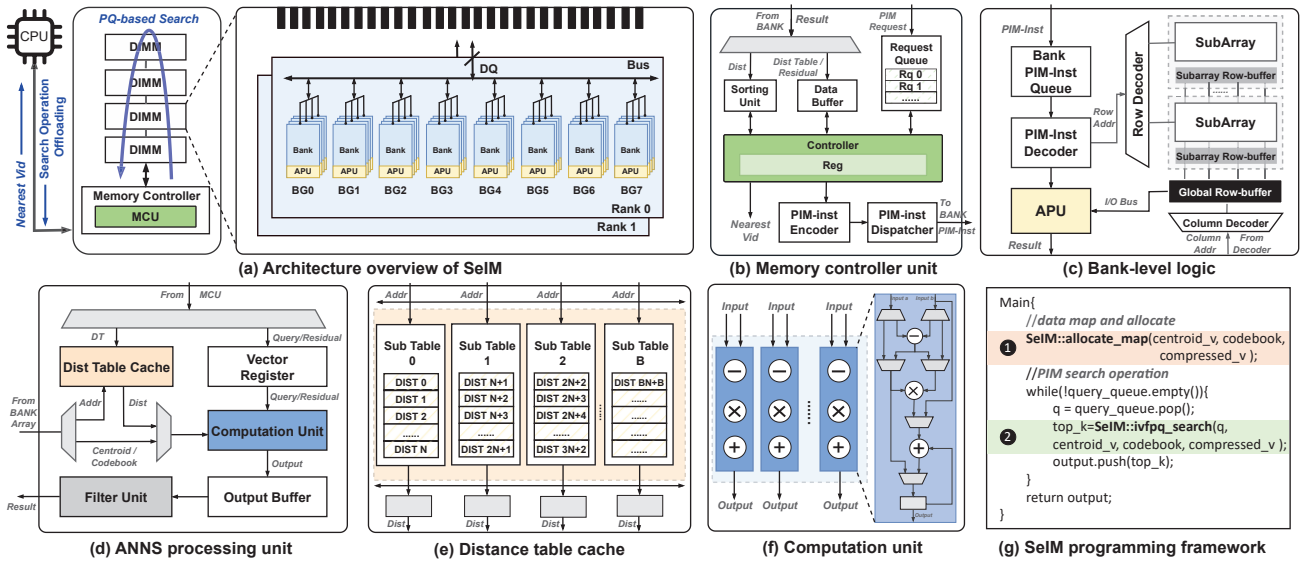


Fig. 3. The SeIM architecture

we propose an adaptive filtering mechanism to reduce the transmission and sorting of invalid results.

III. SeIM Architecture

In this section, we introduce SeIM, a DIMM-based hierarchical PIM architecture designed to accelerate quantization-based ANNS. Although SeIM is implemented using DIMM in this work, its design principles are equally applicable to HBM/HMC.

A. Architecture Overview

Fig. 3(a) depicts the overall architecture of SeIM. Unlike conventional memory systems, SeIM integrates a *memory controller unit* (MCU) within the memory controller, which manages the entire data flow and performs sorting operation. Additionally, components such as *ANNS processing unit* (APU) are deployed on each bank within the DIMMs to support diverse computing tasks. These APUs can simultaneously respond to computing instructions at the bank level, thereby exploiting DRAM's bank-level parallelism and significantly enhancing computational efficiency. Notably, SeIM maintains compatibility with conventional memory devices, as all original interfaces remain unchanged.

B. Hardware Design

***Memory Controller Unit* (MCU).** The detail of the MCU is illustrated in Fig. 3(b).The MCU comprises several components: a controller, a request queue, an encoder, a dispatcher, a data buffer, and a sorting unit. The primary function of the request queue is to temporarily store PIM requests from the host. The controller is responsible for activating either the sorting unit or the encoder based on the current progress. The encoder generates a PIM instruction for each PIM request, which is then transferred by the dispatcher to the corresponding banks within the DIMMs. To overcome C/A bandwidth limitation, SeIM utilizes the compression technique to encode all DDR commands, tags, and other relevant information in a single PIM instruction. The data buffer stores the distance table or residual vectors generated during the search process. The sorting unit is implemented as a hardware priority queue [20] based on the binary heap structure to handle all sorting tasks.

Bank-level Logic. As shown in Fig. 3(c), SeIM integrates the APU directly within each DRAM bank to enable efficient near-memory computation. In addition to the APU, SeIM incorporates an instruction queue and instruction decoder into each bank. These components are responsible for receiving and decoding PIM instructions from the memory controller. This design enables each bank to independently execute its assigned computational tasks in parallel.

***ANNS Processing Unit* (APU).** As depicted in Fig. 3(d), the core component of the APU is the computation unit. Fig. 3(f) details the design of the computation unit, which supports both *vector computation* and *lookup-based computation*. Additionally, for *vector computation*, the unit supports both IP distance and L2 distance metrics, which cover most search scenarios. The APU also includes a distance table cache for storing the distance table, a vector register for caching query and residual vectors, an output buffer for holding computation results, and a filter unit that supports the adaptive transmission filtering technique, as introduced in § III-D.

***Distance Table Cache* (DTC).** Based on the execution model described in §III-C, the APU simultaneously loads multiple compression vectors of the same dimension, which serve as addresses for lookup in the distance table. To enhance read bandwidth, the distance table cache is divided into multiple independently addressable tables, as illustrated in Fig. 3(e). In an ideal scenario, the input addresses will be distributed across different tables, allowing the distance table cache to read multiple distance values concurrently.

C. Data Layout and Execution Model

The tasks offloaded to memory (*vector computation* and *lookup-based computation*) share similar computation pattern. Therefore, we propose a unified execution model to reuse

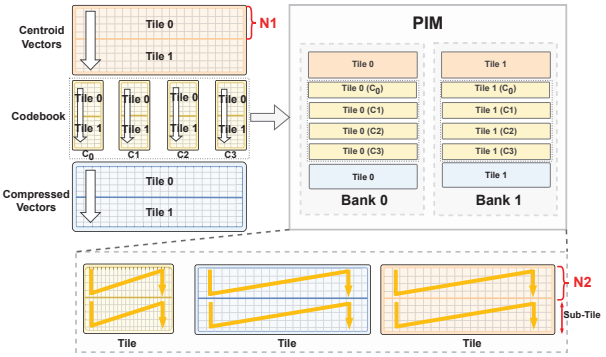


Fig. 4. Data layout and execution model

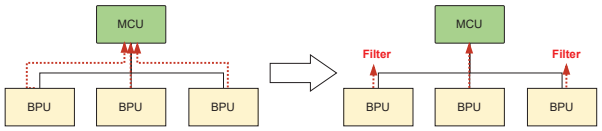


Fig. 5. Adaptive transmission filtering

hardware logic and maximize bank-level parallelism. Meanwhile, the data involved in the computations, such as the centroid vectors, codebook, and compressed vectors, all follow a consistent data layout scheme. The data layout scheme and execution model are shown in Fig. 4.

Across banks, we divide the three types of data into equally sized tiles along the row dimension, and each tile is assigned to a different bank. Each bank processes identical tasks with nearly the same amount of data, ensuring load balancing between processing units. During the search phase, memory operates in all-bank mode, where each bank processes its assigned tasks in parallel.

Within bank, for *lookup-based computation*, processing the compressed vector tiles in row-major order would require storing the entire distance table in the on-chip memory, which could result in significant area overhead. On the other hand, using column-major order would also necessitate a large buffer to store the intermediate results for all rows. To address this issue, we further subdivide the tile along the row dimension into smaller sub-tiles, with the size determined by the buffer capacity. Within each sub-tile, we store and process the compressed vectors in column-major order. To simplify hardware design, the same processing model is applied to *vector computation*.

One particular scenario is when using L2 distance for searching, where we need to compute the residuals between the query vector and each selected centroid vector. To reduce unnecessary row activations, we make a copy of the centroid tiles in each bank with an entirely row-major layout, which only consumes a few extra megabytes of memory.

D. Adaptive Transmission Filtering

The sorting unit must aggregate data from all banks, which may introduce new bottlenecks. We find that most of the distance values read into the memory controller are discarded and do not participate in the actual queue updates. To address this, we design a filtering mechanism, as shown in Fig. 5, which periodically transmits the maximum distance value from the priority queue to each APU and saves it. When the APU calculates new results, it compares them to the saved maximum distance value; if the new result exceeds this value, it is discarded directly without being transferred to the memory controller for sorting. This approach not only reduces the amount of data transmitted in memory but also saves sorting time overhead.

TABLE I
SEIM HARDWARE CONFIGURATION AND PARAMETERS

DRAM Specification
DDR5-4800 MHz, 4 Channels × 1 DIMM × 2 Ranks 64-entry RD/WR queue, FR-FCFS
DRAM Timing Parameters
tRCD=40, tCL=40, tRP=40, tBL=8 ,tCCD_S=8, tCCD_L=12 tRAS=76 , tFAW=32, tRC=116
SeIM Specialized Units
1 APUs per Bank, 1.06KB Cache/Buffer per APU 1MB Buffer per MCU

TABLE II
DATASET INFORMATION

Dataset	#Vector	#Dim	Distance Metric	Domain
Deep1M	10^6	96	L2	Image
Sift1M	10^6	128	L2	Text
Gist1M	10^6	960	L2	Image
Deep1B	10^9	96	L2	Image
Sift1B	10^9	128	L2	Text
TTI1B	10^9	128	IP	Image&Text

E. Programming Model and Execution Flow

SeIM adopts a heterogeneous programming model to enable PIM functionality. Fig. 3(g) illustrates the programming framework of SeIM. Before executing the search, SeIM allocates and maps data—including centroid vectors, codebooks, and compressed vectors—into corresponding memory regions, following the data layout strategy described in § III-C, while mapping indices to physical addresses. When a search operation is initiated, the host receives a batch of query requests. For each query, SeIM activates the PIM kernel to generate PIM requests that are then executed in memory. Upon completion of the search, the top-k neighbor IDs of all query requests in the batch are returned from the memory to the host through the DRAM interface.

IV. EVALUATION

A. Experimental Setup

SeIM Settings. We implement an internal cycle-accurate simulator based on Ramulator [21] for evaluating the performance of SeIM. The basic hardware configuration of SeIM is summarized in Table I. We synthesize the special hardware units using Synopsys Design Compiler [22] under a 32nm technology library at the frequency of 500MHz. The energy consumption, area, and access latency of all buffers in SeIM are estimated by CACTI [23].

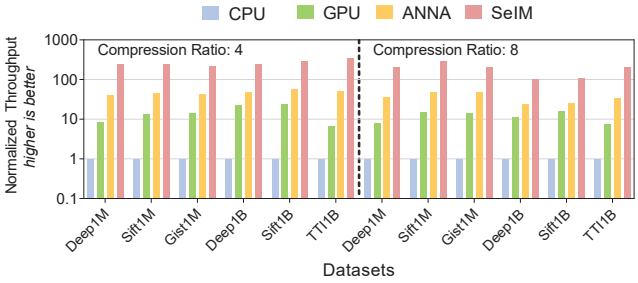


Fig. 6. Throughput comparison (recall 100@1000)

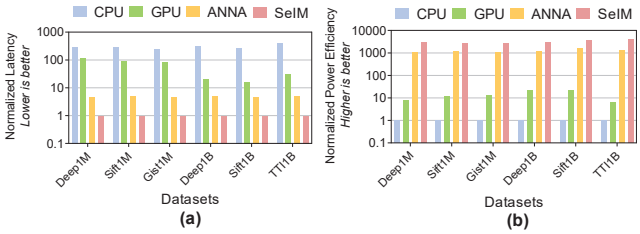


Fig. 7. (a) Latency and (b) power efficiency comparison

Datasets and Metrics. We pick four representative real-world datasets, Sift [19], Gist [19], Deep [24] , and TTI [24], with different data scales. The dataset details are shown in Table II. We compare the performance of the different solutions by measuring the *throughput* (QPS) and *latency* (ms) of the queries with the same *recall* ($R@K$), which refers to the fraction of the true R nearest neighbors included in the K search results. Additionally, we take *energy efficiency* (QPS/Watt) into account to evaluate the balance between performance and energy consumption.

Baselines. We compare SeIM to CPU, GPU, and ASIC baselines. For the CPU baseline, we choose a 14-core, dual-socket Xeon Gold 5117 CPU with 512GB of DDR4 memory, utilizing 56 threads to search in parallel during testing. The GPU baseline runs on an Nvidia V100 with 32GB of HBM2 memory. The performance and power consumption are measured by Intel RAPL and nvprof tools. We further compare SeIM with the state-of-the-art PQ-based ANNS accelerator ANNA [11]. Note that all methods are based on the most popular PQ-based ANNS library FAISS [12], and all parameters in the algorithm are identical.

B. Experimental Results

Throughput Comparison. Fig. 6 illustrates the throughput comparison of SeIM compared to CPU, GPU, and ANNA at different compression rates. Compared with CPU, GPU, and ANNA, SeIM achieves on average 188.07×, 16.99×, and 5.09× throughput improvement at compression rate of 8, and 268.76×, 22.92×, and 5.59× at compression rate of 4, respectively. The performance improvement of SeIM comes from several factors. (1) SeIM utilizes near-memory processing to fully utilize in-memory bandwidth and dramatically reduce data transfer distances. (2) SeIM customizes a highly parallelized near-memory processing unit for searching and pipelines of the data path. (3) SeIM introduces optimizations such as *adaptive Transmission Filtering*, which further reduces data transfer as well as processing.

TABLE III
SEIM HARDWARE OVERHEAD

SPU Hardware Overhead of One DIMM
Power (mW): 73.46 (CU), 320.71 (Buffer&Cache), 394.17 (Total)
Area (mm^2): 0.185 (CU), 0.262 (Buffer&Cache), 0.447 (Total)
MCU Hardware Overhead
Power (mW): 71.35 (SU), 2565.70 (Buffer), 2637.05 (Total)
Area (mm^2): 0.103 (SU), 2.098 (Buffer), 2.201 (Total)

Latency Comparison. We further evaluate the average query latency of SeIM against the baselines when the compression rate is 4. As shown in Fig. 7(a), SeIM reduces the query latency by 306.42×, 59.03×, and 4.85× compared to CPU, GPU, and ANNA. Moreover, SeIM can still process similarity searches in milliseconds for data at a billion scale, which indicates that SeIM is well suited for latency-constrained scenarios to meet strict SLA constraints.

Power and Area. Table III reports the area and power cost of SeIM. In total, SeIM adds $0.447mm^2$ for area and $0.39W$ for power consumption per DIMM. In comparison, a single commercial DIMM chip typically consumes about $10W$ of power and $3500mm^2$ of area, respectively, and the additional overhead introduced by SeIM is negligible. For the upper MCU, the area and power consumptions are $2.63W$ and $2.201mm^2$, respectively.

Power Efficiency Comparison. Fig. 7(b) provides the power efficiency comparison between SeIM and baselines. On average, SeIM is 3081.37×, 287.02×, and 2.49× more power efficient than CPU, GPU, and ANNA, respectively. This increased efficiency can be attributed to two main factors: first, minimizing the distance of data movement reduces energy consumption associated with data transfers; second, the highly parallel and customized computational architecture significantly accelerates search time.

Impact of SeIM Optimization. We compare the throughput of SeIM with and without *Adaptive Transmission Filtering*. On average, at a compression rate of 4, SeIM with *Adaptive Transmission Filtering* achieves 2.13× throughput compared to SeIM without *Adaptive Transmission Filtering*. Similarly, at a compression rate of 8, this leads to an additional speedups of 1.94×.

V. CONCLUSION

In this paper, we present SeIM, a high-performance and energy-efficient in-memory acceleration architecture for ANNS, designed to fully support the entire search process. We conduct a comprehensive analysis of mainstream ANNS tasks, identifying key performance bottlenecks and architectural requirements. Based on this, we propose a hierarchical PIM design that fully leverages internal memory bandwidth to accelerate memory-bound computations and uses controller-level hardware to accelerate compute-bound sorting. Extensive evaluations show that SeIM significantly outperforms state-of-the-art CPU-, GPU-, and ASIC-based solutions in both performance and energy efficiency.

REFERENCES

[1] Y. Cao, H. Qi, W. Zhou, J. Kato, K. Li, X. Liu, and J. Gui, "Binary hashing for approximate nearest neighbor search on big data: A survey," *IEEE Access*, vol. 6, pp. 2039–2054, 2017.

[2] M. Flickner, H. S. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: The QBIC system," *IEEE Computer*, vol. 28, no. 9, pp. 23–32, 1995.

[3] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proceedings of the 34th Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[4] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J. Lespiau, B. Damoc, A. Clark, D. de Las Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. W. Rae, E. Elsen, and L. Sifre, "Improving language models by retrieving from trillions of tokens," in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022, pp. 2206–2240.

[5] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*, 2016, pp. 191–198.

[6] Statista, "Worldwide visits to google.com." [Online]. Available: https://www.statista.com/statistics/268252/web-visitor-traffic-to-googlecom/

[7] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[8] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020.

[9] L. Gong, H. Wang, M. Ogihara, and J. Xu, "idec: Indexable distance estimating codes for approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 13, no. 9, pp. 1483–1497, 2020.

[10] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya, "Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces," *Proceedings of the VLDB Endowment*, vol. 11, no. 8, pp. 906–919, 2018.

[11] Y. Lee, H. Choi, S. Min, H. Lee, S. Beak, D. Jeong, J. W. Lee, and T. J. Ham, "ANNA: specialized architecture for approximate nearest neighbor search," in *Proceedings of the 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 169–183.

[12] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, vol. 7, no. 03, pp. 535–547, 2021.

[13] S. H. Lee, S. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, S. O, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for PIM based on commercial DRAM technology : Industrial product," in *Proceedings of the 48th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43–56.

[14] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. S. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 273–287.

[15] F. Devaux, "The true processing in memory accelerator," in *Proceedings of IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–24.

[16] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. N. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.

[17] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. H. Ahn, "Trim: Enhancing processor-memory interfaces with scalable tensor reduction in memory," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, pp. 268–281.

[18] K. Group, "Sycl overview." [Online]. Available: https://www.khronos.org/sycl/

[19] M. Aumueller, E. Bernhardsson, and A. Faitfull, "Benchmarking nearest neighbors." [Online]. Available: https://github.com/erikbern/ann-benchmarks

[20] R. Bhagwan and B. Lin, "Fast and scalable priority queue architecture for high-speed network switches," in *Proceedings of the IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 2, 2000, pp. 538–547.

[21] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[22] Sysnopsys, "Design compiler." [Online]. Available: https://www.synopsys.com/

[23] H. laboratories, "Cacti 6.0: A tool to model large caches." [Online]. Available: https://www.hpl.hp.com/techreports/2009/HPL-2009-85.pdf

[24] Y. Research, "Benchmarks for billion-scale similarity search." [Online]. Available: https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search

PHOENIX: Pauli-Based High-Level Optimization Engine for Instruction Execution on NISQ Devices

Zhaohui Yang*, Dawei Ding†, Chenghong Zhu‡, Jianxin Chen§, Yuan Xie*
*Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong
† Yau Mathematical Sciences Center, Tsinghua University, Beijing 100084, China
‡ The Hong Kong University of Science and Technology (Guangzhou), Guangdong 511453, China
§Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

***Abstract*—Variational quantum algorithms (VQA) based on Hamiltonian simulation represent a specialized class of quantum programs well-suited for near-term quantum computing applications due to its modest resource requirements in terms of qubits and circuit depth. Unlike the conventional single-qubit (1Q) and two-qubit (2Q) gate sequence representation, Hamiltonian simulation programs are essentially composed of disciplined subroutines known as Pauli exponentiations (Pauli strings with coefficients) that are variably arranged. To capitalize on these distinct program features, this study introduces PHOENIX, a highly effective compilation framework that primarily operates at the high-level Pauli-based intermediate representation (IR) for generic Hamiltonian simulation programs. PHOENIX exploits global program optimization opportunities to the greatest extent, compared to existing SOTA methods despite some of them also utilizing similar IRs. PHOENIX employs the binary symplectic form (BSF) to formally describe Pauli strings and reformulates IR synthesis as reducing the column weights of BSF by appropriate Clifford transformations. It comes with a heuristic BSF simplification algorithm that searches for the most appropriate 2Q Clifford operators in sequence to maximally simplify the BSF at each step, until the BSF can be directly synthesized by basic 1Q and 2Q gates. PHOENIX further performs a global ordering strategy in a Tetris-like fashion for these simplified IR groups, carefully balancing optimization opportunities for gate cancellation, minimizing circuit depth, and managing qubit routing overhead. Experimental results demonstrate that PHOENIX outperforms SOTA VQA compilers across diverse program categories, backend ISAs, and hardware topologies.**

I. INTRODUCTION

Quantum computing offers the potential to revolutionize various fields, driving decades of efforts to develop the required physical hardware. For instance, quantum algorithms can achieve exponential speedups in tasks such as integer factorization [1], solving linear equations [2], and quantum system simulation [3]. In the noisy intermediate-scale quantum (NISQ) era where we have access to dozens or hundreds of qubits susceptible to noise (e.g., qubit decoherence, gate imperfections) [4], variational quantum algorithms (VQA) is a leading class of algorithms proposed to achieve the quantum advantage (e.g., VQE for chemistry and condensed-matter simulation [5], QAOA for combinatorial optimization [6]) due to its modest resource requirements in terms of qubit number and circuit depth as well as its noise-resilience. [7].

The construction of a VQA ansatz circuit is to simulate (approximate) a desired unitary evolution under the system Hamiltonian H and the evolution duration t, through Trotterizing [7] the evolution $U(t)$ given H represented by a linear combination of Pauli strings:

$$U(t) = e^{-iHt} \simeq \left(S_k\left(\tau\right)\right)^r, \tau = \frac{t}{r}, \tag{1}$$

$$H = \sum\nolimits_{j=1}^{L} h_j P_j = \sum\nolimits_{j=1}^{L} h_j \sigma_0^{(j)} \otimes \cdots \otimes \sigma_{n-1}^{(j)}, \tag{2}$$

§Corresponding author. Email: chenjianxin@tsinghua.edu.cn.

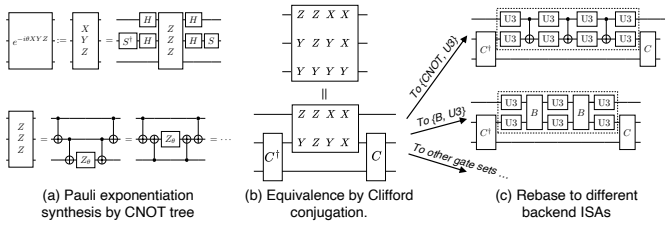


Fig. 1. Conventional Pauli exponentiation synthesis v.s. Simultaneous simplification by Clifford conjugation. (a) Pauli exponentiations are synthesized by gate set $\{H, S, S^\dagger, Z(\theta), \mathrm{CNOT}\}$, through variable CNOT-tree unrolling schemes. (b) Multiple 3Q Pauli exponentiations can be simultaneously simplified into 2Q Pauli exponentiations, through a 2Q Clifford conjugation, where $C = (H \otimes S)\,\mathrm{CNOT}\,(H \otimes S^\dagger)$ in this example. (c) The simplified Pauli exponentiations can be rebased to versatile quantum gate sets, such as CNOT-based and B-based ISAs, with significantly reduced 2Q gate count.

where k and r indicate the Trotter order and time step, respectively. Finer-grained Trotterization results in lower approximation errors. For example, the 1st-order and 2nd-order Trotterization are given by

$$S_1 = \prod\nolimits_{j=1}^{L} e^{-ih_j \tau P_j},\ S_2 = \prod\nolimits_{j=1}^{L} e^{-ih_j \frac{\tau}{2} P_j} \prod\nolimits_{j=L}^{1} e^{-ih_j \frac{\tau}{2} P_j},$$

respectively. Following Trotterization, each term S_k is expressed as a product of individual Pauli exponentiations, which can be easily synthesized using basic 1Q and 2Q gates. The arrangement of these Pauli exponentiations within each Trotter step can be freely chosen without affecting the upper bound of Trotterization (approximation) error [7], which is usually not comparable to physical noise. However, different arrangements offer varying opportunities for optimizing quantum circuits, thereby dominating the accuracy of Hamiltonian simulations on noisy hardware. Consequently, the primary challenge of compiling VQA ansatz circuits lies in synthesizing these Pauli exponentiations from a global perspective, dubbed *Pauli-based intermediate representations (IR)* throughout this paper.

Conventionally, a Pauli exponentiation is synthesized into a 1Q rotation $Z(\theta)$ sandwiched by a pair of symmetric CNOT trees, subsequently conjugated by some H and S gates, as exemplified in Fig. 1 (a). Existing state-of-the-art (SOTA) compilers primarily exploit gate cancellation opportunities exposed by the already synthesized subcircuits, whether by means of the abstract ZX diagram [8], [9], [10] representation or the variants of CNOT trees [11], [12]. Despite utilizing the Pauli-based IR, the optimization process they formulate is limited to subcircuits and local IR patterns. Furthermore, it assumes the conventional CNOT-based quantum instruction set architecture (ISA). We instead present that a set of Pauli exponentiations *can be simultaneously simplified* through appropriate Clifford

transformations. For example, Fig. 1 (b) shows that the list of weight-3 Pauli strings $[ZYY;\ ZZY;\ XYY;\ XXY]$ can be simplified into a weight-2 Pauli string list through the conjugation of a 2Q Clifford operator $C = (H \otimes S)\,\mathrm{CNOT}\,(H \otimes S^\dagger)$. This approach unlocks greater optimization opportunities *entirely at the level of Pauli-based IR* while remaining agnostic to the quantum ISA being employed.

In this work, we propose PHOENIX *(Pauli-based High-level Optimization ENgine for Instruction eXecution)*—a highly effective compilation framework for generic Hamiltonian simulation programs on near-term quantum devices. PHOENIX follows the

"IR grouping → group-wise simplification → IR group ordering"

pipeline to compile real-world VQA programs into basic quantum gates. It is ISA-independent, routing-aware, and tailored to generic VQA programs such as molecular simulation involving heterogeneous-weight Pauli strings and QAOA including only weight-2 Pauli strings. Differing from existing SOTA methods, PHOENIX utilizes an alternative formal description for Pauli-based IR and integrates heuristic optimization strategies to achieve global optimization primarily at the high-level semantic layer. We evaluate PHOENIX across diverse VQA programs, backend ISAs, and hardware topologies, demonstrating its superior performance over existing SOTA compilers. Our main contributions are as follows:

1) We utilize the *binary symplectic form* (BSF) as the formal description of Pauli-based IR and reformulate the IR synthesis process as simultaneous simplification on BSF by appropriate Clifford conjugations that simultaneously reduce the BSF's column weights, enabling global optimization to the largest extent while operating at the high-level IR.
2) We propose a heuristic *BSF simplification* algorithm as the core optimization pass in PHOENIX. It iteratively searches for the most appropriate 2Q Clifford operators to quickly lower the weight of the BSF, until the BSF can be directly synthesized by basic 1Q and 2Q gates.
3) We quantify the circuit depth overhead induced by assembling (ordering) simplified IR groups. A uniform cost function is devised to comprehensively account for circuit depth overhead, gate cancellation, and qubit routing overhead. Under the guidance of that, PHOENIX integrates a *Tetris-like IR group ordering* procedure to reliably achieve a good ordering scheme.

Overall, PHOENIX outperforms the best-known compilers (e.g., TKET [13], PAULIHEDRAL [11], TETRIS [12]), achieving significant reductions in gate count and circuit depth. For example, for logical-level compilation, PHOENIX results in 80.47% reduction in CNOT gate count and 82.72% reduction in 2Q circuit depth on average, compared to the original logical circuits. For hardware-aware compilation with heavy-hex topology, PHOENIX reduces by 36.17% (22.62%) in CNOT gate count and 43.85% (28.12%) in 2Q circuit depth on average, compared to PAULIHEDRAL (TETRIS). The reduction effect in 2Q gate count and circuit depth becomes even more impressive when targeting the newly introduced SU(4) ISA (the representative continuous ISA containing all 2Q gates) [14].

II. RELATED WORKS

Based on ZX diagram representations of Pauli gadgets. Compilers such as TKET [13], [8], PAULIOPT [9], and PCOAST [10] resynthesize quantum circuits into its ZX diagram representation. ZX diagrams are used to visually represent and simplify the commutation relations and phase interactions between Pauli operators, facilitating the reduction of circuit depth and gate count through algebraic and graphical transformations. Although it can efficiently resynthesize Pauli gadgets, the commutation rules it leverages occur locally, and it is hard to operate in a hardware-aware manner.

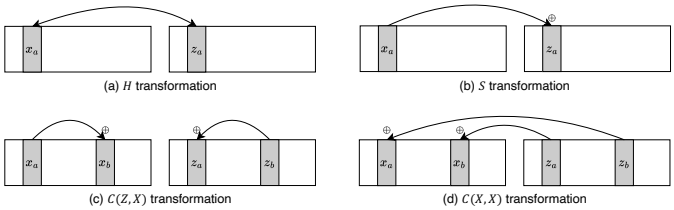


Fig. 2. Examples of Clifford transformations on BSF. Columns a and b, corresponding to qubit a and b, for X and Z blocks are indicated by x_a, x_b, z_a, z_b, respectively. (a) H gate acting on qubit a will exchanges x_a and z_a. (b) S gate acting on qubit a results in $z_a \leftarrow z_a \oplus x_a$. (c) $C(Z,X)_{a,b}$, i.e., the CNOT gate, results in $x_b \leftarrow x_b \oplus x_a$ and $z_a \leftarrow z_a \oplus z_b$. (d) $C(X,X)$ transformation is equivalent with successively applying H on a, $C(Z,X)_{a,b}$, and H on a, resulting in $x_a \leftarrow x_a \oplus z_b$ and $x_b \leftarrow x_b \oplus z_a$.

Based on synthesis variants of Pauli-based IR. Dedicated compilers like PAULIHEDRAL[11] and TETRIS [12] leverage Pauli-based IR to identify gate cancellation opportunities between nearest-neighbor IRs, exposed by the variants of their synthesis schemes based on CNOT-tree unrolling. They proposed sophisticated co-optimization techniques to minimize CNOT gate for both logical-level synthesis and SWAP-based routing achieve good performance especially on limited-topology NISQ devices. However, their optimization scope is confined to a finite set of local subcircuit patterns, and they rely solely on the CNOT-based ISA.

III. BSF AND CLIFFORD FORMALISM

We establish our synthesis scheme by representing Pauli strings in the binary symplectic form (BSF). In the BSF, each n-qubit Pauli string is represented as a row in a tableau, with columns divided into two sections: $[X \mid Z]$. Here, $[X_{i,j} \mid Z_{i,j}]$ denotes the j-th component of the i-th Pauli operator. The encoding maps X to $[1 \mid 0]$, Z to $[0 \mid 1]$, Y to $[1 \mid 1]$, and I to $[0 \mid 0]$. For instance, the Pauli string simplification shown in Fig. 1 (b) follows the formulation

$$\left[\begin{array}{ccc|ccc} 0&1&1&1&1&1\\ 0&0&1&1&1&1\\ 1&1&1&0&1&1\\ 1&0&1&0&1&1 \end{array}\right] \xrightarrow{C(X,Y)_{1,2}} \left[\begin{array}{ccc|ccc} 0&1&0&1&1&0\\ 0&0&0&1&1&0\\ 1&1&0&0&1&0\\ 1&0&0&0&1&0 \end{array}\right],$$

where $C(X,Y)_{1,2}$ is the Clifford gate in Fig. 1 (b) while written in the *universal controlled gate* representation

$$C(\sigma_0,\sigma_1) = \frac{1}{2}((I+\sigma_0)\otimes I + (I-\sigma_0)\otimes\sigma_1),\ \sigma_j \in \{X,Y,Z\}\,.$$

Any universal controlled gate is equivalent to CNOT up to local H and S conjugations. For example, $C(Z,X) = \mathrm{CNOT}$, and $C(X,Y) = (H\otimes S)\,\mathrm{CNOT}\,(H\otimes S^\dagger)$.

By definition, any operator C from the Clifford group is unitary and has the property that $CPC^\dagger$ is also a Pauli operator for any Pauli operator P. The update rules of some Clifford operators in terms of the BSF are demonstrated in Fig. 2, in which only 2Q Clifford operators (Clifford2Q) potentially have nontrivial effects of reducing the Pauli strings' weights. For any $C(\sigma_0,\sigma_1)$, its tableau update rule is a combinatorial sequence of H, S and $C(Z,X)$ (or $C(Z,Z)$) transformations. For example, $C(X,Y)_{a,b}$ exhibits the rule

$$[x_a,\ x_b \mid z_a,\ z_b] \rightarrow [x_a \oplus x_b \oplus z_b,\ z_a \oplus z_b \mid z_a,\ z_a \oplus z_b]. \quad (3)$$

Therefore, the basic approach of our synthesis scheme is to find an appropriate sequence of Clifford2Q operators to simplify the BSF[1] until it can be directly synthesized by basic 1Q and 2Q gates, i.e., the quantity "total weight"

$$w_{\text{tot.}} \coloneqq \|\vee_i (r_x^{(i)} \vee r_z^{(i)})\| \tag{4}$$

is no more than 2, where $r_{x/z}^{(i)}$ is the i-th row of X or Z block of the BSF, and the norm is the sum of all binary entries. The key lies in how to find the Clifford2Q that can reduce the weights of as many Pauli strings as possible within the BSF.

IV. Our Proposal: Phoenix

A. Overview

In the proposed framework Phoenix, Pauli-based IRs (exponentiations) are first grouped according to the same set of qubit indices non-trivially acted on. Then, the *BSF simplification* algorithm is applied to the BSF of each IR group, generating the simplified subcircuit composed of CNOT-equivalent Clifford2Q operators and Pauli exponentiations with weights no more than 2. Phoenix further selectively assemble (order) these unarranged simplified IR groups in a Tetris-like style that minimizes a uniform subcircuit assembling cost function that incorporates gate cancellation, circuit depth increase, and qubit routing overhead. By defining appropriate metric functions and heuristic algorithms, Phoenix achieves superior optimization (less 2Q gate count and circuit depth) compared to SOTA methods, efficiently handling compilation for large-scale VQA programs.

B. BSF simplification for each IR group

To search for appropriate Clifford2Q operators in BSF simplification, it suffices to just focus on a set of Clifford2Q generators of the 2Q Clifford group. We choose the six universal controlled gates

$$\{C(X,X), C(Y,Y), C(Z,Z), C(X,Y), C(Y,Z), C(Z,X)\} \tag{5}$$

that are independent of each other and span the entire 2Q Clifford group [15]. It is a natural choice of sets of Clifford2Q generators, as each of them is Hermitian and equivalent to CNOT, and its corresponding BSF's tableau update rule can be defined as a combination of the update rules for H, S, and CNOT.

To determine which Clifford2Q operator to select from Eq. (5), we define a heuristic cost function

$$\begin{aligned}\text{cost}_{\text{bsf}} \coloneqq w_{\text{tot.}} * n_{\text{n.l.}}^2 + \sum\nolimits_{\langle i,j\rangle} \|r_x^{(i)} \vee r_z^{(i)} \vee r_x^{(j)} \vee r_z^{(j)}\| \\ + \frac{1}{2}\sum\nolimits_{\langle i,j\rangle} (\|r_x^{(i)} \vee r_x^{(j)}\| + \|r_z^{(i)} \vee r_z^{(j)}\|)\end{aligned} \tag{6}$$

to quantify the disparity of the current BSF from a desired BSF that requires no further simplification ($w_{\text{tot.}} \leq 2$). cost_{bsf} is the combined weight overlap of both X-part and Z-part among each pair of Pauli strings of a BSF, with a bias considering the impact of the number of nonlocal (n.l., that is, weight larger than 1) Pauli strings.

Algorithm 1 illustrates the BSF simplification procedure in detail. It takes set of Pauli strings as input and outputs a configuration sequence *cfg*, in which each component is either a Clifford2Q generator from Eq. (5) or a BSF with $w_{\text{tot.}}$ at most 2. Specifically, at each Clifford2Q search epoch, Crefalgo:simplification identifies the Clifford 2Q generator from Eq. (5) and qubit pair to act upon that leads to the greatest reduction in the cost function defined in Eq. (6), and the BSF is updated accordingly. This greedy process continues iteratively until the BSF's $w_{\text{tot.}}$ is no more than 2. Before each search epoch, local Pauli strings are peeled from the BSF, as they represent 1Q Pauli rotations and do not induce synthesis overhead. The output *cfg* represents a simplified IR group, while still in high-level semantics (Clifford2Q, 1Q Pauli-based IRs, and successive 2Q Pauli-based IRs), independent of any specific quantum ISA.

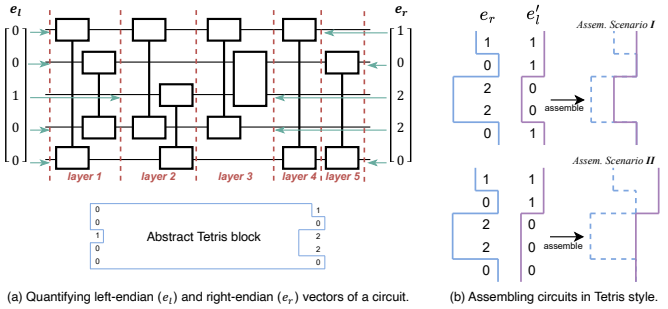
In theory, simplifying a BSF is always achievable, for example, we can reduce an individual Pauli string to be weight-1, peel it off, and then iterate. The efficacy of Algorithm 1 lies in its simultaneous simplification mechanism guided by the cost function in Eq. (6).

[1]The BSF that needs to simplify does not contain any weight-1 Pauli string.

Algorithm 1: Pauli Strings Simplification in BSF

Input : Pauli strings list *pls*
Output: Reconfigured circuit components list *cfg*

cfg ← ∅; *bsf* ← BSF(*pls*); *cliffs_with_locals* ← ∅;
while *bsf*.TOTALWEIGHT() > 2 **do**
  local_bsf ← *bsf*.POPLOCALPAULIS();
  $C \leftarrow \emptyset$; *// Clifford2Q candidates*
  $B \leftarrow \emptyset$; *// Each element of B results from applying each Clifford2Q candidate on bsf*
  costs ← ∅ ; *// Cost functions calculated on each element of B*
  for *cg* ***in*** CLIFFORD_2Q_SET **do**
    for *i, j* ***in*** COMBINATIONS(RANGE(n), 2) **do**
      cliff ← *cg*.ON(i, j) ; *// qubits acted on*
      bsf′ ← *bsf*.APPLYCLIFFORD2Q(*cliff*);
      cost ← CALCULATEBSFCOST(*bsf′*);
      C.APPEND(*cliff*);
      B.APPEND(*bsf′*);
      costs.APPEND(*cost*);
    end
  end
  bsf ← BSFWITHMINCOST(B, *costs*);
  cliff ← CLIFFORDWITHMINCOST(C, *costs*);
  cliffs_with_locals.APPEND((*cliff*, *local_bsf*));
end
cfg.APPEND(*bsf*);
for *cliff, local_bsf* ***in*** *cliffs_with_locals* **do**
  // Clifford2Q operators are added as conjugations, with local Pauli strings peeled before each epoch
  cfg.PREPEND(*cliff*);
  cfg.APPEND(*local_bsf*);
  cfg.APPEND(*cliff*);
end



(a) Quantifying left-endian (e_l) and right-endian (e_r) vectors of a circuit. (b) Assembling circuits in Tetris style.

Fig. 3. Tetris-like circuits assembling. (a) e_l and e_r examples. (b) Scenario I: $\text{cost}_{\text{depth}} = \text{SUM}(e_r + e_l')$; Scenario II: $\text{cost}_{\text{depth}} = \text{SUM}(e_r + e_l' - 1)$.

C. Ordering IR groups in a Tetris-like style

With simplified IR groups compiled via BSF simplification, Phoenix further performs a *Tetris-like IR group ordering* procedure that aims to minimize *a uniform assembling cost metric*. This is unlike [11] and [12] where the ordering is primarily determined by gate

cancellation opportunities between IR groups. We can abstract the circuit structure exhibited by each simplified IR group into something resembling a Tetris block. Then, we choose the ordering based on the circuit-depth cost function of assembling Tetris blocks, with gate cancellation opportunities and qubit routing overhead also taken into account. We primarily have three ingredients:

1) *Endian vectors of circuits and the depth cost function.* As shown in Fig. 3 (a), we define a pair of vectors e_l and e_r for each subcircuit (which would correspond to a simplified IR group in our case). The i-th entry of e_l (e_r) refers to how many layers one has to traverse starting from the left (right) side before qubit i is acted upon. The layers are defined by grouping neighboring 2Q gates that act on different qubits. See Fig. 3 (a) for an example of a generic (not necessarily simplified IR group) circuit. With this definition, the depth overhead for assembling two subcircuits—the proceeding subcircuit with left-endian vectors e_r and the succeeding subcircuit with left-endian vector e'_l is given by
$$\text{cost}_{\text{depth}} \coloneqq \begin{cases} \text{SUM}(e_r + e'_l), \text{ if } \text{ALL}(e_r[e'_l == 0] > 0) \\ \qquad\qquad\qquad \text{and } \text{ALL}(e'_l[e_r == 0] > 0) \\ \text{SUM}(e_r + e'_l - 1), \text{ otherwise} \end{cases}$$
See Fig. 3 (b) for an example.
2) *Clifford2Q cancellation.* The simplified IR group usually exposes Hermitian Clifford2Q operators at the two ends of the subcircuit. Therefore some Clifford2Q gate cancellation opportunities could be exploited, which may or may not decrease overall circuit depth, as depicted in Fig. 4 (a). We can add this consideration into $\text{cost}_{\text{depth}}$: if m pairs Clifford2Q cancelled, while (a) without decreasing subcircuit depth, $\text{cost}_{\text{depth}} \leftarrow \text{cost}_{\text{depth}} - 2m$; (b) with one side's subcircuit depth decreases, $\text{cost}_{\text{depth}} \leftarrow \text{cost}_{\text{depth}} - 2m - n$; (c) with both sides' subcircuit depths decrease, $\text{cost}_{\text{depth}} \leftarrow \text{cost}_{\text{depth}} - 2m - 2n$.
3) *Consideration of qubit routing overhead.* We try to mitigate the routing overhead for hardware-aware compilation by comparing the qubit interaction graphs of subcircuits. This approach is not limited to specific routing schemes (e.g. three-CNOT unrolling of SWAP-based routing, ancilla-based bridge gate [16]) and hardware topologies. Intuitively, two subcircuits with more "similar" qubit interaction behaviors require less mapping transition overhead between them, as demonstrated by Fig. 4 (b). Therefore, we introduce a factor characterized by the similarity between the qubit interaction graphs of two subcircuits in the cost function:
$$\text{cost}_{\text{depth}} \leftarrow \frac{1}{s} \cdot \text{cost}_{\text{depth}}, \quad s = \sum\nolimits_i \frac{\langle D_i, D'_i \rangle}{\|D_i\|_2 \|D'_i\|_2}, \quad (7)$$
where D (D') is the distance matrix of the preceding (succeeding) subcircuit's qubit interaction graph of the tail (head) part. The tail (head) is defined as starting from the right (left) of the subcircuit and incorporating more and more 2Q gates until all qubits are acted upon. $D_{i,j}$ represents the shortest path length between qubit i and qubit j. D_i is the i-th row of D.

In practice, these IR groups are first pre-arranged in descending order of their weight (subcircuit width). PHOENIX looks ahead for a certain number of subcircuits to find the one with the minimum $\text{cost}_{\text{depth}}$ with respect to the last assembled subcircuit. This process iterates until all subcircuits are assembled.

V. EVALUATION

We evaluate the effectiveness of PHOENIX across diverse Hamiltonian simulation programs, quantum ISAs, and device topologies.

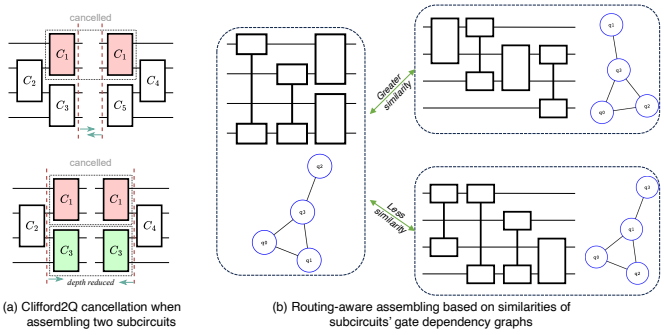


Fig. 4. Gate cancellation opportunities and routing-aware assembling. (a) Clifford2Q cancellation between the preceding and succeeding subcircuits may or may not induce circuit depth decrease. (b) The subcircuit (upper right) whose qubit interaction graph is more similar to that of the already assembled subcircuit (left) is preferred over the other (lower right).

TABLE I
UCCSD BENCHMARK SUITE.

Benchmark	#Qubit	#Pauli	w_{max}	#Gate	#CNOT	Depth	Depth-2Q
CH2_cmplt_BK	14	1488	10	37780	19574	23568	19399
CH2_cmplt_JW	14	1488	14	34280	21072	23700	19749
CH2_frz_BK	12	828	10	19880	10228	12559	10174
CH2_frz_JW	12	828	12	17658	10344	11914	9706
H2O_cmplt_BK	14	1000	10	25238	13108	15797	12976
H2O_cmplt_JW	14	1000	14	23210	14360	16264	13576
H2O_frz_BK	12	640	10	15624	8004	9691	7934
H2O_frz_JW	12	640	12	13704	8064	9332	7613
LiH_cmplt_BK	12	640	10	16762	8680	10509	8637
LiH_cmplt_JW	12	640	12	13700	8064	9342	7616
LiH_frz_BK	10	144	9	2890	1442	1868	1438
LiH_frz_JW	10	144	10	2850	1616	1985	1576
NH_cmplt_BK	12	640	10	15624	8004	9691	7934
NH_cmplt_JW	12	640	12	13704	8064	9332	7613
NH_frz_BK	10	360	9	8303	4178	5214	4160
NH_frz_JW	10	360	10	7046	3896	4640	3674

Although PHOENIX is implemented in Python, it compiles VQA programs of thousands of Pauli strings and more than ten qubits (approximately corresponding to the program size with 10^4-10^6 CNOT gates in conventional synthesis) in dozens of seconds. Code and data are available on GitHub [17]. All experiments are executed on a laptop (Apple M3 Max, 36GB memory).

A. Experimental settings

Metrics. We evaluate PHOENIX using the following metrics: 2Q gate count, 2Q circuit depth, and the algorithmic error for VQA synthesis. Algorithmic error refers to the deviation between the synthesized circuit's unitary matrix and the ideal evolution under the original Hamiltonian, as measured by the infidelity between unitary matrices in our evaluation: $\text{infid} = 1 - \frac{1}{N}|\text{Tr}(U^\dagger V)|$. Notably, we exclude 1Q gates and their count in circuit depth, as 1Q gates are generally considered free resources due to their significantly lower error rates. Additionally, CNOT is not a native operation on most NISQ platforms, requiring extra 1Q drives before and/or after native 2Q gates (e.g., Cross-Resonance [18], Mølmer-Sørensen [19]), making 1Q gate inclusion in metrics potentially misleading.

Baselines. TKET, PAULIHEDRAL, and TETRISare primary baselines to be compared with our method. For TKET, the `PauliSimp` and `FullPeepholeOptimise` passes are adopted for logical circuit optimization. It is similar to TKET's O3 compilation in which the `PauliSimp` pass is particularly effective at optimizing Pauli gadgets. For PAULIHEDRAL, the QISKIT O2 pass is associated by default because the numerous gate cancellation opportunities exposed

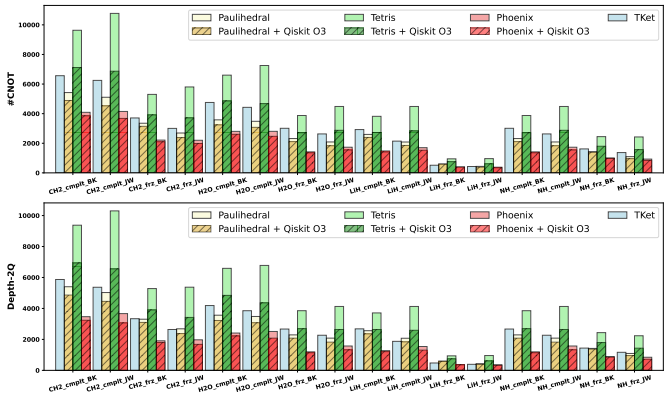


Fig. 5. Logical-level compilation (all-to-all topology).

TABLE II
AVERAGE (GEOMETRIC-MEAN) OPTIMIZATION RATES ON UCCSD.

Compiler	#CNOT opt.	Depth-2Q opt.
TKET	33.07%	30.14%
PAULIHEDRAL	28.41%	29.07%
PAULIHEDRAL + O3	25.72% (-8.54% v.s. no O3)	26.3% (-8.6% v.s. no O3)
TETRIS	53.66%	53.26%
TETRIS + O3	36.73% (-30.94% v.s. no O3)	36.37% (-31.08% v.s. no O3)
PHOENIX	**21.12%**	**19.29%**
PHOENIX + O3	**19.53%** (**-6.64%** v.s. no O3)	**17.28%** (**-8.51%** v.s. no O3)

by PAULIHEDRAL necessitate inverse and commutative cancellations. For hardware-aware compilation, all baselines and PHOENIX are followed by a QISKIT O3 pass with SABRE qubit mapping [20]. For QAOA benchmarking, 2QAN [21] is used as the SOTA baseline.

Benchmarks. We select two representative VQA benchmarks—(1) *UCCSD*: A set of UCCSD ansatzes, including CH_2, H_2O, LiH, and NH: 4 categories of molecule simulation programs. Each category is generated with STO-3G orbitals [22], encoded by Jordan-Wigner (JW) [23] and Bravyi-Kitaev (BK) [24] transformations, in turn approximated by complete or frozen-core orbitals. Details are shown in Tab. I. (2) *QAOA*: A set of 2-local Hamiltonian simulation programs corresponding to random graphs and regular graphs, for which the description and evaluation results are shown in Tab. IV.

B. Main results

Fig. 5 and Tab. II illustrates the main benchmarking results regarding logical-level compilation:

1) PHOENIX significantly outperforms baselines across all benchmarks, with an average (geometric-mean) 21.12% and 19.29% optimization rate in #CNOT and Depth-2Q, respectively, relative to original circuits.[2] That is mostly attributed to the group-wise BSF simplification mechanism, as PHOENIX adopts the same IR grouping method as PAULIHEDRAL and TETRIS.
2) TETRIS performs the worst, falling far behind TKET, PAULIHEDRAL, and PHOENIX. This is because TETRIS focuses primarily on co-optimization techniques to reduce SWAP gates during qubit routing, rather than logical-level synthesis.
3) We also compare PAULIHEDRAL/TETRIS/PHOENIX with and without QISKIT O3, to evaluate their high-level optimization capabilities. The improvement in using QISKIT O3 for PAULIHEDRAL and TETRIS is more pronounced than for PHOENIX. Therefore, PHOENIX's high-level optimization strategy is more impressive, leaving less optimization space for QISKIT O3.

[2]For example, the #CNOT optimization rate is defined as $\frac{\#\text{CNOT}_{\text{after}}}{\#\text{CNOT}_{\text{before}}}$.

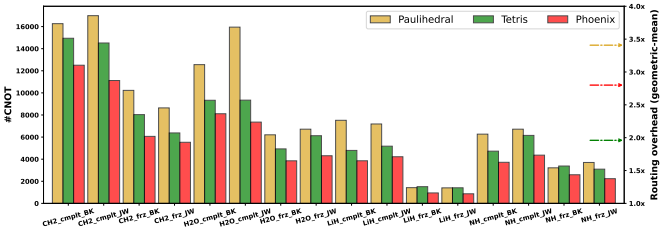


Fig. 6. Hardware-aware compilation on heavy-hex topology. Three dashed lines represent the average multiples of #CNOT within circuits after mapping relative to those after logical optimization, for PAULIHEDRAL (gold), TETRIS (green), and PHOENIX (coral), respectively.

C. Hardware-aware compilation

We use the heavy-hex topology, specifically a 64-qubit coupling graph of IBM's Manhattan processor [25], for hardware-aware compilation. Results are shown in Fig. 6, where TKET is excluded due to its significantly worse performance compared to others. Despite focusing primarily on high-level logical program optimization with humble hardware-aware co-optimization, PHOENIX still outperforms baselines, reducing CNOT gate count by 36.17% (22.62%) and 2Q circuit depth by 43.85% (28.12%) on average, compared to PAULIHEDRAL (TETRIS). Considering the qubit mapping transition overhead mitgation within its IR group ordering process, PHOENIX induces 2.8x #CNOT on average after hardware mapping. It is better than PAULIHEDRAL while worse than TETRIS, as TETRIS specializes in #CNOT cancellation for SWAP-based routing. Consequently, even on limited-topology devices, PHOENIX effectively manages routing overhead and surpasses co-design local optimization strategies.

D. Comparison in diverse ISAs

Quantum ISA, or the native gate set in a narrow sense, serves as an interface between software and hardware implementation. For a specific quantum ISA, the adopted 2Q gate dominates the accuracy and difficulty of hardware implementation, as well as the theoretical circuit synthesis capabilities. While traditional ISAs typically consist of 1Q gates and CNOT-equivalent 2Q gates, recently some works propose integrating complex and even continuous 2Q gates into the ISA design, such as the XY gate family [26], the fractional/partial ZZ and MS gates [27], [28], and the AshN gate scheme which considers all possible 2Q gates within the $\mathrm{SU}(4)$ group as the ISA [14]. Therefore, we further compare PHOENIX with baselines in different ISAs to showcase its ISA-independent compilation advantage.

The ISA-independent advantage is best shown by choosing the most expressive $\mathrm{SU}(4)$ ISA. We evaluate it with both all-to-all and heavy-hex topologies, as the same as the evaluation above for CNOT ISA. For logical-level compilation, PHOENIX directly generates $\mathrm{SU}(4)$-based circuits via its BSF simplification algorithm, while baselines (PAULIHEDRAL and TETRIS equipped with QISKIT O3 by default) require an additional "rebase" (or "transpile") step to convert CNOT-based circuits to $\mathrm{SU}(4)$-based ones. For hardware-aware compilation, all compilers include a rebase step, following the QISKIT O3 hardware-aware compilation pass. Detailed outcomes are summarized in Tab. III, highlighting the geometric-mean relative optimization rates of PHOENIX compared to the baselines' results.

TABLE III
COMPARISON FOR DIVERSE ISAS WITH ALL-TO-ALL AND LIMITED TOPOLOGY.

	CNOT ISA (all-to-all)		SU(4) ISA (all-to-all)		CNOT ISA (heavy-hex)		SU(4) ISA (heavy-hex)	
PHOENIX's opt. rate	#CNOT	Depth-2Q	#SU(4)	Depth-2Q	#CNOT	Depth-2Q	#SU(4)	Depth-2Q
PHOENIX v.s. TKET	63.87%	64.0%	**56.04%**	**54.22%**	**40.63%**	**48.32%**	44.29%	50.71%
PHOENIX v.s. PAULIHEDRAL	82.12%	73.33%	**75.57%**	**65.2%**	62.38%	54.7%	**39.84%**	**35.07%**
PHOENIX v.s. TETRIS	57.52%	53.04%	**56.54%**	**50.55%**	75.97%	71.18%	**62.23%**	**58.74%**

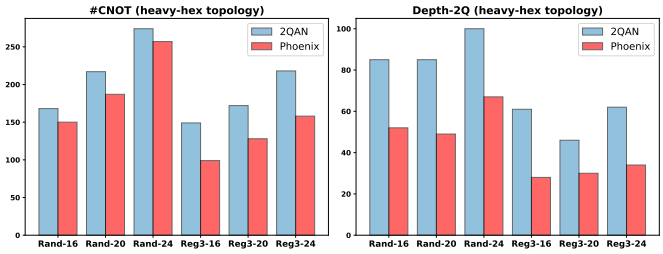


Fig. 7. QAOA benchmarking

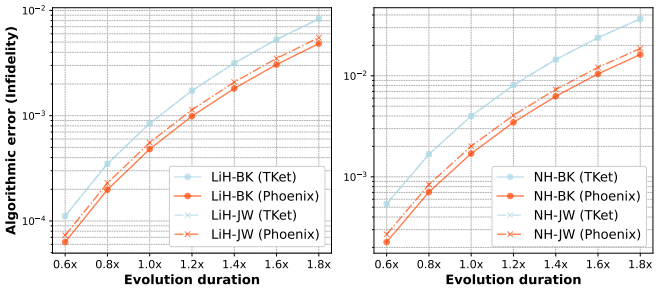


Fig. 8. Algorithmic error comparison of LiH and NH simulation.

TABLE IV
QAOA BENCHMARKING VERSUS 2QAN.

QAOA		#CNOT		Depth-2Q		#SWAP		Routing overhead	
Bench.	#Pauli	2QAN	Phoenix	2QAN	Phoenix	2QAN	Phoenix	2QAN	Phoenix
Rand-16	32	168	150	85	52	37	29	2.62x	2.34x
Rand-20	40	217	187	85	49	47	39	2.71x	2.34x
Rand-24	48	274	257	100	67	63	56	2.85x	2.68x
Reg3-16	24	149	99	61	28	44	17	3.10x	2.06x
Reg3-20	30	172	128	46	30	46	23	2.87x	2.13x
Reg3-24	36	218	158	62	34	62	30	3.03x	2.19x
Avg. improv.		-16.7%		-40.8%		-29.41%		-16.59%	

Again, PHOENIX significantly outperforms baselines when targeting SU(4) ISA. The optimization rates relative to baselines are more impressive than those in CNOT ISA, despite the baselines incorporating sophisticated optimization techniques specifically designed for the CNOT ISA. For instance, the multiple of PHOENIX's #2Q relative to PAULIHEDRAL's in CNOT ISA is 82.12% (62.38%), whereas this value decreases to 75.57% (39.84%) in SU(4) ISA for hardware-agnostic (hardware-aware) compilation. One exception is the hardware-aware compilation comparison with TKET, as TKET's hardware-aware compilation in the CNOT ISA generates circuits with much larger 2Q gate count and circuit depth than other compilers, and there are numerous 2Q subcircuit fusing opportunities such that the rebased circuits involves much fewer SU(4) gates.

E. QAOA benchmarking

For QAOA benchmarking, we focus on the performance in hardware-aware compilation, since the 2Q gate count cannot be reduced and minimizing circuit depth is easy in logical-level compilation. Both 2QAN and PHOENIX can generate depth-optimal QAOA circuits at the logical level in our field test. Fig. 7 and Tab. IV illustrate compilation results on the heavy-hex topology, across six QAOA programs corresponding to both random graphs (each node with degree 4) and regular (each node with degree 3) graphs, with qubit sizes of 16, 20, and 24. PHOENIX outperforms 2QAN across all benchmarks in all metrics (such as #CNOT, #SWAP), especially in Depth-2Q, with an average 40.8% reduction compared to 2QAN. These results further demonstrate the effectiveness of the routing-aware IR group ordering method in PHOENIX.

F. Algorithmic error analysis

We further highlight PHOENIX's advantage in reducing the algorithmic error. We select UCCSD benchmarks with qubits no more than 10 for evaluation, within the matrix computation capabilities of standard PCs. We rescale the coefficients of Pauli strings to control their algorithmic errors within 5×10^{-5} to 10^{-2}, which corresponds to different evolution durations in molecular simulation, as suggested in Fig. 8. In contrast to TKET, PHOENIX typically leads to lower algorithmic errors for both JW and BK encoding schemes. Although this improvement is program-specific, it is more significant for the Pauli string patterns of BK than those of JW, with 57% (42.7%) and 49.5% (34.1%) for NH (LiH) simulation, respectively. As they adopt the same Pauli string blocking approach, we expect the algorithmic errors resulting from PAULIHEDRAL and TETRIS to be comparable to PHOENIX, and so are not shown in Fig. 8. As a result, the impressive algorithmic error reduction effect of PHOENIX brings us closer to a possible quantum advantage on computational chemistry problems.

VI. CONCLUSION AND OUTLOOK

Mainstream circuit synthesis approaches rely on pattern rewrite rules, often restricted to small-scale, local optimizations. In contrast, we present PHOENIX, a framework leveraging high-level Pauli-based IR for Hamiltonian simulation, one of the most prominent NISQ applications. PHOENIX outperforms all SOTA VQA compilers across diverse programs and hardware platforms, showcasing its unmatched performance and versatility. This work not only bridges the gap between impactful quantum applications and physically implementable solutions but also prompts a re-evaluation of compiler optimization.

Beyond producing significantly optimized circuits, high-level IRs, as a refined abstraction layer on top of the traditional quantum ISA, create more opportunities for efficient synthesis and effective hardware mapping. Furthermore, they can act as intermediate building blocks, guiding the design of quantum processors and corresponding control schemes—potentially incorporating multi-qubit control—to implement these high-level IRs more efficiently.

ACKNOWLEDGEMENT

This work is supported by AI Chip Center for Emerging Smart Systems (ACCESS), sponsored by InnoHK funding, Research Grants Council of HKSAR (No. 16213824) and National Key Research and Development Program of China (No. 2023YFA1009403), National Natural Science Foundation of China (No. 12347104), Beijing Natural Science Foundation (No. Z220002).

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[2] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.

[3] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996.

[4] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[5] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[6] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[7] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[8] A. Cowtan, S. Dilkes, R. Duncan, W. Simmons, and S. Sivarajah, "Phase gadget synthesis for shallow circuits," *arXiv preprint arXiv:1906.01734*, 2019.

[9] A. M. van de Griend, "Towards a generic compilation approach for quantum circuits through resynthesis," *arXiv preprint arXiv:2304.08814*, 2023.

[10] J. Paykin, A. T. Schmitz, M. Ibrahim, X.-C. Wu, and A. Y. Matsuura, "Pcoast: a pauli-based quantum circuit optimization framework," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2023, pp. 715–726.

[11] G. Li, A. Wu, Y. Shi, A. Javadi-Abhari, Y. Ding, and Y. Xie, "Paulihedral: a generalized block-wise compiler optimization framework for quantum simulation kernels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 554–569.

[12] Y. Jin, Z. Li, F. Hua, T. Hao, H. Zhou, Y. Huang, and E. Z. Zhang, "Tetris: A compilation framework for vqa applications in quantum computing," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 277–292.

[13] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "$t|ket\rangle$: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.

[14] J. Chen, D. Ding, W. Gong, C. Huang, and Q. Ye, "One gate scheme to rule them all: Introducing a complex yet reduced instruction set for quantum computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. La Jolla, CA, USA: ACM, 2024, pp. 779–796.

[15] D. Grier and L. Schaeffer, "The classification of clifford gates over qubits," *Quantum*, vol. 6, p. 734, 2022.

[16] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, vol. 70, pp. 43–50, 2020.

[17] "PHOENIX GitHub repo," https://github.com/iqubit-org/phoenix.

[18] C. Rigetti and M. Devoret, "Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies," *Physical Review B—Condensed Matter and Materials Physics*, vol. 81, no. 13, p. 134507, 2010.

[19] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, 2019.

[20] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1001–1014.

[21] L. Lao and D. E. Browne, "2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 351–365.

[22] W. J. Hehre, R. F. Stewart, and J. A. Pople, "Self-consistent molecular-orbital methods. i. use of gaussian expansions of slater-type atomic orbitals," *The Journal of Chemical Physics*, vol. 51, no. 6, pp. 2657–2664, 1969.

[23] P. Jordan and E. Wigner, "über das paulische äquivalenzverbot. z phys 47: 631," 1928.

[24] S. B. Bravyi and A. Y. Kitaev, "Fermionic quantum computation," *Annals of Physics*, vol. 298, no. 1, pp. 210–226, 2002.

[25] G. J. Mooney, G. A. White, C. D. Hill, and L. C. Hollenberg, "Whole-device entanglement in a 65-qubit superconducting quantum computer," *Advanced Quantum Technologies*, vol. 4, no. 10, p. 2100061, 2021.

[26] D. M. Abrams, N. Didier, B. R. Johnson, M. P. d. Silva, and C. A. Ryan, "Implementation of xy entangling gates with a single calibrated pulse," *Nature Electronics*, vol. 3, no. 12, pp. 744–750, 2020.

[27] IBM Quantum, "New fractional gates reduce circuit depth for utility-scale workloads," https://www.ibm.com/quantum/blog/fractional-gates, 2024, accessed: Nov. 18, 2024.

[28] IonQ, "Getting started with ionq's hardware-native gateset," https://docs.ionq.com/guides/getting-started-with-native-gates, 2023.

Logic Optimization Meets SAT: A Novel Framework for Circuit-SAT Solving

Zhengyuan Shi[1,4], Tiebing Tang[1], Jiaying Zhu[1,4], Sadaf Khan[1],
Hui-Ling Zhen[2], Mingxuan Yuan[2], Zhufei Chu[3] and Qiang Xu[1,4]
[1]*Department of Computer Science and Engineering*, *The Chinese University of Hong Kong*, Hong Kong S.A.R.
[2]*Noah's Ark Lab*, *Huawei*, Hong Kong S.A.R.
[3]*Faculty of Electrical Engineering and Computer Science*, *Ningbo University*, Ningbo, China
[4]*National Center of Technology Innovation for EDA*, Nanjing, China

***Abstract*—The Circuit Satisfiability (CSAT) problem, a variant of the Boolean Satisfiability (SAT) problem, plays a critical role in integrated circuit design and verification. However, existing SAT solvers, optimized for Conjunctive Normal Form (CNF), often struggle with the intrinsic complexity of circuit structures when directly applied to CSAT instances. To address this challenge, we propose a novel preprocessing framework that leverages advanced logic synthesis techniques and a reinforcement learning (RL) agent to optimize CSAT problem instances. The framework introduces a cost-customized Look-Up Table (LUT) mapping strategy that prioritizes solving efficiency, effectively transforming circuits into simplified forms tailored for SAT solvers. Our method achieves significant runtime reductions across diverse industrial-scale CSAT benchmarks, seamlessly integrating with state-of-the-art SAT solvers. Extensive experimental evaluations demonstrate up to 63% reduction in solving time compared to conventional approaches, highlighting the potential of EDA-driven innovations to advance SAT-solving capabilities.**

I. Introduction

The Circuit Satisfiability (CSAT) problem lies at the heart of many critical tasks in Electronic Design Automation (EDA), such as logic equivalence checking (LEC), automatic test pattern generation (ATPG), and logic synthesis (LS). It involves determining whether a Boolean circuit has an input assignment that satisfies its primary output. As a specialized variant of the Boolean Satisfiability (SAT) problem, CSAT is known to be NP-complete [1], making it both theoretically challenging and practically important in integrated circuit (IC) design and verification.

Modern SAT solvers, such as MiniSAT, Kissat, and CaDiCaL [2], [3], have been highly effective in solving problems represented in Conjunctive Normal Form (CNF). However, when applied to CSAT problems, these solvers face inefficiencies stemming from the required transformation of circuits into CNF format. This transformation often disrupts the natural structure of circuits, leading to representations that are less solver-friendly and more complex to process [4], [5]. As a result, solving CSAT instances may become unnecessarily time-consuming, particularly for large-scale, industrial circuits.

While preprocessing techniques for CNF have been extensively explored [4]–[7], similar advancements tailored specifically for circuit-based representations remain underdeveloped. Circuit-specific methods often focus on reducing circuit size or area [8], which, while useful for design optimization, do not directly address the complexity of SAT solving. This gap highlights the need for a new approach that bridges the structural strengths of circuit representations with the solving power of modern SAT solvers.

To address these limitations, we propose a novel preprocessing framework tailored to CSAT problems. Our approach integrates EDA techniques with reinforcement learning (RL) to transform CSAT instances into SAT-friendly formats. By leveraging logic synthesis tools and a cost-customized Look-Up Table (LUT) mapping strategy, our framework achieves significant reductions in solving complexity. Unlike traditional preprocessing methods, our approach explicitly prioritizes SAT-solving efficiency rather than widely-used optimization metrics like area or delay. Specifically, the main contributions of our work include:

- **EDA-Driven Preprocessing Framework**: We introduce a comprehensive preprocessing pipeline that combines logic synthesis techniques with reinforcement learning to optimize CSAT instances for efficient SAT solving. This integration bridges the gap between raw circuit structures and modern SAT solvers.
- **RL-Guided Logic Synthesis for SAT Solving**: We formulate logic synthesis as a Markov Decision Process (MDP) and train an RL agent to identify synthesis sequences that minimize solving complexity. The agent leverages circuit features inspired by [9], [10] and structural circuit embeddings from DeepGate2 [11].
- **Cost-Customized LUT Mapping for SAT Solving**: We develop a novel LUT mapping strategy that incorporates a customized cost metric reflecting branching complexity during SAT solving. This approach diverges from conventional LUT mapping methods [12], [13] by focusing on solving efficiency.

Using industrial-scale benchmarks derived from LEC and ATPG tasks, we demonstrate significant runtime reductions up to 63% compared to baseline approaches. Our method integrates seamlessly with state-of-the-art SAT solvers, such as Kissat and CaDiCaL.

Corresponding author: Qiang Xu (qxu@cse.cuhk.edu.hk)

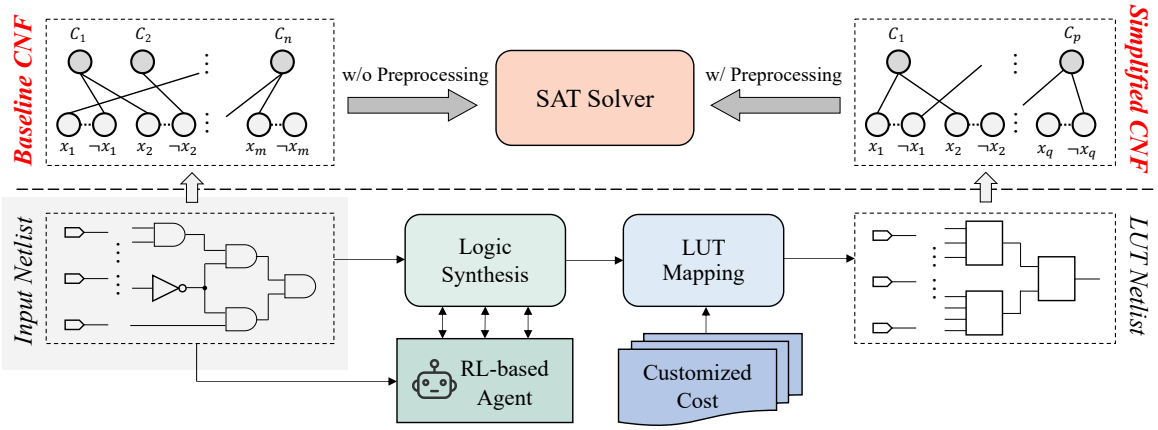


Fig. 1: Overview of EDA-Driven SAT preprocessing framework

II. Related Work

A. SAT and Circuit SAT Problems

SAT problem holds a pivotal position in hardware design, software verification, planning and scheduling, which seeks to identify if there exists at least one assignment that makes a given Boolean formula True. The modern SAT solvers, central in addressing SAT problems, predominantly rely on the Conflict-Driven Clause Learning (CDCL) algorithm [14]. Within the backbone CDCL algorithm, CNF has become a standard format in SAT solving, capable of representing arbitrary Boolean formulas. It is structured as a conjunction of clauses, denoted by $\phi = (C_1 \wedge C_2 \wedge \ldots)$, with each clause being a disjunction of variables or their negations, e.g., $C_i = (x_1 \vee \neg x_2 \vee \ldots)$. Over recent decades, numerous CNF-based heuristics have been developed to enhance SAT solving efficiency, including advanced clause management strategies [15] and innovative branching heuristics [16], [17].

Typically, practical CNF-modeled problems often result in redundant clauses and increased complexity. For example, a tournament scheduling problem with n teams can escalate to n^6 clauses [18]. Consequently, problem reformulation, as known as preprocessing, is essential before solving. Efforts like [4]–[6] target minimizing the CNF instances by resolution and graph transformation, resulting in faster SAT solving.

One of the most representative SAT variants in the realm of EDA is the Circuit SAT (CSAT) problem. In contrast to the general SAT problem, the CSAT problem determines the satisfiability of a Boolean circuit. Over the past decades, circuit-based SAT solvers have been developed to tackle the CSAT problems [19]–[21]. Despite these solvers conducting logic reasoning in natural formats, they still exhibit lower performance levels compared to CNF-based SAT solvers due to the inefficient circuit-based preprocessing and heuristics. For example, the previous circuit-based preprocessing approaches, such as [8], cannot promise to reduce solving time as they only scale down circuits. In this work, we introduce an efficient and effective circuit-based preprocessing framework to directly mitigate solving complexity by integrating EDA tools and reinforcement learning guidance.

B. Boolean Circuit Optimization

Boolean circuits, representable as directed acyclic graphs (DAGs) $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with logic gates and wires serving as nodes $\mathcal{V}$ and edges $\mathcal{E}$, respectively, are pivotal in computational logic. In the EDA community, a wealth of efficient algorithms has been developed for circuit optimization. Logic Synthesis (LS) is a prime example, involving DAG-based transformation techniques to modify circuit's local structure for the optimization of performance, power, and area (PPA).

Besides, technology mapping, a key component in EDA flow, fine-tunes performance metrics under specific constraints [22] and maps the intermediate circuits into standard cells pre-defined in technology library. One widely used technique for technology mapping is Look-Up Table (LUT) mapping [12]. LUT mapping involves the transformation of gate combinations into k-LUTs, where each k-LUT represents a configurable memory unit capable of implementing arbitrary Boolean functions with k variables.

The advent of circuit representation learning [23] marks a progressive trend in the field, focusing on deriving neural representations of circuits to facilitate downstream tasks. The DeepGate family [11], [24], [25], for instance, integrates structural and functional aspects of circuits and has shown impressive results in testability analysis [26] and SAT solving [21]. In our research, we tailor existing EDA tools for specific requirements, utilizing DeepGate2's [11] circuit knowledge learning capabilities to inform our circuit optimization decisions.

III. Methodology

A. Overview of the Proposed Framework

Fig. 1 shows the overview of the proposed circuit-based preprocessing framework. Given a CSAT problem represented in And-Inverter Graphs (AIGs), which only contain three gate types (primary input, AND and NOT) by [27], the baseline pipeline converts it into CNF.

In our proposed framework, we first apply a predetermined sequence of AIG transformation operations to unify the distribution of input circuits [21]. Then, we utilize logic synthesis technique to reformulate SAT instance. Since the

logic synthesis process can be considered as a sequential decision-making task, we train an agent to minimize the solving complexity by reinforcement learning (RL). In the RL framework, the state comprises embeddings of the initial AIG netlist and features of intermediate circuits. Additionally, the RL agent operates within a discrete action space, where each action corresponds to executing a synthesis operation. We approximate the complexity of SAT solving based on the number of variable branching times and define the reduction in branching times as the reward for RL agent.

Finally, we employ the LUT mapping technique to hide the internal logic of circuits. To further decrease the solving time, we design a cost-customized mapping operation with the objective of minimizing the branching times during SAT solving, rather than solely focusing on reducing the problem size. After the LUT mapping, we obtain the LUT netlist with a relatively small number of cells. To ensure compatibility with modern SAT solvers, we transform the LUT netlist back into a simplified CNF.

B. Logic Synthesis Recipe Exploration

We formulate LS as a Markov Decision Process (MDP) process, with the optimization objective of reducing the solving time for a given input instance. Consequently, such sequential decision-making process entails iteratively selecting a LS operation as an action according to the current circuit state.

1) Reinforcement Learning Formulation: To explore the optimal LS sequence (i.e. LS recipe), we employ Deep Q-learning algorithm in this section to train a reinforcement learning (RL) agent. The RL agent is trained to make informed decisions to minimize solving time by learning from its interactions with the circuit environment.

As shown in Fig. 2, at each step t (where $t = 0, 1, ..., T-1$), the features of the current netlist $\mathcal{G}^t$ are extracted to form the state s^t. Subsequently, the Q-learning agent utilizes s^t as input and selects an action a^t. Then, the environment including the LS tool performs the synthesis operation based on a^t and transforms the netlist $\mathcal{G}^t$ into a new netlist $\mathcal{G}^{t+1}$. At the end of this step, the environment provides RL agent with a reward r^t. The above process is repeated for subsequent steps until $t = T - 1$. During the training process, the objective of RL agent is to maximize the cumulative sum of the reward values. As a result, the trained RL agent is able to select LS operations step by step to achieve the optimal solving time reduction.

2) State: Inspired by [9], [10], we extract the following representative features $\mathcal{E}(\mathcal{G}^t)$ of the netlist $\mathcal{G}^t$ into state s^t:

- Ratio between the area of netlist $\mathcal{G}^t$ and netlist $\mathcal{G}^0$.
- Ratio between the depth of netlist $\mathcal{G}^t$ and netlist $\mathcal{G}^0$.
- Ratio between the wire counts of netlist $\mathcal{G}^t$ and netlist $\mathcal{G}^0$.
- Proportion of AND gates in the total gates of netlist $\mathcal{G}^t$.
- Proportion of NOT gates in the total gates of netlist $\mathcal{G}^t$.
- Average balance ratio of netlist $\mathcal{G}^t$. We formally define the average balance ratio br in Eq. (1), where $\mathcal{P}_{i1}$, $\mathcal{P}_{i2}$ are predecessors of two fanin AND gate i, and d denotes the depth.

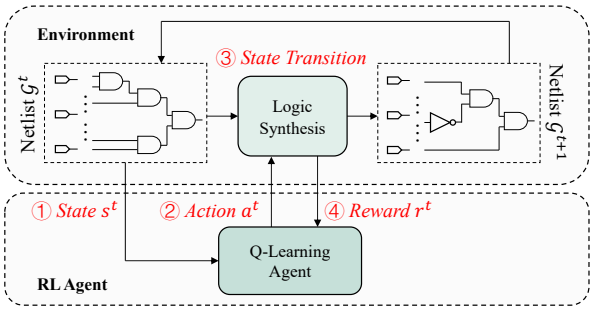


Fig. 2: RL Agent-Environment interaction

$$br = \sum_{i \in AND} \frac{abs(d_{\mathcal{P}_{i1}} - d_{\mathcal{P}_{i2}})}{max(d_{\mathcal{P}_{i1}}, d_{\mathcal{P}_{i2}})} \; / \; \#AND \tag{1}$$

The above features exhibit characteristics that can be effectively optimized by various logic synthesis operations. Consequently, the RL agent can intelligently select appropriate operations based on these features. For example, when a netlist shows a higher average balance ratio, indicating an imbalanced structure in DAG, the agent is more inclined to select the action of operation *balance*. Additionally, a large area ratio prompts the agent to opt for operations that aim to minimize the circuit area, such as *rewrite*.

Besides, the state s^t is incorporated with the primary output embeddings of the initial netlist obtained by pretrained DeepGate2 [11] (denoted as $\mathcal{D}(\mathcal{G}^0)$), which contains rich structural and functional information of the initial problem instance.

$$s^t = \text{Concatenate}(\mathcal{E}(\mathcal{G}^t), \mathcal{D}(\mathcal{G}^0)) \tag{2}$$

3) Action: The action space $\mathcal{A}$ of agent is discrete and encompasses LS operations. In this paper, we set the available operations as follows: *rewrite* [28], *refactor* [29], *balance* [30], *resub* [31] and *end* (marking the end of the LS process). The selection of these operations does not imply that the proposed framework lacks support for additional synthesis operations beyond this set. Instead, these operations are chosen due to their widespread usage and commonality. If there is a need to include additional optional operations, it can be accomplished by expanding the action space within the framework.

4) State Transition: The state transition function is implemented by the LS tool. We call the tool with a specific LS operation determined by RL agent to transform the current netlist $\mathcal{G}^t$ to another functional-equivalent but simplified netlist. More formally, we note the given netlist as $\mathcal{G}^t$ and RL action as a^t at step t. The state transition function is denoted as: $\mathcal{G}^{t+1} = \mathcal{F}(\mathcal{G}^t, a^t)$, where $\mathcal{G}^{t+1}$ is the updated netlist by tool. The updated netlist, $\mathcal{G}^{t+1}$, is then considered as the input for the subsequent decision step at time $t + 1$.

5) Reward: We consider the reduction of variable branching times during SAT solving as the reward. The reward function is shown as Eq. (3), where $\Delta\#Branching$ is the difference in branching times between the final instance and the initial instance during solving. We opt for the terminated reward, which has only one non-zero reward value at the

terminated step, i.e. after performing the entire sequence of synthesis operations.

$$r^t = \begin{cases} 0, t = 0, \cdots, T-2, \\ \text{-}\Delta\#Branching,\ t = T-1 \text{ or } a^t = \textit{end} \end{cases} \tag{3}$$

Prior to solving the SAT instance in circuit format, we employ the LUT mapping to convert the AIG netlist into an LUT netlist. Subsequently, the transformed netlist is converted into CNF, which will be elaborated in the following section. Additionally, we choose not to minimize the solver runtime based on two primary reasons. Firstly, determining the solver runtime for hard SAT instances is time-consuming. Secondly, the solver runtime is challenging to calculate accurately due to the fluctuating CPU status in the case of easy SAT instances. We utilize variable branching times to approximate solving time. This approach allows us to include easy cases with short solver runtime in the training dataset, thereby enhancing the efficiency of training the RL agent.

6) Policy: In the Q-learning framework [32], the policy for maximizing the total reward entails selecting the action with the highest expected value. The action value function Q uses the multilayer perceptron (MLP) to estimate the sum of future rewards after taking a certain action in the current state. The policy $\pi(s^t)$ and action-value function $Q_\theta(s^t, a)$ are shown in Eq. (4), where the θ parametrizes the function Q.

$$\begin{aligned} \pi(s^t) &= \arg\max_{a, a\in\mathcal{A}} (Q(s^t, a)) \\ Q_\theta(s^t, a) &= Index(MLP(s^t), a) \end{aligned} \tag{4}$$

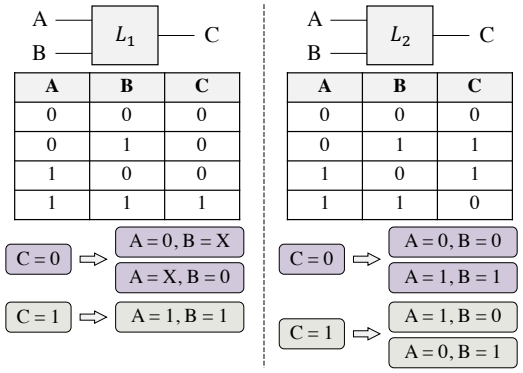
We create the target function $Q_{\hat{\theta}}(s^t, a)$ to stabilize the training process. The parameter $\hat{\theta}$ is copied from θ after k steps. The parameter θ of action-value function Q is updated iteratively by loss function in Eq. (5), where γ is a hyperparameter and represents discount factor.

$$l = ||Q_\theta(s^t, a^t) - r^t - \gamma \arg\max_{a\in\mathcal{A}} Q_{\hat{\theta}}(s^{t+1}, a)||_2 \tag{5}$$

C. Cost-Customized LUT Mapping

We present a cost-customized LUT mapping approach that not only hides the internal nodes during solving, but also forms an easy-to-solve instance. Unlike conventional mapping tools that focus on constructing circuits with low area or delay, our approach aims to create circuits that are easier to solve. Specifically, we begin by defining a cost metric to quantify the branching complexity of SAT solving and compute the cost values for all 4-input LUTs. Next, we modify the cost function in the existing mapping tool to enable minimizing the overall branching complexity. Lastly, we apply the cost-customized mapper to map AIG netlist to LUT netlist and convert the LUT netlist into CNF.

1) Branching Complexity of LUT: The SAT solving process involves iteratively making variable branching decisions, where the solver selects variable assignments as either *true* or *false*. We formulate the solving process within the circuit domain. Specifically, given the logic value of a gate, the SAT solver makes decisions by selecting assignments for its fanin gates. We consider two 2-fanin LUTs, denoted as L_1 (representing an AND gate) and L_2 (representing an XOR gate), and list the truth tables for these LUTs in Fig. 3.



A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3: LUTs with different branching complexity

We observe that both L_1 and L_2 have two possible combinations when the fanout pin (C) is logic-0. However, when the fanout pin is logic-1, L_2 has two possible combinations while L_1 has only one unique branch. We define the *branching complexity* C as the total number of possible fanin combinations. In this example, the LUT L_1 has $C_{L_1} = 3$ and LUT L_2 has $C_{L_2} = 4$. Therefore, L_2 is more hard to be solved than L_1.

2) Cost-Customized Mapper: Since a large number of possible branches result in an exponential search space, various heuristics [14], [33] are employed to reduce the number of potential branches and improve the efficiency of the solving process. Besides, utilizing techinque mapping tools facilitates the concealment of intermediate gates and decreases the overall number of branches [8].

In contrast to [8] that minimizes the size of the post-mapping netlist, our approach prioritizes mapping the logic into cells with lower branching complexity. Our approach is motivated by the observations that instances containing a significant proportion of XOR gates often require more time to be solved [34], [35], where the branching complexity of XOR gate (L_2 in Fig. 3) is higher than other gates.

To prioritize the LUTs during mapping process, we employ a strategy to enumerate all 4-LUTs and integrate their branching complexity into the cost function. We implement a cost-customized mapper based on mockturtle [13] by modifying the area cost to reflect the branching complexity of each LUT and fixing the delay cost as a constraint. By executing a sequence of operations aimed at minimizing the total cost, the resulting post-mapping netlists exhibit minimal overall branching complexity.Our preprocessing strategy ends with a LUT to CNF transformation [36] for compatibility.

As a result, we formally depict our overall circuit-based preprocessing framework coupled with RL-based logic synthesis exploration and cost-customized LUT mapping in Algorithm 1. The framework begins by transforming the input circuit instance $\mathcal{G}_{in}$ into AIG format, using $aigmap$ technique in ABC [37]. Then, the RL agent (elaborated in Section III-B) proceeds to select logic synthesis operations step by step through making action a^t. During this process (line 6-16),

Algorithm 1: CSAT preprocessing framework

Input: The input circuit instance $\mathcal{G}_{in}$
Output: The output simplified CNF instance ϕ_{out}
/* Initialize circuit graph $\mathcal{G}^0$ in AIG */
if *input instance is in AIG* **then**
  $\mathcal{G}^0 = \mathcal{G}_{in}$;
else
  $\mathcal{G}^0 = aigmap(\mathcal{G}_{in})$;
end
/* Explore logic synthesis recipe by RL-agent (see Section III-B) */
Define maximum step count: T;
$t = 0$;
while $t < T$ **do**
  /* Prepare the state of current circuit */
  $s^t = \text{Concatenate}(\mathcal{E}(\mathcal{G}^t), \mathcal{D}(\mathcal{G}^0))$;
  /* Make decision to choose a logic synthesis operation */
  $a^t = \pi(s^t) = \arg\max_{a, a \in \mathcal{A}}(Q(s^t, a))$;
  if a^t *is end* **then**
    break;
  end
  /* Perform logic synthesis for state transition */
  $\mathcal{G}^{t+1} = \mathcal{F}(\mathcal{G}^t, a^t)$;
  $t = t + 1$;
end
/* Perform cost-customized LUT mapping (see Section III-C) */
Define LUT costs: C ;
$\mathcal{G}_{LUT} = mapper(\mathcal{G}^t, C)$;
/* Convert to CNF */
$\phi_{out} = lut2cnf(\mathcal{G}_{LUT})$;
return ϕ_{out};

the circuit is synthesized multiple times to create a simplified circuit. Next, the framework continues to perform cost-customized LUT mapping on the AIG netlist $\mathcal{G}^t$, resulting in a compact LUT netlist $\mathcal{G}$. Finally, the LUT netlist is encoded into CNF as output ϕ_{out}. Due to these simplification approaches, ϕ_{out} has a lower solving complexity than ϕ_{in} and the subsequent SAT solving procedures can be more efficient. We will assess the effectiveness of our preprocessing by the following experiments.

IV. Experiments

A. Experiment Settings

We collect 200 easy (solving time $0.04s - 6.68s$) instances into the RL training dataset and 300 hard instances for testing. All instances are in AIG format and derived from both industrial logic equivalence checking (LEC) and automatic test pattern generation (ATPG) problems. For LEC instances, we modify original industrial datapath circuits and connect their primary outputs (POs) through XOR gates to form CSAT instances, where satisfiability indicates non-equivalent functionality between circuits. For ATPG instances, we introduce stuck-at faults into industrial circuits and connect the POs of faulty and fault-free circuits through XOR gates, where satisfiable assignments serve as test patterns for fault detection. We construct 200 LEC instances and 100 ATPG instances in total, where 200 easy instances are selected for RL training based on their clause count after CNF transformation and solving time without preprocessing. The statistics of training instances are shown in Table I, where # Clauses shows the number of clauses after transforming AIG instances into CNFs and Time represents the solving time without any preprocessing.

TABLE I: Statistics of training dataset

	Avg.	Std.	Min.	Max.
# Gates	4,299.06	4,328.16	60	24,178
# PIs	43.66	25.17	6	102
Depth	66.43	19.98	18	138
# Clauses	10,687.28	10,801.96	131	60,294
Time (s)	2.01	1.96	0.04	6.68

In the RL setting, we assign the maximum number of steps per episode as $T = 10$ and the discount factor $\gamma = 0.98$. During each training episode, the agent randomly selects an instance and makes decisions aimed at transforming the instance to minimize the number of branching times in SAT solving. We conduct a total of $10,000$ episodes, i.e., the RL agent preprocesses instances for $10,000$ times during RL training. The batch size is 32 and learning rate is 10^{-5}.

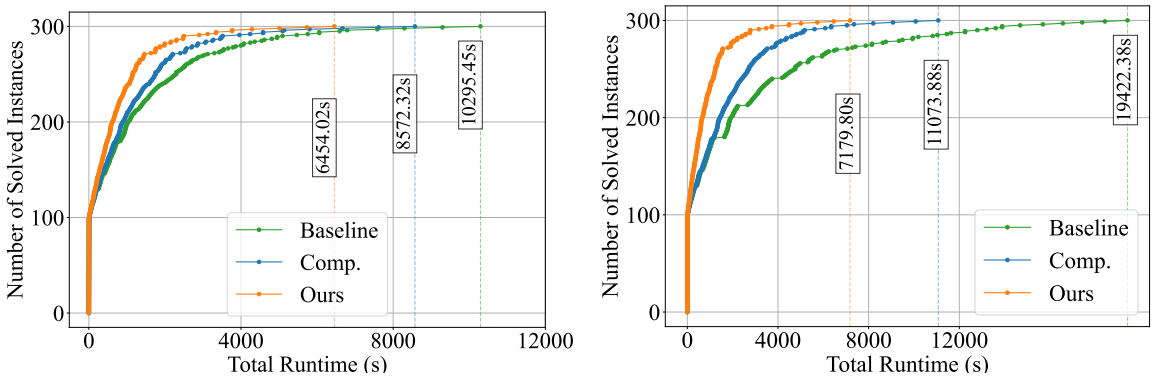
We opt for the ABC tool [37] to synthesize the circuit and build a cost-customized mapper based on mockturtle [13] to map the circuit in LUT netlists. The solver runtime limitation is $1,000$s. Any cases that exceed this time limit are marked with TO (Time-out) and are considered to have a solving time of $1,000$ seconds ($\mathcal{T}_{solve} = 1,000s$). Each instance solving is conducted on a single core of Intel Xeon CPU E5-2630.

B. Runtime Comparison

Figure 4 illustrates the relationship between the number of instances solved and overall runtime (including RL model inference time, transformation time and solving time). The Baseline setting represents the conventional solving pipeline, encoding the circuit-based instances directly into CNFs. Our experiment evaluates the effectiveness of our proposed framework using both Kissat solver (version 4.0.0 [38]) and CaDiCaL solver (version 2.0.0 [39]), with our specific settings distinguished by the orange line (denoted as Ours).

To showcase the efficiency of our preprocessing framework, we compare the solving time with another circuit-based approach[1] [8], denoted as Comp. . To the best of our knowledge, although such approach has been around for over a decade, this is the only circuit-based SAT preprocessing technique.

[1]We do not include comparisons with CNF-based preprocessing approaches because our framework is not mutually exclusive with them and can be used in conjunction. Pratically, we keep the default CNF-based preprocessing.



(a) Kissat (c) Cadical

Fig. 4: Runtime Comparison with Baseline and Comp.

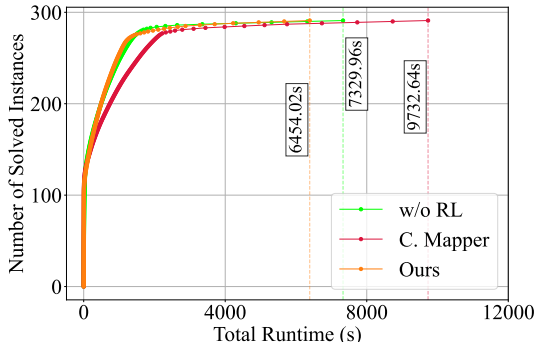


Fig. 5: Ablation Study

According to Figure 4, our proposed circuit-based proprocessing framework can significantly reduce runtime. To be specific, solving these 300 instances necessitates 19,422.38 seconds for the CaDiCaL Baseline pipeline and 11,073.88 seconds for the comparative pipeline (Comp.). In contrast, our method completes the task in a total of 7,179.80 seconds, marking a reduction of 63.03% and 35.16% in solving time compared to Baseline and Comp., respectively.

C. Ablation Studies

In the following experiments, we investigate the effectiveness of our RL-based agent and cost-customized mapper. We keep the same experiment environments and test cases above.

1) Effectiveness of Logic Synthesis Agent: To investigate the effectiveness of the RL agent, we conduct an ablation study by introducing another agent that randomly selects LS operations for $T = 10$ times (noted as w/o RL). The original setting with RL agent is noted as Ours. We test these two settings and summarized the results in Figure 5.

We observe that although the agent with a random policy is capable of employing LS operations to simplify the circuit, it still falls short in terms of achieving optimal solving time reduction. To be specific, solving these 300 instances necessitates 7329,96 seconds for the w/o RL and 6454.02 seconds for our w/ RL agent, marking a reduction of 11.95%.

2) Effectiveness of Cost-Customized Mapper: To showcase the effectiveness of our proposed cost-customized mapper, we replace our cost-customized mapper with a mapper with conventional cost metrics. We denote the new setting as C. Mapper and our original setting as Ours. It is worth noting that both settings utilize the same RL agent with same parameters, and thus, we disregard the agent runtime ($\mathcal{T}_{agent}$) in the reported results.

As shown in Figure 5, the 300 instances processed by the conventional mapper take 9732.64 seconds to solve, which is 50.80% longer than the solving time of Our Mapper. We attribute this difference to the divergent optimization objectives: the conventional mapper minimizes area and delay, while our cost-customized mapper targets solving complexity directly.

V. Conclusion and Future Work

This paper introduces a novel circuit-based preprocessing framework for circuit satisfiability problem, integrating logic synthesis with reinforcement learning to transform circuits into solver-friendly formats. Key innovations include an RL-guided logic synthesis pipeline for optimizing circuit transformations and a cost-customized LUT mapping strategy that minimizes branching complexity during SAT solving. Experiments on industrial benchmarks demonstrate significant runtime reductions—up to 63% compared to baseline approaches—while maintaining compatibility with state-of-the-art solvers. Ablation studies validate the effectiveness of both the RL agent and cost-customized mapper in improving solving efficiency.

This work highlights the potential of combining EDA innovations with machine learning to advance SAT-solving methodologies. Beyond CSAT, the proposed framework offers a promising direction for tackling other SAT variants and related combinatorial optimization problems, enabling more efficient and scalable solutions. Future work will explore extending this approach to sequential circuits and broader EDA applications.

Acknowledgments

This work was partly supported by the General Research Fund of the Hong Kong Research Grants Council (RGC) under Grant No. 14212422 and 14202824 and partly by the National Technology Innovation Center for EDA.

REFERENCES

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd ACM symposium on Theory of computing*, 1971, pp. 151–158.

[2] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *Theory and Applications of Satisfiability Testing – SAT 2005*, vol. 2005, no. 53, pp. 1–2, 2005.

[3] A. Biere, K. Fazekas, A. Fleury, and M. Heisinger, "Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020," *SAT COMPETITION*, vol. 2020, p. 50, 2020.

[4] R. Ostrowski, É. Grégoire, B. Mazure, and L. Sais, "Recovering and exploiting structural knowledge from cnf formulas," in *Principles and Practice of Constraint Programming*. Springer, 2002, pp. 185–199.

[5] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination," in *Theory and Applications of Satisfiability Testing – SAT 2005*. Springer, 2005, pp. 61–75.

[6] S. Subbarayan and D. K. Pradhan, "Niver: Non-increasing variable elimination resolution for preprocessing sat instances," in *Theory and Applications of Satisfiability Testing – SAT 2005*. Springer, 2005, pp. 276–291.

[7] C. Condrat and P. Kalla, "A gröbner basis approach to cnf-formulae preprocessing," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2007, pp. 618–631.

[8] N. Eén, A. Mishchenko, and N. Sörensson, "Applying logic synthesis for speeding up sat," in *Theory and Applications of Satisfiability Testing – SAT 2007*. Springer, 2007, pp. 272–286.

[9] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *Proceedings of the 25th Asia and South Pacific Design Automation Conference*. IEEE, 2020, pp. 581–586.

[10] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD*, 2020, pp. 145–150.

[11] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "Deepgate2: Functionality-aware circuit representation learning," in *Proceedings of the 2023 IEEE/ACM international conference on Computer-aided design*, 2023.

[12] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1991, pp. 613–619.

[13] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage *et al.*, "The epfl logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

[14] J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning sat solvers," in *Handbook of satisfiability*. IOS press, 2021, pp. 133–182.

[15] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009, pp. 399–404.

[16] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, 2001, pp. 530–535.

[17] Z. Shi, M. Li, S. Khan, H.-L. Zhen, M. Yuan, and Q. Xu, "Satformer: Transformers for sat solving," in *Proceedings of the 2023 IEEE/ACM international conference on Computer-aided design*, 2023.

[18] R. Béjar and F. Manya, "Solving the round robin problem using propositional logic," in *Proceedings of the AAAI Conference on Artificial Intelligence, 17*, 2000, pp. 262–266.

[19] C.-A. Wu, T.-H. Lin, C.-C. Lee, and C.-Y. Huang, "Qutesat: a robust circuit-based sat solver for complex circuit structure," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 200*. IEEE, 2007, pp. 1–6.

[20] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko, "A circuit-based sat solver for logic synthesis," in *Proceedings of the 2021 IEEE/ACM international conference on Computer-aided design*, 2021, pp. 1–6.

[21] M. Li, Z. Shi, Q. Lai, S. Khan, S. Cai, and Q. Xu, "On eda-driven learning for sat solving," in *Proceedings of the 60th ACM/IEEE Design Automation Conference*. IEEE, 2023, pp. 1–6.

[22] M. R. Berkelaar and J. A. Jess, "Technology mapping for standard-cell generators," in *Proceedings of the 1988 IEEE/ACM international conference on Computer-aided design*, 1988, pp. 470–473.

[23] L. Chen, Y. Chen, Z. Chu, W. Fang, T.-Y. Ho, R. Huang, Y. Huang, S. Khan, M. Li, X. Li *et al.*, "Large circuit models: opportunities and challenges," *Science China Information Sciences*, vol. 67, no. 10, p. 200402, 2024.

[24] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 667–672.

[25] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: Towards scalable circuit representation learning," in *Proceedings of the 2024 IEEE/ACM international conference on Computer-aided design*, 2024.

[26] Z. Shi, M. Li, S. Khan, L. Wang, N. Wang, Y. Huang, and Q. Xu, "Deeptpi: Test point insertion with deep reinforcement learning," in *2022 IEEE International Test Conference (ITC)*. IEEE, 2022, pp. 194–203.

[27] A. Biere, "Aiger (aiger is a format, library and set of utilities for and-inverter graphs (aigs))," https://fmv.jku.at/aiger/, 2006.

[28] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting a fresh look at combinational logic synthesis," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 532–535.

[29] R. K. Brayton, "The decomposition and factorization of boolean expressions," *International Symposium on Computer Architecture (ISCA 1982)*, pp. 49–54, 1982.

[30] J. Cortadella, "Timing-driven logic bi-decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 675–685, 2003.

[31] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean resubstitution with permissible functions and binary decision diagrams," in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1991, pp. 284–289.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[33] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for sat solvers," in *Theory and Applications of Satisfiability Testing – SAT 2016*. Springer, 2016, pp. 123–140.

[34] H. Haanpää, M. Järvisalo, P. Kaski, and I. Niemelä, "Hard satisfiable clause sets for benchmarking equivalence reasoning techniques," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 27–46, 2006.

[35] J. M. Dudek, K. S. Meel, and M. Y. Vardi, "The hard problems are almost everywhere for random cnf-xor formulas," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 600–606.

[36] A. Ling, D. P. Singh, and S. D. Brown, "Fpga technology mapping: a study of optimality," in *Proceedings of the 42nd ACM/IEEE Design Automation Conference*, 2005, pp. 427–432.

[37] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www. eecs. berkeley. edu/alanmi/abc*, vol. 17, 2007.

[38] M. J. Heule, M. Iser, M. Järvisalo, and M. Suda, "Proceedings of sat competition 2024: Solver, benchmark and proof checker descriptions," 2024.

[39] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "Cadical 2.0," in *International Conference on Computer Aided Verification*. Springer, 2024, pp. 133–152.

HeteroSVD: Efficient SVD Accelerator on Versal ACAP with Algorithm-Hardware Co-Design

Xinya Luan[1], Zhe Lin[2†], Kai Shi[1], Jianwang Zhai[1], and Kang Zhao[1†],
[1]Beijing University of Posts and Telecommunications [2]Sun Yat-sen University
{luanxinya, shikai, zhaijw, zhaokang}@bupt.edu.cn, linzh235@mail.sysu.edu.cn

***Abstract*—Singular value decomposition (SVD) is a matrix factorization technique widely used in signal processing and recommendation systems, etc. In general, the time complexity of SVD algorithms is cubic to the problem size, making SVD algorithms difficult to meet stringent performance requirements in real-time. However, existing FPGA and GPU solutions fall short of jointly optimizing latency, throughput, and power consumption. To settle this issue, this paper proposes HeteroSVD, a heterogeneous reconfigurable accelerator for SVD computation on the Versal ACAP platform. HeteroSVD introduces a system-level SVD decomposition mechanism and proposes an algorithm-hardware co-design method to optimize SVD ordering jointly and AI engine (AIE)-centric dataflow and placement with Versal. Furthermore, in order to improve the quality of results (QoR) and facilitate micro-architecture selection, we introduce an automatic optimization framework that performs accurate performance modeling and fast design space exploration. Experiment results demonstrate that HeteroSVD reduces the latency by 1.98× over existing FPGA accelerators and outperforms GPU solutions with an improvement of up to 7.22× in latency, 1.77× in throughput, and 13.18× in energy efficiency.**

I. Introduction

In linear algebra, singular value decomposition (SVD) is a commonly used approach for identifying singular values and constructing a set of singular vectors or matrices from a given matrix. It is widely adopted in applications such as wireless communication [1]–[3], recommendation system [4], [5], etc., where data approximation, compression, and denoising are indispensable. Despite its importance and popularity in data processing, SVD is highly computationally intensive. For example, the Hestenes-Jacobi decomposition method, one of the mainstream SVD algorithms, conducts orthogonalization transformation, namely, multiple vector multiplications, for each pair of column vectors, in a single iteration. This leads to $N \times (N-1)/2$ rounds of vector multiplication computation in an iteration for an input matrix with N columns. To make matters worse, SVD algorithms usually demand multiple iterations until results finally converge. The high computation cost of SVD makes efficient implementation extremely difficult, especially under rigorous real-time requirements.

To boost the performance of real-time SVD processing, researchers have presented SVD accelerator architectures based on field-programmable gate array (FPGA) or graphics processing unit (GPU) platforms. FPGAs [6]–[8] are featured with programmable logic and wirings that can realize any boolean function via hardware reconfiguration. As a result, using FPGAs as the substrate, researchers can take advantage of a number of flexible logic blocks and wirings to tailor the control and data paths of SVD implementation, finally achieving low latency per matrix transformation. However, the scarcity of on-chip memory hinders FPGA solutions from achieving high throughput through effective parallelism. Conversely, GPUs [9]–[13] have a large number of well-constructed computation kernels and this facilitates the simultaneous processing of a set of data. Therefore, researchers investigate batch SVD implementation on GPU and demonstrate higher throughput compared with FPGA design. Nevertheless, this is accompanied by high power consumption. Besides this, GPU solutions incur high latency in computing SVD for single and small-scale matrices since the computation kernels are mostly underutilized in this case. In summary, the existing solutions for efficient SVD implementation fail to jointly optimize latency, throughput, and power consumption due to the high computational intensity of SVD and the inherent limitations of the target platforms.

Recently, AMD/Xilinx has introduced the Versal ACAP architecture, which integrates AI engines (AIEs), programmable logic (PL), and processor systems (PS) to address the limitations of GPUs and FPGAs. The AIEs are optimized for vector and matrix multiplication, allowing for high performance and low power consumption. However, due to the complexity of Versal ACAP, there are three key challenges for efficient SVD acceleration on Versal ACAP. First, the problem size of SVD often exceeds the computational power of a single AIE kernel, necessitating careful partitioning of the input matrix for effective processing. Second, as the usage of AIEs increases with the growing design scale, the limited communication bandwidth between AIEs and PL may incur a bottleneck. Lastly, the versatility of AIEs and the flexibility of PL give rise to a vast design space, making it non-trivial to identify the optimal configurations of micro-architecture that can offer high performance and energy efficiency.

To address these challenges, in this work, we propose an acceleration paradigm of SVD on the heterogeneous computing platform Versal ACAP, which is named HeteroSVD. HeteroSVD constructs the hardware of SVD with a specific focus on the system-level optimization through effectively harnessing the AIE and PL resources. It presents the techniques of workload allocation, communication optimization, and algorithm-hardware co-design. To go one step further, HeteroSVD presents an automatic optimization framework to take into account the effect of various micro-architecture parameters, which helps to rapidly identify the optimal hardware solutions under different problem sizes. The main contributions are as follows:

- We present HeteroSVD[‡], a high-performance and scalable hardware architecture of SVD on the Versal ACAP. To the best of our knowledge, this is the first work that seeks to optimize SVD on this novel heterogeneous computing platform.
- We propose a co-design methodology that jointly optimizes the SVD ordering from an algorithm perspective and the AIE-centric dataflow and placement from a hardware perspective, which boosts the overall performance and energy efficiency.
- We introduce an automatic design optimization framework with a performance analysis model and an efficient design space exploration engine, which navigates the rapid tuning of micro-architecture parameters and ultimately leads to a fast exploration of optimal designs given different problem sizes.

II. Background

A. Block Hestenes-Jacobi Algorithm for SVD

The SVD problem can be factorized as:

$$\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times n}\boldsymbol{\Sigma}_{n\times n}\boldsymbol{V}^{\boldsymbol{T}}_{n\times n}, \tag{1}$$

†Corresponding author.
‡Our method is open-source at https://github.com/zhaokang-lab/HeteroSVD.

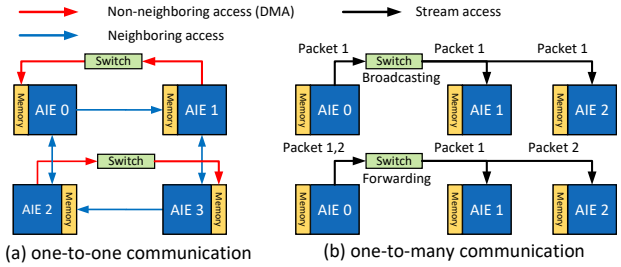


Fig. 1 Various AIE communication mechanisms.

where both U and V are unitary matrices and Σ is a diagonal matrix, and the diagonal elements of Σ are called singular value of A. The key steps of Hestenes-Jacobi [14] denoted as orthogonalization:

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{AV} = \boldsymbol{U\Sigma}, \\ \boldsymbol{V} &= \boldsymbol{J_1 J_2 J_3 J_4} \cdots . \end{aligned} \tag{2}$$

where the rotation matrix [15] $\boldsymbol{J}$ comprised of c and s is shown as:

$$[\boldsymbol{B}_i, \boldsymbol{B}_j] = [\boldsymbol{A}_i, \boldsymbol{A}_j] \cdot \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, where\ \boldsymbol{B}_i^T \cdot \boldsymbol{B}_j = \boldsymbol{0}, \tag{3}$$

$$c = \frac{1}{\sqrt{1+t^2}}, \text{ and } \mathrm{s} = a_i^T a_j \frac{tc}{|a_i^T a_j|}, \tag{4}$$

$$t = \frac{\mathrm{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}, \text{ and } \tau = \frac{a_j^T a_j - a_i^T a_i}{2|a_i^T a_j|}. \tag{5}$$

After multiple orthogonalizations for all column pairs, the convergence rate is less than the specified precision:

$$\frac{\boldsymbol{B}_i^T \boldsymbol{B}_j}{\sqrt{(\boldsymbol{B}_i^T \boldsymbol{B}_i)(\boldsymbol{B}_j^T \boldsymbol{B}_j)}} < precision. \tag{6}$$

Finally, normalization is performed to calculate Σ and U by:

$$\boldsymbol{\Sigma} = \sqrt{\boldsymbol{B^T B}}, \quad \boldsymbol{U} = \boldsymbol{B}/\boldsymbol{\Sigma}. \tag{7}$$

To solve the SVD of large matrices with limited resources, the block Jacobi algorithm is proposed. It divides a large-scale matrix into manageable sub-matrix blocks, enumerates block pairs, and further orthogonalizes all column pairs in a specific sequence such as ring [16] and round-robin [17].

B. Versal ACAP

Versal ACAP is a high-performance heterogeneous computation architecture consisting of central processing units (CPUs), PL, and AIEs, which are connected by a high-bandwidth network on chip (NoC). Each AIE comprises an AI-oriented computation core, a tile interconnect module (denoted as a switch), and a memory module with four banks of 8KB. Additionally, there are multiple interfaces called PLIO between the PL and AIEs, supporting a bandwidth of 24 GB/s from AIE to PL and 32 GB/s in the reverse direction.

Fig. 1 illustrates different mechanisms of data movement between AIEs. Fig. 1(a) depicts one-to-one communication where all AIEs are placed next to each other. The blue arrows indicate that the source AIE can access the memory of the target AIE, because the computation core and memory are adjacent, with no other computation cores or memory blocking them in between, thus considering them neighbors. In contrast, non-neighboring AIEs' communication is supported by the direct memory access (DMA) mechanism, which requires twice the memory resources and has a lower data transmission rate. Fig. 1(b) depicts two types of one-to-many communication mechanisms, including statically broadcasting packets from one AIE to several specific destinations and dynamically forwarding packets to different destinations according to the packet header. Both utilize stream access for communication, which has a speed comparable to that of DMA.

Algorithm 1: SVD Algorithm in HeteroSVD.

Input: $A_{m\times n}$
Output: U,Σ

1 $convergence_rate = 0$;
2 **while** $convergence_rate > precision$ **do**
3 // AIE-PL communication
4 **for** *each block pair*(A_u, A_v) *in* $[A_1, A_2, \cdots, A_p]$ **do**
5 send A_u, A_v to orth-AIEs;
6 // AIE interconnection
7 **for** *each column pair*(A_i, A_j) *in block pair*(A_u, A_v) **do**
8 // AIE computation of orthogonalization (orth-AIE)
9 calculate $conv(A_i, A_j)$;
10 update $conv(A_u, A_v)$;
11 calculate t, τ, c, s;
12 update(A_i, A_j);
13 **end**
14 receive A_u, A_v, $conv_(A_u, A_v)$ from orth-AIEs;
15 update convergence_rate;
16 **end**
17 **end**
18 // AIE-PL communication
19 **for** *each block*(A_i) *in* $[A_1, A_2, \cdots, A_p]$ **do**
20 send A_i to norm-AIEs;
21 **for** *each column*(A_j) *in block*(A_i) **do**
22 // AIE computation of normalization (norm-AIE)
23 calculate Σ_j, U_j;
24 **end**
25 receive Σ_i, U_i from norm-AIEs;
26 **end**

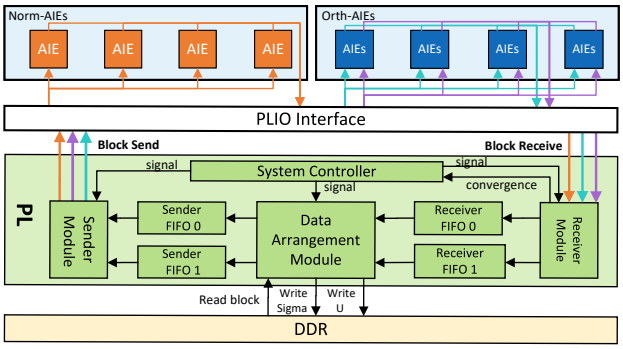


Fig. 2 HeteroSVD architecture.

III. HeteroSVD Accelerator Design

A. Overview

SVD Decomposition. Algorithm 1 provides the SVD algorithm that is restructured for HeteroSVD. The key idea behind Algorithm 1 is to split a large-scale matrix into smaller blocks, each of which is further divided into a set of column vectors. Subsequently, column pairs are constructed by grouping any two column vectors from two blocks, which are then sent to the AIE array for result computation. The AIEs are responsible for performing two core types of operations on each pair of column vectors, namely, orthogonalization (lines 7-13) and normalization (lines 21-24).

Overall Architecture. HeteroSVD orchestrates PL and AIE to facilitate the system-level hardware optimization, as depicted in Fig. 2. To start with, the data arrangement module reads the holistic matrix $A_{m\times n}$ with the size of $m \times n$ from DDR and splits $A_{m\times n}$ into smaller ones with the size of $m \times k$. Subsequently, it reorders the incoming data from DDR or the receiver FIFOs in a round-robin manner and passes block pairs to two sender FIFOs respectively. Next, the sender module leverages the dynamic forwarding mechanism and packs each column into packets with headers to ensure correct routing to the respective AIEs. After AIEs finish processing, the receiver

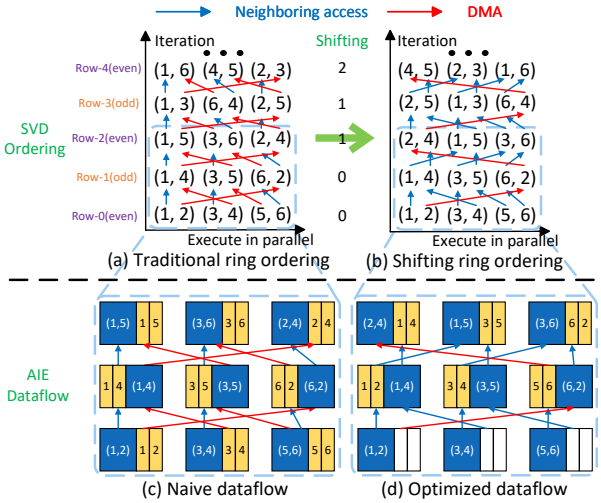


Fig. 3 The comparison of ordering and dataflow between traditional ring ordering and HeteroSVD method.

module reunites packets from AIEs, sorts them into columns, stores them in the corresponding receiver FIFOs, and then sends the deduced convergence rate to the system module. If the convergence rate is less than the user-specified precision, the system module will terminate the orthogonalization stage and proceed into the normalization stage. Finally, the data arrangement module stores the results of Σ and U into DDR and releases a signal of completion.

Note that naive SVD computation may lead to a complicated data transmission scenario between AIEs, which can severely impede communication efficiency. In light of this problem, we propose an algorithm-hardware co-design strategy to jointly optimize both the SVD ordering from an algorithm perspective and the AIE-centric dataflow from a hardware perspective. Finally, we present the method of AIE placement that instantiates the SVD on Versal ACAP.

B. Algorithm-Hardware Co-Design

SVD ordering refers to the practice of determining the processing order of orthogonalizing different column pairs. Unfortunately, the traditional SVD ordering, such as the widely used ring ordering [16], does not fit in well with AIE's inherent characteristics. Specifically, the AIEs are arranged in rows in the AIE array, with neighboring rows having different topologies: *in even rows, each computation core is located on the left side of its internal memory, whereas in odd rows, the relative position of each computation core and its memory gets reversed*. The traditional ordering of SVD algorithms maintains a monolithic data movement pattern across different iterations, which is inconsistent with the AIE alignment in different rows. This discrepancy makes it necessary to use a large number of DMA transmissions, a type of less efficient data communication between non-neighboring AIEs that leads to reduced transmission speed and increased memory usage, as discussed in Section II-B. Fig. 3 (a) and (c) illustrate this issue with matrix $A_{m\times 6}$ as an example, where the number in parentheses denotes the column index, the red arrow indicates DMA transmission and the blue represents neighboring access.

HeteroSVD proposes to co-optimize the SVD algorithm and the Versal AIE array, which can significantly reduce the number of DMA transmissions from $2k(k-1)$ to $2(k-1)$ for a matrix with the size of $m \times 2k$, as shown in Fig. 3 (b) and (d). Specifically, we propose the shifting ring ordering method to augment the SVD algorithm, and the AIE dataflow that reduces communication and memory usage.

Shifting Ring Ordering. To accommodate the AIE array's asymmetric characteristics, we propose the shifting ring ordering mecha-

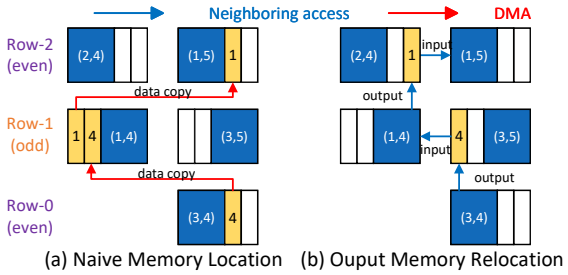


Fig. 4 Dataflow Comparison between different memory location strategies.

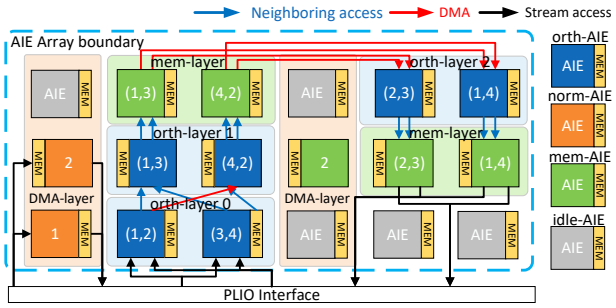


Fig. 5 AIE placement with three types of AIE.

nism for SVD. The main idea is to dynamically change the pattern of data movements in an effort to align with the topology of each AIE row, as depicted in Fig. 3 (b). For row-i, we shift the column pairs to the right by a step size of $\lfloor \frac{i}{2} \rfloor$ in a cyclic manner. For instance, in Fig. 3 (b), all the column pairs in row-2 are shifted by one step cyclically: the right-most column pair (2,4) becomes the left-most, the left-most column pair (1,5) shifts to the middle, and so on. By ordering in this way, the original straight and leftward movements from odd rows to even rows are respectively transformed into rightward and straight movements, while the movements from even rows to odd rows remain unchanged. This method synchronizes the data movement with the topologies of the AIE array.

AIE-Centric Dataflow. To optimize the inter-AIE communication, we relocate the position of output storage to convert the non-neighboring DMA transmission into direct neighboring access. As illustrated in Fig. 4 (a), using the naive memory configuration, each AIE's output is stored in its own memory, necessitating DMA to copy the data due to the lack of a direct interface between the source AIE and target AIE. This means that the orthogonalization operation requires a memory block twice the size of the result to store it, and an expensive DMA communication is engaged as well. To address this issue, we subtly assign each AIE's output to the subsequent AIE's memory in the next row as shown in Fig. 4 (b). This allows the target AIE to communicate with the source AIE through two neighboring accesses via the intermediate AIE memory. Untimately, except for the long-distance communication between the first and the last AIE columns, all data movements from the shifting ring ordering can be achieved with neighboring direct access, as shown in Fig. 3 (d).

C. AIE Placement

AIE placement means the instantiation of AIEs and the corresponding interconnect to implement the SVD algorithm on hardware. However, this problem is challenging. The complete SVD algorithm comprises two phases: orthogonalization and normalization. It is essential to wisely assign AIEs to complete both phases. In particular, the shifting ring ordering for orthogonalizing matrix $A_{m\times 2k}$ requires an array of $(2k-1)\times k$ AIEs, which is incompatible with the 8×50

size of the AIE array on our target board. To this end, we propose an AIE placement strategy, and Fig. 5 illustrate it with matrix $A_{m\times 4}$ as an example.

We define three types of AIE, namely orth-AIE, norm-AIE, and mem-AIE, which are designed for orthogonalization, normalization, and the storage of intermediate data, respectively. For the orthogonalization stage, we partition the shifting ring ordering of size of $(2k-1)\times k$ into $2k-1$ orth-layers, each consisting of k orth-AIEs. Then we place the $2k-1$ orth-layers in a row-wise manner. Notably, at the boundary of the AIE array, namely the first row and the last row, we place a mem-layer instead of an orth-layer as the maximum row limitation of the AIE array prevents a subsequent row from storing the current layer's output. Consequently, we choose another k columns to continue placing the remaining orth-layers, albeit at the expense of some unavoidable DMA transmissions. Moreover, we assign the columns on both sides of the orth-layers as DMA-layers, allocating mem-AIEs to store the copied data due to the DMA mechanism, This is necessary because the orth-AIEs with DMA have already stored data from the previous orth-layers and lack sufficient memory to accommodate the additional copied data. For the normalization stage, we allocate the norm-AIE within the remaining idle-AIE.

We further establish a dynamic forwarding rule to ensure that the block pair from PL is correctly routed to the placed AIEs. Specifically, we designate that odd and even columns are sourced from different blocks within the block pair, utilizing four PLIOs for their respective transmission. Different columns from a single block are routed to different AIEs as shown in Fig. 5. For the norm-AIE, we only use two PLIOs, as the two blocks in the block pair are transmitted sequentially between the PL and AIE.

IV. Automatic Design Optimization Framework

In this section, we first describe the micro-architecture parameters derived from two levels of parallelism. After that, we discuss our performance modeling and design space exploration (DSE) methods.

A. Micro-Architecture Parameters

HeteroSVD has two levels of parallelism that expand its design space, as shown in Fig. 6. The first parallelism is AIE-level parallelism (denoted as P_{eng}), which determines the number of AIEs used in parallel to execute the SVD computation for a single task. The second parallelism is task-level parallelism (denoted as P_{task}), which represents the number of tasks processed simultaneously in the computing system. These two parallelism factors, together with the given operating frequency of PL, determine the system-level performance and guide the selection of parameters in the HeteroSVD design, making them first-order micro-architecture parameters, as listed in TABLE I. Additionally, second-order micro-architecture parameters include the number of PLIOs and the counts of AIEs designated as orth-AIE, norm-AIE, and mem-AIE.

In summary, by specifying P_{eng}, P_{task}, and PL frequency, we can determine the architecture of HeteroSVD. However, searching for the optimal and feasible HeteroSVD architecture by enumerating all possible micro-architecture parameters requires the full invocation of the EDA front-end, back-end, and cycle-accurate simulation flow, which is extremely time-consuming. For instance, running a single design point for an SVD of a 128×128 matrix would take more than seven hours. Considering a design space of (P_{eng}, P_{task}) that leads to 286 design points, the cost of searching for optimal architecture is unaffordable. Therefore, HeteroSVD proposes an automation design approach with a performance model and a design space exploration flow to construct the optimal HeteroSVD structure of a given problem size within minutes.

TABLE I HeteroSVD architecture parameters at different hierarchies.

Hierarchy	Parameter	Range or Value
First order	P_{eng}	$n \in [1, 11]$
	P_{task}	$k \in [1, 26]$
	PL frequency (MHz)	-
Second order	the number of orth-AIE	$n(2n-1)k$
	the number of norm-AIE	nk
	the number of mem-AIE	-
	the number of PLIO	$6k$

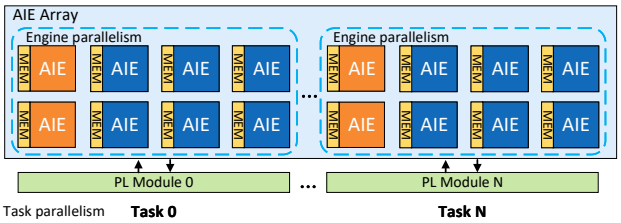


Fig. 6 Engine parallelism and task parallelism.

B. Performance Model

To efficiently evaluate the performance with different AIE-level and task-level parallelisms, we first present a performance model to estimate the latency and throughput of the overall computing system. The complete pipeline of the orthogonalization is shown in Fig. 7. This pipeline can be further decomposed into data sending (Tx) from PL to AIE, data receiving (Rx) of PL from AIE, and execution of orthogonalization (orth-AIEs). And since orth-AIEs have two inputs and output, we use left(L) and right(R) to distinguish them.

The execution time of Tx and Rx is determined by PL frequency and PL bandwidth:

$$t_{Tx,Rx} = \frac{databits}{bandwidth \cdot frequency}, \tag{8}$$

and the AIE time is estimated by the AIE simulator in advance. Besides, there are three main types of latency.

AIE-wait latency ($t_{AIEwait}$) arises from the imbalance between the execution time of AIE and the time taken for data transmission:

$$t_{AIEwait} = max(t_{orth} - P_{eng} \cdot t_{Tx}, 0). \tag{9}$$

Specifically, when multiple packets are transmitted serially to different AIEs through a single PLIO, the incoming data must wait for the completion of AIE's last execution.

Algorithm latency (t_{algo}) results from the data dependency between the second Rx-R and the first Tx-R in the round-robin algorithm:

$$t_{algo} = t_{Tx} + t_{AIEwait}. \tag{10}$$

The lack of sending data causes data-wait latency ($t_{datawait}$):

$$\begin{aligned} t_{datawait} = max(-(num-1)\cdot(t_2 + t_{Tx}) \\ + t_{AIEs} + t_{Rx} + t_{algo}, 0), \end{aligned} \tag{11}$$

where num and t_{AIEs} represent the number of block data pairs and the total AIEs time between Tx and Rx.

Beyond these latencies, there is also latency caused by HLS (t_{hls}) and DDR (t_{DDR}). HLS [18] incurs additional cycles when switching between loops, thus, t_{hls} can be calculated based on the loop structure in the code. The inability to load block pairs from DDR simultaneously results in significant latency during the first iteration:

$$t_{DDR} = num \cdot t_{Tx}. \tag{12}$$

Then we can calculate the time required for one iteration:

$$\begin{aligned} t_{blocks} &= num \cdot (t_{Tx} + t_{AIEwait}) + t_{algo} + t_{datawait}, \\ t_{iter} &= (num-1)\cdot t_{blocks} + AIE_{total} + t_{Rx}. \end{aligned} \tag{13}$$

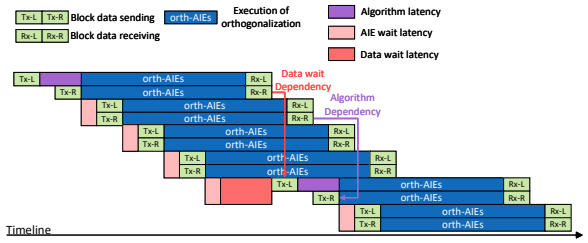


Fig. 7 HeteroSVD pipeline model at orthogonalization stage

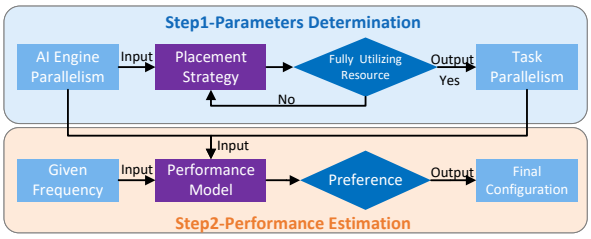


Fig. 8 DSE flow with two stages.

Norm-AIE's pipeline resembles orth-AIE's but with single input/output and no algorithm latency. We define the normalization time as t_{norm}. The task time can be computed by summing the execution times of orthogonalization, normalization, and additional latency. The system time can then be derived using:

$$\begin{aligned} t_{task} &= t_{DDR} + ITER \cdot t_{iter} + t_{norm} + t_{hls}, \\ t_{sys} &= \lceil \frac{num_{task}}{P_{task}} \rceil \cdot t_{task}, \end{aligned} \tag{14}$$

where ITER represents the specified iterations and num_{task} represents the number of tasks.

C. Design Space Exploration (DSE)

Problem Formulation. In general, the goal of our architecture optimization framework is to increase the performance of the SVD computing system. Specifically, given the size of the matrix ($M \times N$) and batch size (B), we need to properly select the first-order parameters listed in TABLE I, i.e., the parallelism factor of AIEs (P_{eng}) and tasks (P_{task}), and the practical frequency ($Freq$) of PL, while keeping all types of resources, including AIE kernel, PLIO, BRAM and URAM, under their budgets. To this end, we formally define our problem of design space exploration as follows:

$$\begin{aligned} \min \quad & runtime(P_{eng}, P_{task}, Freq), \\ \text{s.t.} \quad & Resource_i(P_{eng}, P_{task}), \leq C_i \\ & i \in \{AIE,\ PLIO, BRAM, URAM\}. \end{aligned} \tag{15}$$

DSE Flow. Fig. 8 illustrates the design exploration flow to search for the optimal HeteroSVD architecture. In the first stage, we enumerate the engine parallelism and maximize task parallelism by fully utilizing resource according to our placement strategy. Equation (16) summarized all constraints C_i, where $i \in \{AIE, PLIO, BRAM, URAM\}$, represent the number of hardware resource limits. The actual usage of AIE resource, num_{orth}, num_{norm}, num_{mem}, and num_{PLIO}, can be determined after placement while the resource usage of PL memory, including num_{BRAM} and num_{URAM} can be estimated by the array size of PL design. In the second stage, with engine parallelism, task parallelism, and the user-specified frequency as input, we utilize our performance model to estimate the overall computing system's performance. Furthermore, We determine the optimal HeteroSVD architecture based on the demand of latency or throughput.

$$\begin{aligned} num_{orth} + num_{norm} + num_{mem} &\leq C_{AIE}, \\ num_{PLIO} &\leq C_{PLIO}, \\ num_{BRAM} &\leq C_{BRAM}, \\ num_{URAM} &\leq C_{URAM}. \end{aligned} \tag{16}$$

V. Experiment Results

A. Experimental Setup

Our experiments are conducted on AMD/Xilinx Versal AI Core Series VCK190 [19] evaluation board. We use the Vitis 2023.2 toolkit to build up the whole HeteroSVD acceleration system. The AIEs operate at 1.25 GHz, while the frequency of the PL design depends on the specific system configurations. We use the AMD BEAM tool [20], to measure the exact power of the system while running on the VCK190 board. We compare our SVD acceleration system with the SOTA SVD accelerators implemented on AMD/Xilinx XC7V690T FPGA and GeForce RTX 3090 GPU.

B. Performance and Energy Efficiency

We compare HeteroSVD, with the state-of-the-art high-performance hardware implementations on FPGA [6] and GPU [11], both of which offer better latency or throughput compared to other implementations on the same platform. To enable a fair comparison, we perform six iterations per matrix and configure the FPGA accelerator to its maximum task parallelism at an achievable peak frequency of 200 MHz. Generally speaking, the GPUs perform better with large-scale data, while the FPGAs are featured with low-latency data processing. TABLE II presents the latency comparison and resource utilization between HeteroSVD and FPGA. The experimental results show that the HeteroSVD solution supports more flexible architecture and significantly outperforms the FPGA solution regarding latency in all matrix sizes, achieving 1.27× – 1.98× speedup over FPGA. Meanwhile, HeteroSVD consumes fewer computational resources than FPGA, indicating that HeteroSVD has greater potential for parallelism in handling multiple tasks.

Furthermore, we evaluate the latency, throughput, and energy efficiency for batch data processing of SVD between HeteroSVD and GPU as listed in TABLE III. We obtain the optimal micro-architecture configuration from HeteroSVD's DSE flow. In addition, we perform multiple iterations of SVD until the results converge at a rate of 10^{-6} for both HeteroSVD and GPU solution. HeteroSVD achieves 1.15× – 7.22× latency speedup over GPU for small matrix. Regarding throughput, HeteroSVD shows up to 1.77× speedup with a small matrix size. However, as the matrix size increases, the GPU solution outperforms our HeteroSVD solution in throughput. Fig. 9 illustrates the underlying mechanism. As the design size increases, the GPU solution exhibits higher utilization of its computation cores and memory, significantly boosting its performance. In contrast, due to limitations in PL memory, HeteroSVD experiences reduced task parallelism, leading to lower overall utilization. Moreover, larger design sizes increase the complexity of the PL, which reduces the maximum achievable frequency and further diminishes HeteroSVD's latency and throughput. Therefore, with adequate RAM resources and optimized operating frequency, we believe that HeteroSVD has the potential to outperform GPU solutions with similar core utilization.

As for energy efficiency, HeteroSVD demonstrates 4.36× – 13.18× improvement over GPU solutions. In summary, the tremendous gains justify that HeteroSVD is a highly competitive alternative to both FPGA and GPU in achieving high-performance yet energy-efficient SVD accelerators.

TABLE II Latency comparison and resource utilization between HeteroSVD and FPGA.

Matrix Size	FPGA [6]				HeteroSVD				HeteroSVD
	Latency(s)	LUT	BRAM	DSP	Latency(s)	LUT	URAM	AIE	Speedup
128×128	0.0014	212K (30.6%)	519.5 (31.4%)	1602(44.5%)	0.0011	15.1K(1.68%)	4(0.86%)	128(32%)	1.27×
256×256	0.0113				0.0057	15.2K (1.69%)	20 (4.32%)		1.98×
512×512	0.0829				0.0435	15.5K (1.72%)	64 (13.82%)		1.90×
1024×1024	0.6119				0.3415	15.7K (1.75%)	244 (52.70%)		1.79×

TABLE III Latency, throughput and energy efficiency comparison between HeteroSVD and GPU.

Matrix Size	GPU [11] (270W)			HeteroSVD (<39W)			HeteroSVD vs GPU		
	Latency (s)	Throughput (Tasks/s)	Energy Efficiency (Tasks/s/Watt)	Latency (s)	Throughput (Tasks/s)	Energy Efficiency (Tasks/s/Watt)	Latency Speedup	Throughput Speedup	EE Gain
128×128	0.0166	1351.35	5.005	0.0023	2389.69	65.940	7.22×	1.77×	13.18×
256×256	0.0429	217.39	0.805	0.0130	239.48	6.251	3.30×	1.10×	7.76×
512×512	0.1237	27.55	0.102	0.1076	24.42	0.663	1.15×	0.89×	6.50×
1024×1024	0.6857	3.52	0.013	0.7937	1.27	0.057	0.86×	0.36×	4.36×

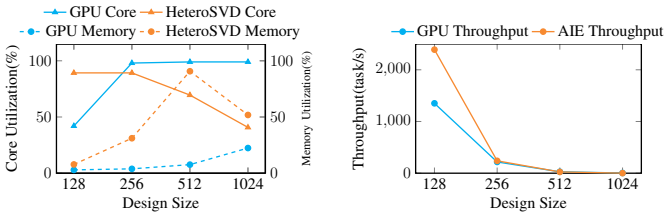


Fig. 9 The throughput and utilization of core and memory comparison with different design sizes between GPU and HeteroSVD

TABLE IV The processing time (ms) of SVD single iteration from the performance model and on-board measurement under a frequency of 208.3 MHz.

Matrix Size	P_{eng}	On-board Meas.	Perf. Model	Error
128×128	2	0.993	1.022	2.92%
256×256		6.151	6.338	3.03%
512×512		43.229	42.020	2.80%
128×128	4	0.395	0.391	1.03%
256×256		2.853	2.806	1.66%
512×512		21.584	21.265	1.48%
128×128	8	0.214	0.219	2.57%
256×256		1.475	1.476	0.05%
512×512		10.965	10.903	0.56%

TABLE V The processing time (ms) of SVD with one iteration from the performance model and on-board measurement under various application scenarios.

Matrix Size	Batch	Freq.(MHz)	P_{eng}	P_{task}	On-board Meas.	Perf. Model	Error
128×128	1	450	8	1	0.357	0.384	7.52%
256×256	1	420	8	1	1.202	1.120	6.82%
512×512	1	350	8	1	7.815	7.510	3.90%
1024×1024	1	310	8	1	58.885	58.255	1.02%
128×128	100	330	4	9	6.099	6.412	5.12%
256×256	100	310	4	9	27.836	26.623	4.36%
512×512	100	310	4	7	238.002	224.301	5.76%
1024×1024	100	310	8	1	5872.181	5878.970	0.12%

C. Effectiveness of DSE

Accuracy of Performance Model. The accuracy of the performance model is key to developing a high-quality design space exploration approach. Therefore, in this experiment, we first evaluate the effectiveness of our performance model. We apply different factors of engine parallelism and the input matrix size and examine the processing time of HeteroSVD in terms of a single iteration. The frequency of the PL side is fixed at 208.3 MHz. TABLE IV shows the results of the HeteroSVD performance model and the on-board measurement. We can observe that the maximum error of our model is 3.03% and the average error is 1.78%, which shows the efficacy of our performance model in the evaluation of latency.

To further validate the generalization ability of the performance model across different application scenarios, we use the HeteroSVD DSE approach to identify the optimal configuration with minimum execution time. Then, we get the latency of processing a single matrix and the throughput of processing 100 matrices from both the model and on-board measurement. As shown in TABLE V, our performance model demonstrates a maximum error of 7.52% and an average error of 4.33%. This highly accurate performance model ensures the effectiveness of our DSE flow with design point enumeration.

TABLE VI Latency(ms), throughput(task/s) and power(W) comparison between different design points. The matrix size is 256×256 and the PL frequency is reported in 208.3MHz

P_{eng}	P_{task}	AIE	URAM	Latency	Throughput	Power
2	26	293(73.25%)	416(89.85%)	35.689	707.501	44.16
4	9	357(89.25%)	144(31.10%)	19.303	508.436	34.63
6	4	366(91.50%)	120(25.92%)	13.117	306.876	30.79
8	2	322(80.50%)	32(6.91%)	9.247	219.257	26.06

Discussion on DSE Results. TABLE VI evaluates how micro-architecture parameters influence latency, throughput, and power consumption of SVD implementation, with each design point executing six iterations. Increasing P_{eng} allows for more AIEs to be allocated to a single SVD, effectively reducing latency. However, a design point with high P_{eng} is not suitable for scenarios with strict throughput requirements, as it restricts task parallelism. Conversely, higher task parallelism (P_{task}) improves throughput but increases URAM usage, leading to significant power consumption. In summary, we favor lower P_{eng} and higher P_{task} for high throughput, while the opposite configuration is preferable for minimizing latency. Moreover, a comprehensive consideration of micro-architecture parameters is essential for effectively managing power consumption.

VI. Conclusion

In this paper, we propose HeteroSVD, an optimized hardware accelerator of block-Jacobi SVD based on the novel heterogeneous computing platform, Versal ACAP. For HeteroSVD accelerator design, we decompose the computation of SVD and propose the algorithm-hardware co-design for SVD ordering, AIE dataflow, and AIE placement. To further optimize the design with a set of micro-architecture parameters, we introduce a design space exploration flow to expedite the search for the best hardware solutions at different problem sizes.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB2901100), the National Natural Science Foundation of China (No. 62404257, 62404021), the Beijing Natural Science Foundation (No. 4244107, QY24216, QY24204), and the Guangdong Basic and Applied Basic Research Foundation (2023A1515110769).

REFERENCES

[1] J. Löfgren, S. Mehmood, N. Khan, B. Masood, M. I. Z. Awan, I. Khan, N. A. Chisty, and P. Nilsson, "Hardware implementation of an SVD based MIMO OFDM channel estimator," in *2009 NORCHIP*, 2009, pp. 1–4.

[2] M. V. Athi, S. R. Zekavat, and A. A. Struthers, "Real-Time Signal Processing of Massive Sensor Arrays via a Parallel Fast Converging SVD Algorithm: Latency, Throughput, and Resource Analysis," *IEEE Sensors Journal*, vol. 16, no. 8, pp. 2519–2526, 2016.

[3] G. Lebrun, J. Gao, and M. Faulkner, "MIMO transmission over a time-varying channel using SVD," *IEEE Transactions on Wireless Communications*, vol. 4, no. 2, pp. 757–764, 2005.

[4] W. Wu, L. Zhao, Q. Wu, X. Wang, T. Tian, and X. Jin, "An SSD-Based Accelerator for Singular Value Decomposition Recommendation Algorithm on Edge," in *IEEE High Performance Extreme Computing Conference (HPEC)*, 2022, pp. 1–5.

[5] X. Guan, C.-T. Li, and Y. Guan, "Matrix factorization with rating completion: An enhanced SVD model for collaborative filtering recommender systems," *IEEE Access*, vol. 5, pp. 27 668–27 678, 2017.

[6] T. Hu, X. Li, X. Yu, S. Ren, L. Yan, X. Bai, Z. Xu, and S. Zhu, "A Novel Fully Hardware-Implemented SVD Solver Based on Ultra-Parallel BCV Jacobi Algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5114–5118, 2022.

[7] X. Wang and J. Zambreno, "An FPGA Implementation of the Hestenes-Jacobi Algorithm for Singular Value Decomposition," in *IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, pp. 220–227.

[8] Y. Wang, J.-J. Lee, Y. Ding, and P. Li, "A Scalable FPGA Engine for Parallel Acceleration of Singular Value Decomposition," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 370–376.

[9] C. Toolkit, Cuda toolkit documentation, 2018. [Online]. Available: http://docs.nvidia.com/cuda/index.html

[10] R. Huang, T. Yu, S. Liu, X. Zhang, and Y. Zhao, "A Batched Jacobi SVD Algorithm on GPUs and Its Application to Quantum Lattice Systems," in *Parallel and Distributed Computing, Applications and Technologies*, H. Shen, Y. Sang, Y. Zhang, N. Xiao, H. R. Arabnia, G. Fox, A. Gupta, and M. Malek, Eds., 2022, pp. 69–80.

[11] J. Xiao, Y. Pang, Q. Xue, C. Shui, K. Meng, H. Ma, M. Li, X. Zhang, and G. Tan, "W-Cycle SVD: A Multilevel Algorithm for Batched SVD on GPUs," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–16.

[12] S. Lahabar and P. Narayanan, "Singular value decomposition on GPU using CUDA," in *IEEE international symposium on parallel & distributed processing*, 2009, pp. 1–10.

[13] W. H. Boukaram, G. Turkiyyah, H. Ltaief, and D. E. Keyes, "Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression," *Parallel Computing*, vol. 74, pp. 19–33, 2018.

[14] M. R. Hestenes, "Inversion of Matrices by Biorthogonalization and Related Results," *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, no. 1, pp. 51–90, 1958.

[15] N. L. Bihan and S. J. Sangwine, "Computing the SVD of a quaternion matrix," 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:18661147

[16] "A Parallel Ring Ordering Algorithm for Efficient One-Sided Jacobi SVD Computations," *Journal of Parallel and Distributed Computing*, vol. 42, no. 1, pp. 1–10, 1997.

[17] R. P. Brent and F. T. Luk, "The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 1, pp. 69–84, 1985.

[18] AMD Vitis High-Level Synthesis User Guide (UG1399). [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction

[19] AMD VCK190 board. [Online]. Available: https://china.xilinx.com/products/boards-and-kits/vck190.html

[20] AMD BEAM Tool for VCK190 Evaluation Kit. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/2549186622/Power+Management+-+Getting+Started#Tools-to-Estimate-and-Measure-Power

Mixed Structural Choice Operator: Enhancing Technology Mapping with Heterogeneous Representations

Zhang Hu[1], Hongyang Pan[2], Yinshui Xia[1], Lunyao Wang[1] and Zhufei Chu[1]*
[1]Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China
[2]School of Microelectronics, State Key Laboratory of Integrated Chips and System, Fudan University, Shanghai 200433, China
*Email: chuzhufei@nbu.edu.cn

***Abstract*—The independence of logic optimization and technology mapping poses a significant challenge in achieving high-quality synthesis results. Recent studies have improved optimization outcomes through collaborative optimization of multiple logic representations and have improved structural bias through structural choices. However, these methods still rely on technology-independent optimization and fail to truly resolve structural bias issues. This paper proposes a scalable and efficient framework based on *Mixed Structural Choices* (MCH). This is a novel heterogeneous mapping method that combines multiple logic representations with technology-aware optimization. MCH flexibly integrates different logic representations and stores candidates for various optimization strategies. By comprehensively evaluating the technology costs of these candidates, it enhances technology mapping and addresses structural bias issues in logic synthesis.**

Notably, the MCH-based lookup table (LUT) mapping algorithm set new records in the *EPFL Best Results Challenge* by combining the structural strengths of both And-Inverter Graph (AIG) and XOR-Majority Graph (XMG) logic representations. Additionally, MCH-based ASIC technology mapping achieves a 3.73% area and 8.94% delay reduction (balanced), 20.35% delay reduction (delay-oriented), and 21.02% area reduction (area-oriented), outperforming traditional structural choice methods. Furthermore, MCH-based logic optimization utilizes diverse structures to surpass local optima and achieve better results.

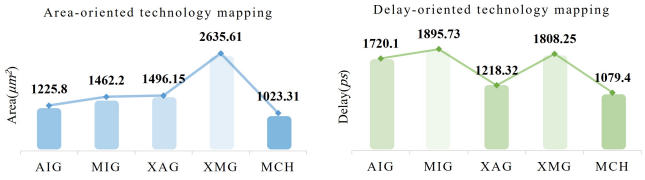
***Index Terms*—logic synthesis, technology mapping, structural bias, structural choices, heterogeneous representations.**

I. Introduction

Logic synthesis is one of the key steps in the *electronic design automation* (EDA) workflow. The process of logic synthesis mainly includes translation, technology-independent logic optimization, and technology mapping, aiming to convert RTL designs into mapped netlists with optimized power, performance, and area (PPA). Achieving high-quality synthesis results is a significant and challenging scientific task.

During the logic synthesis phase, the design representation evolves from truth tables, *sum of products* (SOPs), *binary decision diagrams* (BDDs), to *directed acyclic graphs* (DAGs), while the scale of the design gradually expands. Currently, mainstream logic representations in the logic synthesis process include *AND-Inverter graph* (AIG), *Majority-Inverter graph* (MIG) [1], *XOR-AND graph* (XAG) [2], and *XOR-Majority graph* (XMG) [3]. For different types of circuits, each representation has its unique advantages. For example, some novel logic representations perform better in arithmetic-intensive circuits [4]. In recent years, some studies have focused on exploring optimization methods and application for different logic representations [5–8]. Modern IC designs consist of complex functional modules that can be specifically optimized using different logic representations.

Recent studies have achieved better optimization results by combining different logic representations [9–12]. For example, the LSOracle framework [10, 11] introduces a heterogeneous synthesis approach. It partitions the circuit into segments optimized using different logic representations, which are later unified into a single representation for mapping. However, these methods are often limited by ***structural bias***, a well-known issue in logic synthesis [13]. They convert all representations into a single form for mapping, which compromises the structural advantages of other representations.



(a) Mapping area information (b) Mapping delay information

Fig. 1: Technology mapping for different logic representations

As design types and requirements become increasingly complex, the limitations of relying on a single logic representation are becoming more apparent. Fig. 1 illustrates the conversion of the "Max" circuit from the *EPFL benchmark suite* [14] into various logic representations, along with information on technology mapping using the *Arizona State Predictive PDK 7nm* (ASAP7) library [15]. Results indicate that, for delay-oriented mapping, using XAG yields better performance for this circuit, while for area-oriented mapping, AIG achieves superior results. The mapping results are strongly influenced by the circuit's structure. Therefore, an effective technology mapping method should be able to jointly evaluate different logic representations, enhancing adaptability across different synthesis scenarios to achieve superior mapped netlists. In fact, our method achieves this by utilizing *Mixed Structural Choices* (MCH) to effectively overcome the *structural limitations* of single logic representations, resulting in more competitive outcomes.

The issue of structural bias permeates all stages of logic synthesis. The root cause lies in the independence between logic optimization and technology mapping. This causes logic optimization may misalign with mapping, wasting resources and missing optimal structures. Some studies employ machine learning and heuristic approaches to explore better mapping structures [16–18]. Another promising approach is structural choices, which preserve different structural candidates during logic optimization to address structural bias [13, 19–21]. However, these methods are limited to searching for optimal mapping structures within the optimized graph. It is difficult to cope with the structural limitations of the graph when the optimization is close to a local optimum. On the other hand, existing structural choice methods rely on equivalence checks between technology-independent optimized networks, significantly limiting the diversity of candidates and reducing the efficiency of constructing choice networks. In recent studies, E-Syn utilize e-graphs to explore optimal structures [22]. However, its efficiency is relatively low when handling large-scale designs.

Fig. 2 illustrates the limitations of the traditional synthesis flow through a simple example. The original network is an AIG with 11 nodes. After technology-independent optimization, the number of AIG nodes is reduced to 8, but the mapping cost increases instead.

This work was supported in part by the NSFC under Grant No. 62274100, in part by the Key R&D Program of Zhejiang Province under Grant No. 2024C01111 and Ningbo City under Grant No. 2023Z071, and in part by the Zhejiang Provincial NSFC under Grant Nos. LDT23F0402 and LDT23F04021F04.

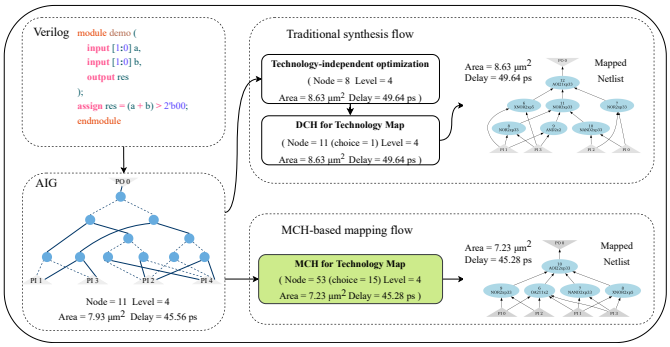


Fig. 2: Comparison of the MCH-based mapping flow with the traditional synthesis flow

Subsequently, traditional structural choice algorithms [13] (denoted as DCH in ABC [23]) still fail to effectively address the impact of structural limitations. This issue is particularly common in large-scale designs. Thus, achieving synergy between logic optimization and technology mapping remains a significant technical challenge.

Based on existing issues, this paper introduces the *Mixed Structural Choices* (MCH) operator. They are mixed networks incorporating heterogeneous logic representations, eliminating the constraints of relying on a single logic representation during logic synthesis. Unlike existing heterogeneous synthesis methods, MCH preserves the original logic structure through structural choices while incorporating new logic representations as candidates. Furthermore, to obtain diversified candidate structures, MCH introduces a multi-strategy structural choices algorithm. It combines node path classification with synthesis strategies targeting different objectives, preserving equivalence rather than performing local replacements before optimization, thereby retaining diverse optimization candidates. Finally, MCH integrates with technology mapping, using technology information to evaluate candidate logic representations and structures, thereby enhancing technology mapping.

The main contributions of this paper are as follows:

- The concept of MCH is proposed, enabling the simultaneous evaluation of different logic representations through choice networks.
- Multi-strategy structural choices differentiate critical paths and apply diverse optimization strategies, preserving varied logic representations and optimization candidates.
- Forms an MCH-based technology mapping method, which evaluates candidate logic representations and optimizations using technology information.
- In tests based on the EPFL benchmark suite, MCH-based *application specific integrated circuits* (ASIC) technology mapping achieved improvements across multiple metrics. MCH-based *field programmable gate array* (FPGA) technology mapping set new records in the *EPFL Best Results Challenge*. MCH-based graph mapping overcomes local optima, improving logic optimization and post-mapping results.

II. Background

A. Boolean Network

In logic synthesis, Boolean networks are typically represented as directed acyclic graphs. DAGs consist of logic nodes representing Boolean functions, *primary inputs* (PI), *primary outputs* (PO), and directed edges connecting these nodes. The level of node n is defined as the length of the longest path from any PI to node n. The level of a Boolean network is determined by the maximum level of its internal nodes. The output of a node n can connect to other nodes, known as the node's fanout, and the set of all fanout nodes forms the *transitive fanout* (TFO) cone. The *transitive fanin* (TFI) cone of a node n includes all nodes reachable through its fanin connections. A cut C for a node n in a DAG is a set of nodes that represent a portion of the logic function, where the nodes in C, known as leaf nodes, ensure that every path from the PI to n passes through at least one leaf node. The cut enumeration algorithm [24] can compute the set of cuts for all nodes in a Boolean network for further analysis. The *maximum fanout-free cone* (MFFC) of a node n is a subset of its TFI, where all paths to any PO pass through n.

B. Technology Mapping

Technology mapping is the process of converting a logic network represented as a DAG into a netlist composed of actual technology library cells. Technology mapping is typically divided into standard cell mapping for ASICs and *K-input lookup table* (K-LUT) mapping for FPGAs. This process involves steps such as cut enumeration, Boolean matching, and iterative mapping, using gates from the library that implement the same Boolean functions to cover the nodes in the network. The quality of technology mapping is usually evaluated in terms of area, delay, and power consumption. A delay-oriented mapping focuses on reducing the delay along the longest path in the cover, while an area-oriented mapping seeks to minimize the total area of the cover [25].

The quality of the mapped netlist depends on the matching possibility between the technology library cells and the DAGs to be mapped. Therefore, the quality of the mapped netlist is influenced by the DAGs to be mapped, and two DAGs representing the same Boolean function but with different structures may lead to significantly different results [26].

C. Structural Choices

Structural choice networks retain different structures that are functionally equivalent or complementary within a Boolean network. Fig. 3 illustrates an example of a choice network. A cut-based technology mapper can consider different options to decide which of the two equivalent structures to cover, which often helps alleviate structural bias, resulting in higher-quality outcomes [27]. The concept of lossless logic synthesis was introduced in [13], applying choice networks to address structural bias issues. However, the creation of choice networks relies on technology-independent optimization, leading to structural limitations. LCH effectively addresses the inefficiency of traditional structural choice algorithms but does not resolve the inherent structural limitations [21]. A novel structural choice method based on logic optimization candidates is used to address structural bias issues; however, its support for large-scale circuits remains limited [28].

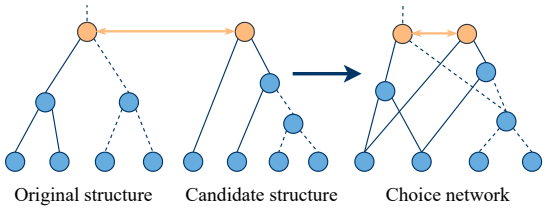


Fig. 3: Combine networks to generate a choice network

III. Our Approach

This chapter provides an overview of the workflow for the *Mixed Structural Choices for Technology Mapping* framework. The framework is a heterogeneous mapping approach that integrates mixed

choice networks, technology-aware evaluation, and mapping-base optimization methods. The algorithms have been open-sourced as the `mch` command in the open-source logic synthesis tool *also*[1]. As shown in the Fig. 4, the main innovations of this work consist of three parts:

(a) *Mixed Structural Choice*: This section explains the concept and creation of mixed structural networks. It preserves the structural characteristics of different logic representations through mixed structural networks and obtains candidate structures based on a path-classification multi-strategy synthesis approach.
(b) *Technology Mapping with MCH*: This section introduces the technology mapping algorithm based on MCH, which evaluates candidates in MCH using technology mapping costs to enhance the mapping process.
(c) *Extending Applications*: This section presents the application of MCH in mapping-based logic optimization, demonstrating its excellent scalability.

A. Mixed Structural Choices

This section introduces the mixed structural choices, a heterogeneous logic representation approach based on structural choices, capable of integrating structural characteristics from different logic representations. MCH retains the original structure while storing other logic representations as candidates, enabling simultaneous evaluation of different logic representations during mapping. This enables the technology mapping phase to select the most suitable logic form, yielding higher-quality mapping results.

AIG, as one of the most popular logic representations, is structurally compatible with XAG, MIG, and XMG. Therefore, we can create mixed choice networks based on AIG combined with any logic representation. Fig. 4 (a) illustrates an example of an mixed choice network, where the original network is an AIG network with a simple structure, while its choice nodes are in MIG, a form with more compact logic expressions. Through MCH, both structural characteristics are simultaneously available, allowing technology mapping to overcome the limitations of single logic representations and achieve an optimized logic network structure with enhanced mapping quality.

The MCH construction algorithm flow is shown in Algorithm 1. The algorithm takes a Boolean network $\mathbb{N}$, a maximum cut size parameter k, a maximum number of cuts for each node parameter l, a maximum number of PIs in the MFFC parameter K, a constant parameter r that controls the proportion of critical path nodes and the synthesis strategies library lib as inputs. The output of the algorithm is the mixed choice network G.

When creating MCH, the first step involves storing the input network within a different logic representation format (line 1). For instance, the input AIG network is one-to-one mapped into the MIG data structure. This process traverses the input AIG nodes and creates corresponding AND nodes, constants, PIs, and POs within the MIG structure, ensuring that the original AIG characteristics are fully retained.

Algorithm 1: Mixed Structural Choices

Input: Boolean network $\mathbb{N}$, cut size k, cut limit l, MFFC max_pi K, ratio r, synthesis strategies library lib
Output: Mixed choice network G

1 more expressive logic representation $\mathbb{N}' \leftarrow$ `One-to-One_Mapping`($\mathbb{N}$);
2 hash table $f \leftarrow$ `Critical_Path_Collection`($\mathbb{N}'$, r);
3 cuts set $C \leftarrow$ cut enumeration ($\mathbb{N}'$, k, l);
4 `MCH` *info.* $\leftarrow$ `Multi-strategy_Structural_Choices` ($\mathbb{N}'$, f, C, K, lib); // Algorithm 2
5 $G \leftarrow$ generate structural choice network (*info.*);
6 return G;

[1]https://github.com/nbulsi/also

Algorithm 2: Multi-strategy Structural Choices Algorithm

Input: Boolean network $\mathbb{N}'$, hash table f, cuts set C, MFFC max_pi K, synthesis strategies library lib
Output: Mixed Structural Choices information *info.*

1 **for** *each gate* $n \in \mathbb{N}'$ **do**
2 **if** $n \in f$ **then**
3 **for** *each cut* $c \in C(n)$ **do**
4 candidate set $E \leftarrow$ Level-oriented synthesis (lib, c);
5 **for** *each gate* $n' \in E$ **do**
6 *info.* $\leftarrow$ mark representative and choice nodes;
7 **else**
8 MFFC $m \leftarrow$ MFFC computation ($\mathbb{N}'$, K);
9 **for** *each cut* $c \in C(n)$ **do**
10 candidate set $E \leftarrow$ Area-oriented synthesis (lib, c);
11 candidate set $E \leftarrow$ Area-oriented synthesis (lib, m);
12 **for** *each gate* $n' \in E$ **do**
13 *info.* $\leftarrow$ mark representative and choice nodes;
14 return *info.*;

Subsequently, MCH uses a hash table to collect critical path nodes, forming the critical path node set f. These nodes include PO nodes with logic depths greater than or equal to the network logic depth * r, along with all nodes on paths from these POs to the PIs (line 2). Next, MCH runs a cut enumeration algorithm to calculate cut information, which is saved according to cut size k and cut number limit l (line 3).

After completing the above steps, MCH uses a *multi-strategy structural choice algorithm* to obtain diversified candidate information, as shown in Algorithm 2. *Mixed structural choices information* (*info.*) refers to the equivalence information between the obtained logic representations and optimization candidates and the original network nodes. The algorithm traverses all nodes. For nodes in the critical path node set f, the focus is on obtaining structures with level advantages. Therefore, the cuts for these nodes are processed using level-oriented synthesis strategies, such as the 4-input NPN library [29], to generate a diverse set of candidate options (lines 2-6). For non-critical path nodes, their MFFCs are computed (line 8). Area-oriented synthesis strategies, such as SOP and DSD, are then applied to generate candidate options with area advantages (lines 9-13). It is worth noting that these synthesized subcircuits do not replace the original network but are retained alongside it as structurally different but functionally equivalent options. After synthesis and restructuring, the obtained candidates will be preserved with new logic representation structures. The *multi-strategy structural choice algorithm* further enriches the diversity of structural choices by combining path classification with multi-objective logic optimization strategies.

Finally, the mixed choice network G is constructed based on MCH information (line 5). Nodes in the original network are defined as `representative nodes`, and their equivalent candidates are added to the network as `choice nodes`. This MCH creation approach achieves a balance between depth and area, and MCH can also create networks optimized for area or delay by adjusting parameter r and synthesis strategies to meet different design goals.

B. Technology Mapping with MCH

Thanks to the characteristics of MCH, we can achieve synergy between logic optimization and technology mapping. This section will explore how to combine MCH with technology mapping algorithms to utilize technology information for technology-aware evaluation.

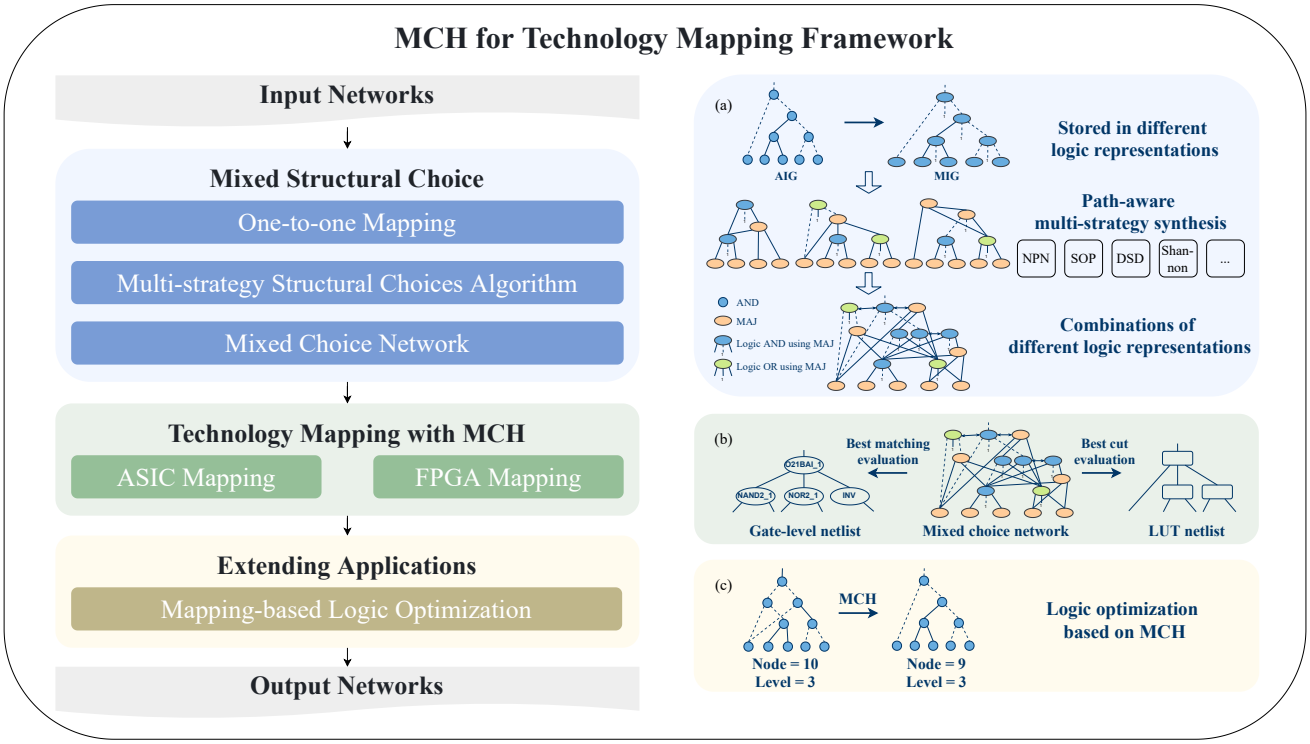


Fig. 4: Mixed Structural Choices for Technology Mapping Framework

MCH-based technology mapping integrates the evaluation and selection of candidate structures from the choice network into existing mapping steps, guiding the mapper to choose the optimal structure for mapping. The process pseudocode for the MCH-based technology mapping algorithm is shown in Algorithm 3. The input of this algorithm is the mixed choice network G obtained from subsection A, and the output is the mapping netlist $\mathbb{M}$. Thanks to the high flexibility of MCH, our method supports technology mapping for both ASIC and FPGA.

First, the mixed choice network is traversed to compute and save each node's cuts (line 1). Next, MCH transfers the candidate structural information from choice nodes to their representative nodes. This process involves iterating through the cut sets of choice nodes and integrating their cuts into the cut sets of the corresponding representative nodes. Finally, a simple sorting is performed to remove cuts exceeding the cut set limit (lines 2-8). When the graph to be mapped contains some structurally different nodes marked as equivalent, cut-based technology mappers need to evaluate different covering options to select the optimal structure for coverage. Therefore, sharing the cut sets of choice nodes is essential.

During technology mapping, the mapper sorts the cut sets based on mapping objectives, selects the optimal cut, and determines the best cell by combining required time and Boolean matching. Since the candidate structures for logic representation and optimization have been successfully stored in the mixed choice networks, the mapper can select the structure with the lowest technology cost based on mapping requirements and technology information. Mapping is a dynamic programming process, for ASIC mapping, the MCH-based mapper continuously updates and selects the cut and matching cell with the lowest cost from numerous candidates. For FPGA mapping, the best cut is dynamically selected in each iteration based on the mapping target (lines 9-13). Finally, the mapped netlist is generated and output, completing the mapping process (lines 14-15).

MCH-based technology mapping achieves a seamless integration of logic optimization and technology mapping. The heterogeneous logic representations and optimization candidates stored in MCH are directly utilized by the mapper, which selects structures with lower technology costs from the candidates. This approach addresses the limitations of the original network structure and directly improves the QoR of technology mapping.

Algorithm 3: MCH-based technology mapping

Input: Mixed choice network G, cut limit l, tech library $tech_lib$
Output: Mapping netlist $\mathbb{M}$

```
1  cuts set C ← Cut enumeration (G, l);
2  for each gate n ∈ G do
3  |  n' ← n's choice node;
4  |  while n' have choice node do
5  |  |  for each cut c ∈ C(n') do
6  |  |  |  insert cut c to C(n);
7  |  |  n' ← n''s choice node;
8  |  limit C(n)'s size <= l;
9  for compute mapping do
10 |  for each gate n ∈ G do
11 |  |  sort C(n) (delay / area);
12 |  |  compute required time;
13 |  |  compute delay-oriented / area-oriented mapping (G, C,
   |  |    tech_lib);
14 M ← generate mapping netlist;
15 return M;
```

C. Extending Applications

Structural bias partly arises from the limitations of logic optimization, which are related to the logic optimization operators applied. Due to constraints imposed by graph structures or the optimization algorithms themselves, logic optimization algorithms often fall into local optima, thereby limiting the discovery of global optimal solutions. This phenomenon has been confirmed in recent *IWLS Programming Contests*[2], where many participating teams have

[2]https://www.iwls.org/iwls2024/

focused on addressing this issue.

MCH offers a new approach to addressing such issues. MCH can be easily extended to mapping-based logic optimization algorithms. MCH-based logic optimization leverages its unique features, combining different logic representations to help the original algorithms overcome local optima. MCH inherently provides a broader range of optimization candidates, and when integrated with logic optimization, it further expands the solution space, making it possible to achieve better optimization results.

Graph mapping is a mapping-based method that remaps the original network to achieve the conversion between logic representations or to optimize logic networks [25]. Fig. 5 illustrates the MCH-based graph mapping algorithm. The newly added work is marked in green in the figure. Our method uses MCH as an intermediate representation for graph mapping, performing cut enumeration and Boolean matching based on the mixed choice network. As shown in the figure, the algorithm can select the most advantageous structure for matching and mapping from three candidate structures. It then completes subsequent mapping and ultimately produces an optimized logic network. At this point, the output network corresponds to the new logic representation selected from the MCH candidates. Therefore, when implementing logic optimization, the output network can be mapped one-to-one to the target logic representation again. And then MCH-based graph mapping is performed to obtain the final optimized network. Through iterative cycles, the structural characteristics of multiple logic representations are combined to achieve higher-quality graph mapping optimization.

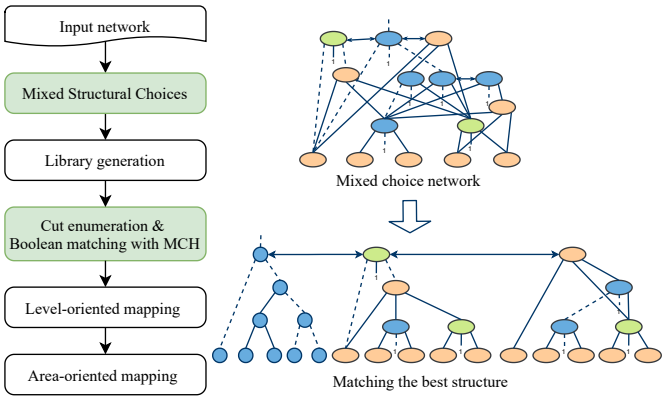


Fig. 5: MCH-based graph mapping

IV. Experimental Results

In this section, we conducted experiments and evaluations of the technology mapping and logic optimization based on MCH. The experiments include technology mapping based on MCH, an evaluation of structural choice effects (comparing MCH and DCH), and an assessment of logic optimization results with MCH. We developed algorithms related to MCH based on the open-source logic synthesis framework *Mockturtle* [30] and conducted experiments. All experiments were performed on a 3.70 GHz AMD(R) Ryzen 5(R) 7500F CPU with 32GB of main memory. Our test cases come from the EPFL benchmark suite [14] which includes arithmetic and random_control benchmark circuits. All results have been formally verified with ABC's `cec` command.

A. MCH for Technology Mapping

1) ASIC Technology Mapping: In the ASIC technology mapping experiment, we evaluated the effectiveness of technology mapping based on MCH. The experiment first used ABC's `compress2rs` flow for iterative optimization to simulate the logic optimization process. Then, the mapping quality is evaluated using the ASAP7 technology library. The experimental results are presented in Table I, which summarizes the area (μm^2) and delay (ps) information reported from the mapped netlist. The CPU runtime (Time) is recorded in seconds. The geometric mean (Geomean) and gain (Improvements) of each data point are also presented.

The experiment primarily compares mapping and structural choices algorithms in ABC. We use the netlist results obtained from the command combinations for different mapping objectives in ABC as the baseline. In contrast, *MCH Balanced* creates the choice network based solely on the input AIG, with candidate structures generated through a strategy that combines critical path and non-critical path node classification. Finally, achieving a balanced improvement in the mapping results. *MCH Delay-oriented* adopts a more aggressive strategy. First, it transforms the input AIG into an XAG. Then, it retains the structural advantages of XAG and creates mixed choice networks of XAG and AIG. During the creation of candidate structures, the range of collected critical path nodes is also expanded. Finally, a delay-oriented mapping is performed. *MCH Area-oriented* combines AIG and XMG to create mixed choice networks and performs area-oriented mapping based on MCH.

The results show that *MCH Balanced* achieves a 3.73% area reduction and a 8.94% delay optimization compared to the baseline. In the experiments for delay-oriented and area-oriented mapping scenarios, DCH yields very limited optimization benefits due to structural bias. In comparison, MCH leverages the structural advantages of XAG and XMG, achieving superior mapping results. Specifically, *MCH Delay-oriented* achieves a 20.35% delay improvement while sacrificing only 9.75% of the area. On the other hand, *MCH Area-oriented* achieves a similar area gain to ABC but significantly reduces the delay. This is because MCH utilizes heterogeneous representations to offer more possibilities for mapping. Additionally, although the computation and evaluation of candidate structures slightly increase mapping time. But for the larger circuits such as "Hypotenuse", MCH can achieve more competitive results in a shorter time. This highlights the significant advantage of MCH for large-scale designs.

2) FPGA Technology Mapping: The *EPFL Combinational Benchmark Suite*'s *Best Results Challenge* focuses on obtaining 6-LUT networks with the minimal LUT count. We use this challenge to evaluate the LUT mapping with MCH. Many studies have demonstrated its effectiveness through this challenge[3].

The experiment first converts the existing results into AIG networks using ABC's `strash` command. Note that the AIGs obtained through this conversion contain many redundant structures, and direct mapping of these AIGs yields poorer LUT netlists compared to the original results. We use the converted AIG networks as inputs to create the mixed choice networks. These mixed choice networks retain the original AIG structures while creating equivalent XMG candidate structures. Finally, we perform area-focused LUT mapping on the mixed choice networks.

Table II presents the results of the test cases in which MCH achieved new records. We set new records for the minimum 6-LUT count in four test cases compared to the *best 6-LUT count results of 2023* (Best (2023)) and in three test cases compared to those of 2024 (Best (2024)). Notably, this experiment directly evaluated the effectiveness of the MCH-based mapper without any logic optimization or post-mapping optimization steps.

In this experiment, our method dynamically evaluates the structural features of both XMG and AIG. MCH stores them as selectable substructures, rather than simplifying the entire circuit into a single

[3]https://github.com/lsils/benchmarks/tree/master/best_results

TABLE I: Experimental results of ASIC technology mapping

Benchmarks	&nf			&dch -m;&nf			dch;map -a			MCH balanced			MCH Delay-oriented			MCH Area-oriented		
	Area	Delay	Time	Area	Delay	Time	Area	Delay	Time	Area	Delay	Time	Area	Delay	Time	Area	Delay	Time
Adder	790.73	1641.45	0.03	790.73	1641.45	0.06	815.72	1755.19	0.03	**776.14**	**1199.59**	0.31	941.08	**875.02**	0.2	**662.94**	2172.97	0.33
Bar	1715.27	124.08	0.06	2015.64	119.56	0.13	1049.28	192.44	0.36	**1706.64**	**113.82**	1.21	2873.49	**114.38**	1.62	1088.16	**187.74**	0.99
Div	14630	28460.6	0.4	15172.8	28503.4	1.69	12406.8	33895	2.67	**13846.8**	**21337.3**	9.18	17006.75	**14816.8**	23.89	**12297.2**	**33317.1**	11.55
Hyp	171964	154106	2.08	158875	156469	321.67	149746	251234	285.81	**163843**	**139545**	12.48	165663.9	**93686.7**	51.95	153742	**146165**	15.71
Log2	22093.4	2762.87	0.97	24377.2	2299.41	3.92	18471	4041.43	9.46	**20660.5**	**2459.82**	8.3	27347.54	**2085.4**	43.61	**18358.3**	**3771.08**	21.73
Max	1477.73	1042.65	0.06	1852.63	1019.86	0.16	1170.16	1380.93	0.66	**1346.4**	**996.11**	0.66	**1428.01**	**965.06**	0.51	**1082.09**	**1100.75**	0.87
Multiplier	20779	1845.75	0.43	20809.6	1841.55	1.45	16815.2	2883.92	3.44	21221.3	**1475.42**	3.6	28432.87	**1313.29**	5.3	**15306.3**	**2553.4**	7.93
Sin	4500.62	1253.97	0.15	4860.48	1181.02	0.56	2919.4	2291.02	2.12	4622.06	**1151.02**	2.56	5889.58	**1007.25**	9.4	3073.9	**1782.68**	6.03
Sqrt	19098.2	38212.1	0.24	18257.5	35005.6	3.98	10103.5	64636.5	5.46	**16452**	**34349.3**	1.25	**13480.84**	**26358.8**	1.48	11781.2	**36746.8**	1.66
Square	12725.6	1556.2	0.22	12698.3	1558.31	0.7	12746.4	2180.82	1.38	**12038.3**	**1206.85**	1.57	12997.8	**842.45**	2.6	**12131.5**	**1734.43**	3.51
Arbiter	1222.77	558.8	0.15	1276.27	562.04	2.48	948.2	598.87	4.96	**1135.37**	**558.8**	4.28	1436.4	**553.81**	5.21	986.5	**581.54**	2.31
Cavlc	231.72	122.49	0.03	274.32	113.35	0.06	171.43	176.98	0.11	244.16	**119.92**	0.16	**257.26**	117.64	0.16	182.06	**162.12**	0.11
Ctrl	56.46	64.78	0.03	56.23	64.78	0.04	40.81	117.8	0.02	**55.09**	**64.1**	0.11	62.52	**62.61**	0.1	**40.58**	**97.8**	0.06
Dec	288.71	42.42	0.03	288.71	42.42	0.04	288.5	44.14	0.07	288.94	42.42	0.2	288.94	42.42	0.2	**288.48**	**43.07**	0.16
I2c	525.32	114.36	0.04	553.24	101.3	0.07	466.12	290.33	0.13	**485.44**	**113.1**	0.41	**534.05**	**95.96**	0.48	**426.28**	**192.75**	0.39
Int2float	86.26	115.65	0.03	104.88	89.33	0.04	78.04	151.16	0.03	**85.83**	**114.36**	0.12	**91.21**	105.23	0.11	**77.18**	**140.22**	0.07
Mem_ctrl	11672.5	666.61	0.6	12170.5	523.48	4.8	8779.48	1052.24	9.88	**9285.38**	**630.69**	9.89	**10920.68**	**522.14**	10.71	9482.35	**1038.76**	10.07
Priority	237.65	433.19	0.04	235.81	433.19	0.06	226.9	424.58	0.1	**225.97**	**424.58**	0.47	**222.27**	**421.38**	0.46	**223.58**	453.28	0.35
Router	91.86	129.6	0.02	103.08	119.5	0.04	78.8	180.49	0.06	108.7	**116.77**	0.16	112.87	112	0.19	**75.99**	**151.22**	0.12
Voter	11442.8	492.94	0.13	11427	492.94	0.51	6850.8	831.15	0.88	**10601.4**	**483.2**	1.36	12771.23	**485.14**	3.42	**6779.99**	**748.13**	1.13
Geomean	2039.15	813.86	0.11	2151.96	768.43	0.42	1623.93	1172.79	0.68	**1963.02**	**741.09**	1.06	2237.97	**648.22**	1.53	**1610.42**	1015.79	1.12
Improvements	-	-		-5.53%	5.58%		20.36%	-44.10%		**3.73%**	**8.94%**		-9.75%	**20.35%**		**21.02%**	-24.81%	

logic representation for comparison. The results demonstrate that the combination of different logic representations effectively overcomes the limitations of a single logic representation structure.

TABLE II: Best area results for the EPFL benchmarks

Benchmarks	Best (2023)		MCH (2023)		Best (2024)		MCH (2024)	
	Node	Lev	Node	Lev	Node	Lev	Node	Lev
Sin	1053	92	**1051**	**81**	1023	110	**1020**	**107**
Sqrt	2983	1526	**2981**	**1374**	-	-	-	-
Square	2959	172	**2956**	**167**	-	-	-	-
Hyp	-	-	-	-	36507	4631	**36506**	**4581**
Voter	1180	30	**1179**	**28**	1166	34	**1165**	**27**

B. MCH for Logic Optimization

This section conducts tests on the MCH-based logic optimization algorithm. The experiment selected the graph mapping algorithm from [25] (*Graph Map*) for comparison. The experiment evaluated the logic optimization effect on the XMG network and the LUT mapping results (*LUT Map*) of the optimized network. This experiment is conducted based on the mixed choice networks composed of MIG and XMG. The experiment performed iterative optimization on the logic network until no further improvement could be achieved, assessing the node count and logic level of the optimized XMG network. Subsequently, the optimized network was mapped to 6-LUT network, and the node count and logic level of the LUT network were evaluated.

The experimental results are presented in Fig. 6. Test cases also come from the *EPFL Combinational Benchmark Suite*. The experiments use the results of the original algorithm iterated to a local optimum as the baseline (Baseline). The horizontal and vertical axes represent the improvements in level and node count achieved by our method for the test cases. The blue squares in the figure represent the improvements in graph mapping optimization for each case under the influence of MCH (*MCH for Graph Map*). The red circles represent the improvements for each case brought by the 6-LUT network generated from the optimized XMG using MCH-based LUT mapping (*MCH for LUT Map*). The blue and red star markers represent the geometric mean of MCH's gains for *Graph Map* and *LUT Map*, respectively. It is evident that with the assistance of MCH, most cases exhibit significant improvements. In some cases, after breaking through local optima, certain metrics improve by as much as 30% or more. For some test cases, such as "Round-robin arbiter" and "Square-root", both the node count and level count were reduced by approximately 50%. For *Graph Map*, our method reduced the XMG level count by an average of 18.59% and the node count by 11.56%. After LUT mapping, the 6-LUT level count and node count were further reduced by 4.71% and 7.31%, respectively.

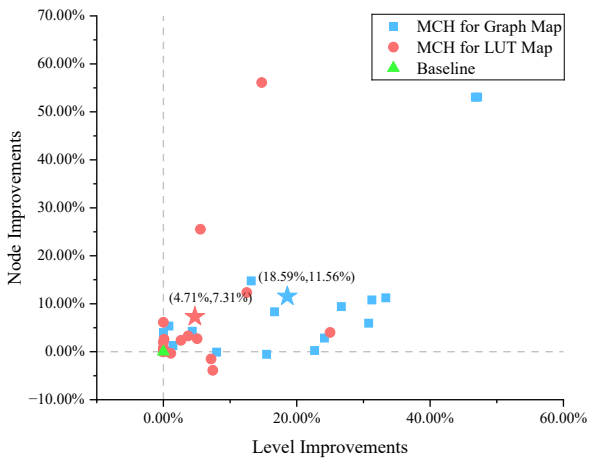


Fig. 6: Experimental results of Graph Map Optimization

V. Conclusion

This paper presents a technology mapping framework based on mixed structural choices operator. The framework introduces mixed structural choices to retain multiple logic representations simultaneously and provides a rich set of candidates through an extensible multi-strategy structural choice method. MCH-based technology mapping effectively combines MCH with technology mapping algorithms, enabling dynamic evaluation of different logic representations and logic optimization candidates based on technology information. Additionally, the framework supports extension to various mapping-based optimization algorithms, helping to overcome the limitations of local optima. The method proposed in this paper is not only of reference value in heterogeneous synthesis and mapping of multi-logic domain, but also in structural bias problems based on structure choices.

REFERENCES

[1] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *TCAD*, vol. 35, no. 5, pp. 806–819, 2016.

[2] E. Testa, M. Soeken, L. Amaru, and G. Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *DAC*, pp. 1–6, 2019.

[3] W. Haaswijk, M. Soeken, L. Amaru, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *ASP-DAC*, pp. 151–156, 2017.

[4] H. Riener, E. Testa, W. Haaswijk, A. Mishchenko, L. Amaru, G. De Micheli, and M. Soeken, "Scalable generic logic synthesis: One approach to rule them all," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[5] W. J. Haaswijk, M. Soeken, L. Amaru, P.-E. Gaillardon, and G. De Micheli, "LUT mapping and optimization for majority-inverter graphs," in *IWLS*, 2016.

[6] W. Haaswijk, M. Soeken, L. Amaru, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *ASP-DAC*, pp. 151–156, 2017.

[7] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. De Micheli, "Advanced functional decomposition using majority and its applications," *TCAD*, vol. 39, no. 8, pp. 1621–1634, 2019.

[8] Z. Chu, L. Shi, L. Wang, and Y. Xia, "Multi-objective algebraic rewriting in XOR-majority graphs," *Integration*, vol. 69, pp. 40–49, 2019.

[9] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "MIXSyn: An efficient logic synthesis methodology for mixed XOR-AND/OR dominated circuits," in *ASP-DAC*, pp. 133–138, 2013.

[10] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "LSOracle: A logic synthesis framework driven by artificial intelligence," in *ICCAD*, pp. 1–6, 2019.

[11] M. Austin, S. Temple, W. L. Neto, L. Amaru, X. Tang, and P.-E. Gaillardon, "A scalable mixed synthesis framework for heterogeneous networks," in *DATE*, pp. 670–673, 2020.

[12] W. L. Neto, Y. Li, P.-E. Gaillardon, and C. Yu, "FlowTune: End-to-end automatic logic optimization exploration via domain-specific multiarmed bandit," *TCAD*, vol. 42, no. 6, pp. 1912–1925, 2022.

[13] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *TCAD*, vol. 25, no. 12, pp. 2894–2903, 2006.

[14] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IWLS*, 2015.

[15] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[16] W. L. Neto, M. T. Moreira, Y. Li, L. Amaru, C. Yu, and P.-E. Gaillardon, "SLAP: A supervised learning approach for priority cuts technology mapping," in *DAC*, pp. 859–864, IEEE, 2021.

[17] P. Wang, A. Lu, X. Li, J. Ye, L. Chen, M. Yuan, J. Hao, and J. Yan, "Easymap: Improving technology mapping via exploration-enhanced heuristics and adaptive sequencing," in *ICCAD*, pp. 01–09, IEEE, 2023.

[18] J. Liu, L. Ni, X. Li, M. Zhou, L. Chen, X. Li, Q. Zhao, and S. Ma, "AiMap: Learning to Improve Technology Mapping for ASICs via Delay Prediction," in *ICCAD*, pp. 344–347, IEEE, 2023.

[19] A. Mishchenko, S. Chatterjee, R. Brayton, X. Wang, and T. Kam, "Technology mapping with Boolean matching, supergates and choices," *Berkeley Logic Synthesis and Verification Group*, 2005.

[20] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices," in *FPGA'10*, pp. 181–184, 2010.

[21] A. Grosnit, M. Zimmer, R. Tutunov, X. Li, L. Chen, F. Yang, M. Yuan, and H. Bou-Ammar, "Lightweight Structural Choices Operator for Technology Mapping," in *DAC*, pp. 1–6, 2023.

[22] C. Chen, G. Hu, D. Zuo, C. Yu, Y. Ma, and H. Zhang, "E-Syn: E-Graph Rewriting with Technology-Aware Cost Functions for Logic Synthesis," in *DAC*, pp. 1–6, 2024.

[23] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pp. 24–40, Springer, 2010.

[24] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *ICCAD*, pp. 354–361, 2007.

[25] A. T. Calvino, H. Riener, S. Rai, A. Kumar, and G. De Micheli, "A versatile mapping approach for technology mapping and graph optimization," in *ASP-DAC*, pp. 410–416, 2022.

[26] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *TCAD*, vol. 16, no. 8, pp. 813–834, 1997.

[27] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs," in *FPGA'06*, pp. 41–49, 2006.

[28] Z. Hu, C. Ma, and Z. Chu, "A Novel Structural Choices Generation Method for Logic Restructuring," in *ISEDA*, pp. 306–311, 2024.

[29] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko, "Fast Boolean matching based on NPN classification," in *2013 International Conference on Field-Programmable Technology (FPT)*, pp. 310–313, 2013.

[30] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage, *et al.*, "The EPFL logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

VSpGEMM: Exploiting Versal ACAP for High-Performance SpGEMM Acceleration

Kai Shi[1], Zhe Lin[2†], Xinya Luan[1], Jianwang Zhai[1] and Kang Zhao[1†]
[1]Beijing University of Posts and Telecommunications [2]Sun Yat-sen University
{shikai, luanxinya, zhaijw, zhaokang}@bupt.edu.cn, linzh235@mail.sysu.edu.cn

***Abstract*—Sparse general matrix-matrix multiplication (SpGEMM) serves as a fundamental operation in real-world applications such as deep learning. Different from general matrix multiplication, matrices in SpGEMM are highly sparse and therefore require a compact representation. This places an additional burden on data preprocessing and exchanging and also causes irregular memory access patterns, which can in turn lead to communication and computation bottlenecks. To break these bottlenecks, we present VSpGEMM, a hardware accelerator for SpGEMM that is tailored and optimized on Versal ACAP. Firstly, a new storage format called BCSX is proposed in VSpGEMM, which offers a unified and block-wise compression strategy to deal with both row-major and column-major representation of non-zero data, enabling fixed-pattern memory accesses and effective data preloading. Secondly, a multi-level tiling mechanism is introduced to decompose the holistic SpGEMM into multiple computation granularities that fit into the AI Engines (AIEs) on Versal in a hierarchical manner, enhancing data reuse. Thirdly, a hybrid partitioning scheme is presented to orchestrate both the AIEs and programmable logic (PL) for intermediate product merging, which together resolve the issues of high memory utilization and communication demand. Experimental results demonstrate a 2.65× speedup over state-of-the-art (SOTA) GEMM design on Versal and an average 33.62× improvement in energy efficiency compared to cuSPARSE on RTX 4090 GPU, showing the efficacy of VSpGEMM.**

I. Introduction

General matrix-matrix multiplication (GEMM) is a widely used operation defined as $A \times B = C$, characterized by accessing matrices A and B in a fixed pattern. However, in many real-world applications, matrices usually have high sparsity, meaning most elements are zeros. In these cases, the conventional GEMM solution becomes inefficient due to the large number of redundant computations performed on zero elements. This inefficiency highlights the need for SpGEMM, where the matrices involved in the computation are sparse. SpGEMM must handle non-zero elements with random spatial locations, which poses challenges including the increased communication overhead of transferring intermediate products and the limited computational intensity with unpredictable memory access to index the non-zero elements from compressed matrix elements, which can severely slow down the applications.

In academia, some methods have been proposed to accelerate SpGEMM on different hardware platforms, including CPUs [1], [2], GPUs [3]–[6], and FPGAs [7], [8]. These solutions mainly focus on the optimization of three aspects, namely, storage formats for storing and indexing sparse matrices [9], algorithms to perform matrix multiplication [10], and workload distribution methods [11]. CPUs and GPUs offer flexible control units and parallelized computation cores to optimize SpGEMM in all computation phases, but they incur high power consumption, which undermines energy efficiency. FPGAs enable customized operations for indexing elements from storage formats but they are limited by on-chip resources when the problems become complicated. Recently, AMD/Xilinx introduced the Versal Adaptive Compute Acceleration Platform (ACAP) [12], a heterogeneous system composed of AI Engines (AIEs), programmable logic (PL), and a processing system (PS). This platform excels in delivering flexible customization for hardware implementation, with computationally intensive multiplications handled by high-performance AIEs and complex control logic managed by PL, which is a promising substrate for efficient SpGEMM implementation.

Despite the high computational power provided by Versal, accelerating SpGEMM on Versal is non-trivial. The recent work CHARM [13] has proposed an efficient method to accelerate GEMM on Versal, which leverages the regular data access patterns of GEMM to achieve efficient pipelining. Nevertheless, new challenges arise when the matrices become sparse. The first challenge lies in excessive memory access to the local memory of AIEs, where the predominant row-wise computation pattern under compressed sparse row (CSR) or compressed sparse column (CSC) format contributes to irregular and unpredictable data accesses, leading to poor data reuse and avoiding data preloading when indexing in the AIE memories. Second, the limited bandwidth in Versal and the high communication demand of transferring intermediate products of SpGEMM can trigger a communication bottleneck, severely hampering overall performance.

To fully unleash the computational power of Versal for SpGEMM acceleration, we propose three innovative optimization strategies: a new compressed storage format for efficient data processing, a multi-level tiling scheme for scalable matrix decomposition, and a hybrid workload partitioning scheme to coordinate different computation resources. These strategies collectively facilitate parallelism of computation, enhance memory management, and reduce communication overhead, finally delivering salient gains in performance and energy efficiency over existing SpGEMM implementation on mainstream computation platforms. The contributions of this paper are summarized as follows:

†Corresponding author

- We introduce a new compressed storage format, BCSX, unifying the representation of CSR and CSC in blocks. This storage format enables fixed-pattern data accesses, preserves high data locality, and supports data preloading.
- We propose a multi-level tiling scheme to hierarchically distribute the computation of SpGEMM to multiple AIEs while enhancing data reuse during computation.
- We develop a hybrid workload partitioning method that efficiently allocates the intermediate product merging operations to AIEs and PL, ensuring minimal communication overhead.
- To the best of our knowledge, we present the first attempt to optimize SpGEMM on Versal ACAP, finally achieving a 2.65× speedup over CHARM on Versal and a 33.62× energy efficiency gain against cuSPARSE on the RTX 4090 GPU.

II. Related Work

Previous works on accelerating SpGEMM using GPUs, CPUs, and FPGAs have proposed various optimizations, including hash tables and the ESC method. However, these approaches often suffer from poor data locality or high communication overhead. For instance, the work [5] on NVIDIA Pascal GPUs reduces scratchpad memory by grouping matrix multiplication (MM) computation tasks and predicting memory size for output matrices. However, it incurs significant overheads from probing hash tables and repeatedly accessing discontinuous memory under the CSR format. The ESC method [14] enhances the parallelism of GPU cores by merging the sorted, index-adjacent MM results, but it faces substantial communication overhead due to the transfer of intermedia products generated during the expansion stage of ESC. FSpGEMM [8] improves data reuse in Gustavson's method [15] with row-wised computation pattern by introducing the compressed sparse vector (CSV) format and a row reordering strategy, maximizing on-chip resource utilization on FPGAs. Nonetheless, the CSV format does not fundamentally eliminate the overheads brought by the row-wised computation pattern, and the row reordering strategy introduces additional computational overhead. HASpGEMM [16] improves workload balance concerning computation and memory access through a micro-benchmarking scheme on asymmetric CPUs but remains constrained by the overheads inherent in the row-wise pattern. In summary, existing methods on SpGEMM acceleration still do not effectively address the problems of high communication costs and random memory access during computation.

On the Versal platform, CHARM [13] is the SOTA work on accelerating GEMM across all heterogenous conponents of Versal, whereas it still struggles with sparse matrix processing. This is because the inherent strategies to process the dense matrices are incompatible with sparse matrix operations that primarily involve compact data representations with non-zero elements. In contrast, our approach seeks to establish fixed memory access patterns and reduce communication overhead for exchanging intermediate products in SpGEMM, which are essential for efficiently processing sparse matrices.

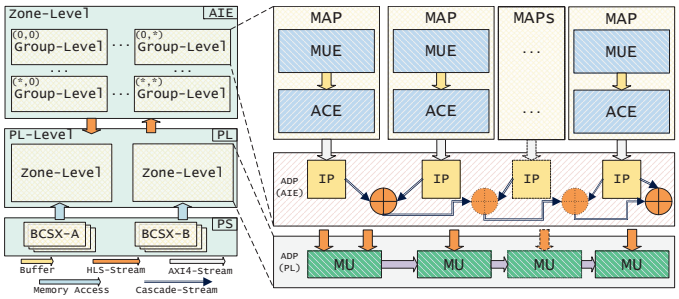


Fig. 1 Architecture of VSpGEMM

III. Versal ACAP Architecture

The Versal ACAP is a heterogeneous platform that integrates AIEs, PL, and PS with a Network on Chip (NoC). The AIEs consist of very-long instruction word (VLIW) processors equipped with single instruction multiple data (SIMD) vector units, operating at a maximum frequency of 1.25 GHz, enabling high parallelism for computation-intensive applications. Each processor in the AIEs is denoted as an AIE tile, which comprises an AIE core for computation, a data memory for program storage and caching, and an AXI4-Stream switch [17] for stream access to the data memory.

Communications within the AIEs are categorized into two types: buffer-based access and stream-based access. The local data memory inside the AIE tile is composed of eight memory banks, allowing for horizontal or vertical data access to neighboring tiles. This configuration provides a four-way memory access which exposes four data memories to one AIE tile. The AXI4-Stream switches divide stream-based access into two modes: cascade-stream for horizontally adjacent tiles and normal streams for all tiles. Notably, the cascade-stream offers a 384-bit data width to transfer large data in one single clock cycle, while normal streams merely support a 32-bit data width per cycle. The PLIO interfaces are responsible for the communication between AIE and PL, each of which offers eight 64-bit output channels and six 64-bit input channels operating at the PL clock frequency, totaling 1.2 TB/s for VCK190 with 39 PLIOs. Data transmission between the PS and PL should use DDR memory as an intermediate buffer and the bandwidth is 100 GB/s. Consequently, a communication gap arises between the high performance of AIE tiles and the limited PLIO and DDR bandwidth, which could become a bottleneck to deploying applications on Versal.

IV. Design Methodology

In this section, we introduce the acceleration paradigm, VSpGEMM, whose overall architecture is depicted in Fig. 1. It decomposes the holistic computation process of SpGEMM into two types of partitions: the MAC-Partition (MAP) and the hybrid ADD-Partition (ADP).

The MAP conducts MMs at a fine-grained level, while the ADP merges the intermediate products to form the final results at a coarse-grained level. Based on this, the MAP can be further divided into a multiplication engine (MUE) and an accumulation engine (AUE) to perform the outer product for SpGEMM, which constitutes the main part of MM. The

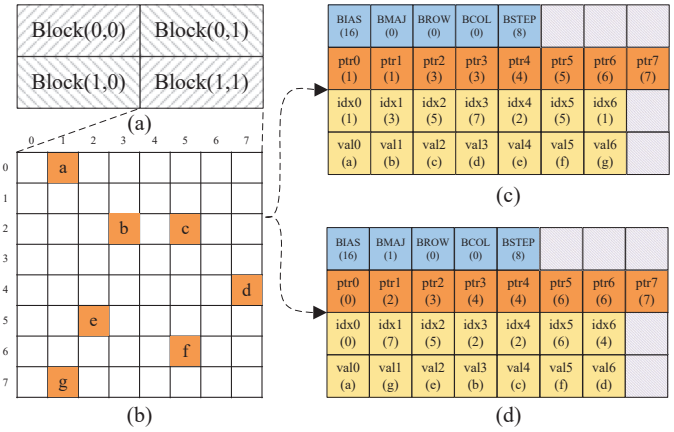


Fig. 2 BCSX Storage Format: (a) The structure of block-wise matrices in BCSX format in memory. (b) The matrix Block(1,1) in 2D dense form. (c) The BCSX format of Block(1,1) in row-major. (d) The BCSX format of Block(1,1) in column-major. Attribute descriptors in BCSX are marked in blue, pointers in red, and indices and values in yellow.

hybrid ADP is designed to be conducted partially in AIEs and PL to minimize the transmission of intermediate products generated by MAPs. The complete architecture of VSpGEMM is supported by the proposed BCSX storage format, multi-level tiling scheme, and hybrid workload partitioning strategy.

A. BCSX Storage Format

The commonly used storage formats like CSR and CSC form the foundation of various works on SpGEMM since they preserve row-wise and column-wise data locality, respectively. The prior arts attempt to use a single storage format to construct the two input matrices (denoted as A and B), and adopt Gustavson's algorithm for SpGEMM. In such cases, although matrix A exhibits high data locality, matrix B usually experiences irregular and unpredictable memory accesses, preventing effective memory optimizations such as data preloading and reuse. Moreover, CSR and CSC are monolithic representation methods that compress the entire matrix at once, making it difficult to partition and distribute computation workloads to hardware resources at a fine-grained level. To address these issues, in this paper, we propose a new storage format called Blocked Compressed Sparse eXtension (BCSX). This storage format aims to merge the strengths of these conventional storage formats, offer a consistent storage method for both row- and column-major representations, and facilitate data access in a regular and block-wise pattern. In the rest of this section, we illustrate the details of the BCSX format, the implementation of BCSX on AIE, and the memory access patterns supported by BCSX.

Construction of BCSX Format. BCSX utilizes five descriptors named BIAS, BMAJ, BROW, BCOL, BSTEP, and three arrays named idx*, ptr*, and val* to capture the structural information and non-zero elements in a per-block manner. Fig. 2 demonstrates BCSX with an 8×8 matrix as an instance.

The five descriptors identify the arrays in matrix blocks from memory and facilitate vectorized loading. The BIAS serves as a relative offset of the idx* array to the memory address of the current block. The BMAJ is a binary value that specifies whether the matrix is constructed in row-major (0) or column-major (1) order. When both the input matrices A and B are constructed in row-major order with BMAJs set to 0, a row-wise computation pattern is enabled. If the two BMAJs are set to 1 and 0 respectively, an inner product method can be applied. Conversely, when the BMAJs are set to 0 and 1, the outer product can be conducted, which is the adopted algorithm in this paper. With BMAJ to indicate the order, all computation patterns can be flexibly supported. The BROW and BCOL reveal the spatial coordinates of the current block within the entire sparse matrix, aiding both AIE and PL in organizing matrix blocks for multiplication and merging. The BSTEP determines the vector length of AIEs to fetch data from data memory, which could be adjusted in light of matrix sparsity and workload for a single AIE tile, thus enabling a dynamic load step for AIE tiles working on various matrix blocks.

TABLE I Features of BCSX compared to CSR and CSC

Storage Format \ Memory Access	Row Major	Column Major	Block-wise Structrue	Vectorized Loading
CSR	✓	×	×	×
CSC	×	✓	×	×
BCSX	✓	✓	✓	✓

The ptr* array records the number of non-zero elements in each line of the current block. Specifically, each element of ptr* records the cumulative number of non-zeros for the current and preceding rows or columns. The idx* and val* arrays store all the non-zeros of the current block in BMAJ order, with each element in these two arrays corresponding one-to-one.

Vectorized Loading on AIE. Different from CSR and CSC, BCSX skips the prefix 0 and captures the exact number of rows or columns in blocks to facilitate vectorized loading in AIE kernels. Within each AIE tile, memory access to BCSX for blocks in B is vectorized, which loads data in vectors of length BSTEP, which is a power of two. However, the prefix 0 in conventional formats impedes data alignment because the length of ptr* becomes non-divisible by a vector step, requiring an extra load to fetch the last few ptr* entries. In an AIE tile, the iterator for vector can only iterate at an entire vector step, causing misalignment when loading ptr* and leading to incorrect vector iterator for the idx* and val* arrays. To enable efficient vectorized loading in AIE tiles, BCSX eliminates the prefix 0, records the vector step in descriptor BSTEP, and utilizes the BIAS descriptor to mark the relative starting address of the idx* array, ensuring an aliquot length of ptr* for efficient vectorized loading. In general, BCSX facilitates MMs on AIEs with the features shown in TABLE I.

Memory Access Patterns with BCSX. Suppose the input matrices are in square with non-zeros uniformly distributed across rows and columns, and let NNZ denote the average number of non-zeros per row. In row-wise production, as illustrated in Fig. 3(a), to calculate row $C_{(0,*)}$ of matrix C, the non-zeros $A_{(0,0)}$ and $A_{(0,2)}$ in row $A_{(0,*)}$ are loaded. For each non-zero in row $A_{(0,*)}$, the corressponding rows $B_{(0,*)}$ and

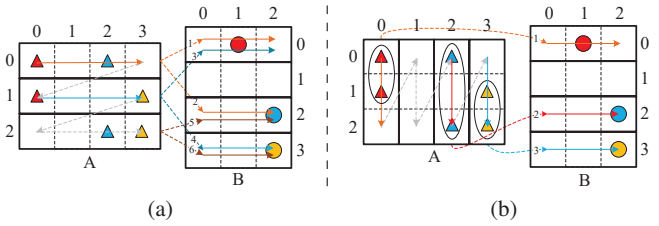


Fig. 3 Memory access patterns of two representative SpGEMM algorithms: (a) Gustavson's method; (b) Outer product. Gustavson's method involves six irregular memory accesses to matrix B, while the outer product has only three accesses in a fixed pattern with data locality.

$B_{(2,*)}$ are loaded as multipliers. Similarly, while calculating row $C_{(1,*)}$, row $B_{(0,*)}$ and $B_{(3,*)}$ are loaded, resulting in worse performance due to the random and discontinuous accesses to the same row $B_{(0,*)}$, preventing data preloading in AIE kernels, since the indexing of those non-zeros in matrix A is non-prior knowledge to AIEs. In contrast, the memory access becomes regular by using BCSX, as illustrated in Fig. 3(b), with each column $A_{(*,k)}$ and corresponding row $B_{(k,*)}$ loaded simultaneously for calculating the output matrix C. This allows non-zero $A_{(0,k)}$ and $A_{(1,k)}$ from $A_{(*,k)}$ to get multiplied by vector $B_{(k,*)}$, creating fixed memory access patterns to column $A_{(*,k)}$ and row $B_{(k,*)}$ due to the prior knowledge of k that is considered in design, which enables data preloading for columns in A and rows in B, thus opening up new opportunities for performance improvement.

B. Multi-Level Tiling Scheme

We propose a multi-level tiling scheme that distributes the computation workload of SpGEMM to PL and AIEs. The key idea is to fully utilize the computation power of AIEs to perform multiplication and addition operations at a fine-grained level while taking advantage of the abundant memory resources of PL to accomplish the merging of intermediate results at a coarse-grained level. This tiling scheme can significantly reduce the *reuse distance* [18] of matrix elements from a matrix scale to a block scale, thereby enhancing performance by minimizing redundant data transfers. It is worth noting that the proposed tiling process is performed hierarchically, progressively breaking down the SpGEMM computation from the PL level to the zone level, then to the group level, block level, and finally, the kernel level, as shown in Fig. 4.

PL- and Zone-Level Tiling. Originally, matrix A with the size of $M \times K$ and matrix B with the size of $K \times N$ are stored in DDR in the BCSX format at a PL level. Matrix A and B are then divided into $PX \times PZ$ and $PZ \times PY$ zones, respectively, where each zone is transferred from DDR to on-chip BRAMs of PL and then mapped to the AIEs. As shown in Section III, the bandwidth of DDR is much smaller than that of the PLIOs. Hence, we allocate ping-pong BRAMs to store the zone data for both of the input matrices, as shown in Fig. 5. Specifically, one BRAM acts as a ping buffer to receive a zone from DDR, while the other functions as a pong buffer to produce a zone for AIEs, allowing parallel communication between DDR and PL, as well as between PL and AIEs.

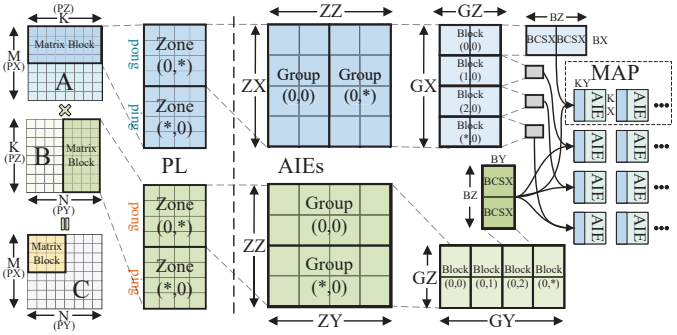


Fig. 4 Multi-Level Tiling Scheme across AIEs and PL

Group-Level Tiling. We further divide the zones into smaller groups to enhance the parallel execution of AIEs. We note that the size of a zone may exceed the capacity of the AIE array. Therefore, the zones of matrix A and B are subdivided into $\frac{ZX}{GX} \times \frac{ZK}{GK}$ and $\frac{ZK}{GK} \times \frac{ZZ}{GZ}$ groups, respectively, where each group corresponds to a group-level region on AIEs. To obtain the result $Group_{(0,0)}$ for zone C, all groups from A and $ZZ \times Group_{(*,0)}$ from B must be sent to the AIEs. Consequently, these groups from zone B need to be transferred ZZ times to get multiplied with all the groups from zone A. Hence, using the on-chip BRAMs to store all groups of zone B, the entire ZZ times of accesses could be reused, avoiding repetitively loading data from the DDR in the monolithic methods. Furthermore, these groups of zone B need to get transferred to AIEs ZZ times as well, which heavily increases the communication burdens. Therefore, on the zone level, we utilize circuit switches to broadcast the $Group_{(*,0)}$ to group-level tiles on AIEs simultaneously, achieving an ZZ-fold data reuse ratio compared to non-titling scenes.

Block- and Kernel-Level Tiling. Matrices A and B at the group level are further divided into $GX \times GZ$ and $GZ \times GY$ blocks, respectively, where data accesses to A are column-wise and to B are row-wise at the block level. Finally, the blocks are then divided into kernel-level tiles, and the outer product method is performed at this fine-grained level. To be more specific, the outer product at the kernel level can be reduced to the scalar-vector multiplications using elements in columns from A and corresponding rows from B in a predetermined execution order. This approach boosts the computation efficiency of AIE cores by facilitating data reuse in a fixed pattern.

Again using Fig. 3(b) as an example, to complete the outer product, matrix A follows a per-column data fetching pattern. Both the two elements in the first column of A are fetched and multiplied with the element in the first row of B (marked as red triangles for A and circles for B, respectively). Since the access pattern is determined beforehand, the first row of B only needs to be fetched once, whereas the Gustavson's computation pattern shown in Fig. 3(a) necessitates fetching the same row of B multiple times. For instance, the first row of B should be multiplied with both the first element in the two rows of A. However, this is a random way of fetching the rows in B due to the inherent irregularity of the spatial distribution of non-zeros from rows of A in Fig. 3(a), thus impeding data reuse. In summary, our method subtly leverages

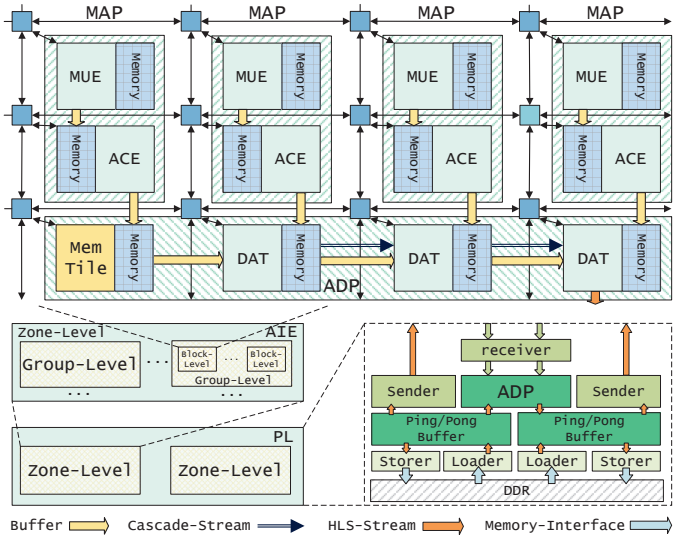


Fig. 5 Hybrid ADD-Partition on AIEs and PL

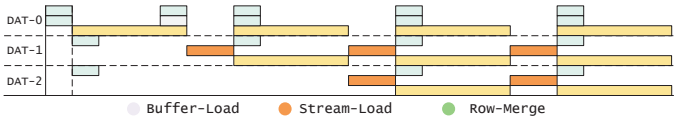


Fig. 6 Pipeline model of DATs

both the fixed memory access pattern enabled by BCSX and the outer product method of MM to maximize the data reuse during the data fetching of matrices A and B at the kernel-level.

C. MAC-Partition and Hybrid ADD-Partition

The MM of $A_{M\times K} \times B_{K\times N}$ generates K batches of intermediate products (IP), which are sent to adders to be merged into the final results. We design MAPs on AIEs to perform detailed MMs using the outer product method with the support of BCSX. Nonetheless, merging all IPs to standalone adders within AIEs is impractical due to limited data memory and communication bandwidth for non-neighboring AIEs. The situation gets exacerbated when performing all the merges with frequent data transfers via the PLIOs between AIEs and PL. To address these issues, we propose the hybrid ADP to perform partial merges at fine-grained levels within AIEs to reduce the size of IP locally. The remaining merge operations are completed on PL to leverage the large on-chip memory, thereby further reducing the communication burden.

MAC-Partition on AIEs. The MUE at the kernel level produces sparse vectors, composing at most K batches of IPs with the size of $M \times N$ when both the column from A and the corresponding row of B contain non-zeros. Once a batch of IP is produced, it is transferred to the ACE through AXI-Stream. The ACE accumulates the IPs to the data memory on the local AIE tile to form the first-stage IPs. Since the MUE and ACE are physically adjacent to AIEs, compared to transferring IPs entirely to PL through PLIOs or accumulating entirely on AIEs, the adjacent AXI-Swithes enable fewer cycles for moving IPs to accumulators, significantly reducing communications latency. Besides merging operations in the accumulating stage, combining corresponding blocks for a result matrix requires additional transfers and data memory as well. Therefore, we propose the hybrid ADP to address this.

ADD-Partition on AIEs. As illustrated in Fig. 5, each ADP is positioned vertically above or below a MAP, where the MAP shares the same memory bank with the ADP to achieve maximum bandwidth in AIEs. Inside each ADP, dense accumulation tiles (DAT) are connected in cascade-stream to combine rows of each matrix simultaneously. Specifically, the first-stage IPs in the output from the first two MAPs are input into the first DAT through a single cycle load via buffer-based communication manner. Subsequent DATs receive one IP stored in a shared memory bank from MAP and accept rows for accumulation from the cascaded proceeding DATs, providing high communication bandwidth. The first DAT queries two IPs for merging and utilizes an additional neighboring AIE tile as shared memory storage, allowing parallel operation of these chained DATs. Consequently, the pipeline of this IP merging scheme at the kernel level is presented in Fig. 6. For each block level in AIEs, the ADP merges BZ blocks with $(BZ-1)$ DATs and outputs one block as the second-stage IP to PL, reducing the communication amount of IPs at a fine-grained level.

ADD-Partition on PL. On the PL side, ADPs are instantiated in multiple Merging Units (MU). For each zone-level second-stage IPs produced from AIEs, MUs in ADPs of PL receive $(ZX \times ZZ \times ZY) \times (GZ \times GY)$ block-level IPs for merging, which operate sequentially with ping-pong buffers as illustrated in Section IV-B. The MUs are chained in a dataflow manner, with the header MU merging two IPs in row order, simultaneously forwarding the sum of each row to the next MU as an augend. This enables ADP on PL with the initiation interval (II) = 1, which allows for a minimum number of clock cycles between successive loop iterations, facilitating continuous data processing without stalls. Under the cooperation of ADPs across AIEs and PL, we achieve low latency with reduced communication burdens in merging IPs.

V. EXPERIMENTS

We evaluate the performance, power consumption, energy efficiency, and latency of VSpGEMM. To demonstrate the effectiveness of VSpGEMM, we compare our results with the SOTA work CHARM on Versal ACAP and the cuSPARSE library on NVIDIA GPUs.

A. Experimental Setup

For experiments on the Versal ACAP, we use the VCK190 toolkit to evaluate VSpGEMM and CHARM. These experiments are built with Vitis 2024.1, and results are collected from an onboard testbench. We set the PL running at 250 MHz and the AIEs at 1.25 GHz, and use BEAM Tool [20] and xbutil [21] to evaluate the onboard power of VCK190. For comparisons with cuSPARSE, the experiments are conducted on an NVIDIA RTX 4090 GPU with CUDA 12.6 and an Intel Xeon Platinum 8375C CPU with 128 cores at 3.5 GHz for the host-side program. We use nvidia-smi [22] to evaluate the power of the RTX 4090. Additionally, iterations for all of these experiments are configured for more than one minute to obtain a stable performance at runtime.

TABLE II Throughput of VSpGEMM

Datasets			CHARM [13]			VSpGEMM (Ours)			Speedup
Benchmark (ID-Name)	M × N	NNZ	AIE Tiles	PLIOs	Throughput (GOPS)	AIE Tiles	PLIOs	Throughput (GOPS)	
2397-football	115 × 115	663	144	64	26.53	58	12	46.64	1.76 ×
1945-TF11	216 × 236	1607	144	64	179.59	176	32	435.65	2.42 ×
1982-GL6_D_10	163 × 341	2053	144	64	147.28	264	44	568.01	3.85 ×
2209-Trefethen_500	500 × 500	4489	198	146	339.17	176	32	1204.39	3.55 ×
2210-Trefethen_700	700 × 700	6677	192	96	1219.91	264	44	2055.49	1.68 ×

TABLE III Latency, power and energy efficiency (EE) of VSpGEMM

Benchmark (ID-Name)	cuSPARSE [19]			CHARM			VSpGEMM (Ours)			EE. Gain	
	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	V.S. cuSPARSE	V.S. CHARM
2397-football	39.21	89	0.08	0.36	22.28	1.19	0.21	15.45	3.02	37.75×	2.54×
1945-TF11	28.60	90	0.21	0.36	23.26	7.72	0.26	24.20	18.00	85.71×	2.33×
1982-GL6_D_10	24.81	89	1.50	0.36	22.41	6.57	0.26	28.19	20.15	13.45×	3.07×
2209-Trefethen_500	25.67	92	2.58	0.55	28.23	12.01	0.46	25.38	47.45	18.38×	3.95×
2210-Trefethen_700	25.91	95	5.61	0.67	31.08	39.25	0.64	28.64	71.77	12.79×	1.83×

We utilize datasets with sparse matrices from the SuiteSparse Matrix Collection [23], and we assume that matrices are pre-stored in the BCSX format, similar to the work on GPU [24]. The CHARM implementation was initially based on Vitis 2021.1, where some AIE APIs are deprecated, and the AIE clock operates at a maximum of 1.0 GHz, different from Vitis 2024.1, which operates at 1.25 GHz. Therefore, we port CHARM to Vitis 2024.1 to ensure a fair comparison.

B. Evaluation of Performance

In this experiment, we evaluate the performance of VSpGEMM and compare our results with the SOTA GEMM implementation on Versal, the CHARM [13] framework. The AIE kernels in VSpGEMM are currently implemented using the INT16 datatype, so we limit the values in datasets to the range of 16-bit integer for both input and output while maintaining the original spatial distribution of non-zeros. VSpGEMM schedules the number of MAPs based on the size of input matrices to efficiently utilize AIE tiles while maintaining high communication efficiency within MAPs. In current settings, the zone on AIEs performs SpGEMM with two maximum matrix dimensions of 256 or 384 per iteration, which are mapped to 176 or 264 AIE tiles respectively. VSpGEMM determines the usage of AIE tiles according to the input matrix size that is closest to the multiples of these two maximum dimensions, ultimately achieving optimal tile utilization. The experimental results are shown in TABLE II. As can be seen, for smaller matrices, VSpGEMM utilizes fewer AIE tiles because the dimensions of each level in tiling are small, which can fit into one single MAP. As the problem size increases, a large number of MAPs are mapped into AIEs and a higher level of parallelism is achieved, resulting in a performance boost. We can observe that VSpGEMM consistently outperforms CHARM on different problem sizes, achieving an average speedup of 2.65×.

C. Evaluation of Energy Efficiency

We investigate the latency, power, and energy efficiency of VSpGEMM, with a comparison to cuSPARSE [19] and CHARM [13]. Given that cuSPARSE does not support INT16 datatype, we set the datatype of cuSPARSE to FP32 to perform the calculation. The latencies of VSpGEMM and CHARM both increase as the matrix scale increases, while the latency of cuSPARSE decreases due to the low utilization of SMs on GPU at small matrix scales. For all problem sizes, VSpGEMM achieves the lowest latency and outperforms the CHARM on Versal and the cuSPARSE on GPU, which fully justifies the capability of the Versal platform on SpGEMM acceleration.

The power consumption of VSpGEMM is primarily composed of AIE cores, the PL domain, and the PS domain [25]. Among all the platforms, VSpGEMM achieves the lowest power consumption of 15.45W on the problem size of 115×115. Moreover, VSpGEMM demonstrates the highest energy efficiency among all the problem sizes, with an average energy efficiency gain of 33.62× and 2.74× compared to cuSPARSE and CHARM, respectively. Therefore, the experimental results show that VSpGEMM can achieve high performance and high energy efficiency at the same time, which is an ideal solution for real-world applications, especially on edge devices that have a stringent requirement for energy efficiency.

VI. Conclusion

In this work, we design VSpGEMM, an accelerator framework for SpGEMM on Versal ACAP. We introduce the BCSX format, a unified compressed storage format for storing sparse matrices in a block-wise row or column-major order, facilitating efficient memory access patterns with efficient data preload. We also propose a multi-level scheme under BCSX to enhance data reuse across AIEs and PL hierarchically. Additionally, the hybrid ADD-partition method is developed to reduce communication overhead for intermediate products. As a result, our approach achieves an average speedup of 2.65× compared to CHARM and a 33.62× energy efficiency gain compared to cuSPARSE.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB2901100), the National Natural Science Foundation of China (No. 62404257, 62404021), the Beijing Natural Science Foundation (No. 4244107, QY24216, QY24204), and the Guangdong Basic and Applied Basic Research Foundation (2024A1515013155).

REFERENCES

[1] Z. Gu, J. Moreira, D. Edelsohn, and A. Azad, "Bandwidth optimized parallel algorithms for sparse matrix-matrix multiplication using propagation blocking," in *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2020, pp. 293–303.

[2] M. M. A. Patwary, N. R. Satish, N. Sundaram, J. Park, M. J. Anderson, S. G. Vadlamudi, D. Das, S. G. Pudov, V. O. Pirogov, and P. Dubey, "Parallel efficient sparse matrix-matrix multiplication on multicore platforms," in *IEEE International Conference on High Performance Computing (HiPC)*, 2015, pp. 48–57.

[3] M. Winter, D. Mlakar, R. Zayer, H.-P. Seidel, and M. Steinberger, "Adaptive sparse matrix-matrix multiplication on the GPU," in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2019, pp. 68–81.

[4] Z. Gu, J. Moreira, D. Edelsohn, and A. Azad, "Bandwidth Optimized Parallel Algorithms for Sparse Matrix-Matrix Multiplication Using Propagation Blocking," in *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2020, pp. 293–303.

[5] Y. Nagasaka, A. Nukada, and S. Matsuoka, "High-Performance and Memory-Saving Sparse General Matrix-Matrix Multiplication for NVIDIA Pascal GPU," in *International Conference on Parallel Processing (ICPP)*, 2017, pp. 101–110.

[6] S. Luo, B. Wang, Y. Shi, X. Zhang, Q. Xue, and S. Ma, "Sparm: A Sparse Matrix Multiplication Accelerator Supporting Multiple Dataflows," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2024, pp. 122–130.

[7] L. Song, Y. Chi, A. Sohrabizadeh, Y.-k. Choi, J. Lau, and J. Cong, "Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2022, pp. 65–77.

[8] E. B. Tavakoli, M. Riera, M. H. Quraishi, and F. Ren, "FSpGEMM: A Framework for Accelerating Sparse General Matrix–Matrix Multiplication Using Gustavson's Algorithm on FPGAs," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 2024.

[9] Z. Xie, G. Tan, W. Liu, and N. Sun, "A pattern-based SpGEMM library for multi-core and many-core architectures," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 33, no. 1, pp. 159–175, 2021.

[10] M. Parger, M. Winter, D. Mlakar, and M. Steinberger, "Speck: Accelerating gpu sparse matrix-matrix multiplication through lightweight analysis," in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2020, pp. 362–375.

[11] C. Zhang, M. Bremer, C. Chan, J. Shalf, and X. Guo, "ASA: A ccelerating S parse A ccumulation in Column-wise SpGEMM," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 4, pp. 1–24, 2022.

[12] AMD, "AI Engine Kernel Coding User Guide," 2024. [Online]. Available: https://docs.amd.com/r/zh-CN/ug1079-ai-engine-kernel-coding

[13] J. Zhuang, J. Lau, H. Ye, Z. Yang, S. Ji, J. Lo, K. Denolf, S. Neuendorffer, A. Jones, J. Hu *et al.*, "Charm 2.0: Composing heterogeneous accelerators for deep learning on versal acap architecture," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 3, pp. 1–31, 2024.

[14] S. Dalton, L. Olson, and N. Bell, "Optimizing Sparse Matrix—Matrix Multiplication for the GPU," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 4, pp. 1–20, 2015.

[15] F. G. Gustavson, "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," *ACM Transactions on Mathematical Software (TOMS)*, vol. 4, no. 3, pp. 250–269, 1978.

[16] H. Cheng, W. Li, Y. Lu, and W. Liu, "HASpGEMM: Heterogeneity-Aware Sparse General Matrix-Matrix Multiplication on Modern Asymmetric Multicore Processors," in *International Conference on Parallel Processing (ICPP)*, 2023, pp. 807–817.

[17] AMD, "Versal AI Engine Documentation," 2024. [Online]. Available: https://docs.amd.com/r/en-US/am009-versal-ai-engine

[18] R. K. Maeda, Q. Cai, J. Xu, Z. Wang, and Z. Tian, "Fast and accurate exploration of multi-level caches using hierarchical reuse distance," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 145–156.

[19] NVIDIA, "cuSPARSE Documentation," 2023. [Online]. Available: https://docs.nvidia.com/cuda/cusparse/

[20] Xilinx, "BEAM Tool for VCK190 Evaluation Kit," 2024. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/973078551/BEAM+Tool+for+VCK190+Evaluation+Kit

[21] AMD, "Vck190 evaluation board user guide," 2023. [Online]. Available: https://docs.amd.com/r/en-US/ug1366-vck190-eval-bd/Environmental

[22] NVIDIA, "NVIDIA System Management Interface Documentation," 2024. [Online]. Available: https://docs.nvidia.com/deploy/nvidia-smi/index.html

[23] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, 2011.

[24] Y. Niu, Z. Lu, H. Ji, S. Song, Z. Jin, and W. Liu, "TileSpGEMM: A Tiled Algorithm for Parallel Sparse General Matrix-Matrix Multiplication on GPUs," in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2022, pp. 90–106.

[25] AMD, "Power Design Manager User Guide," 2024. [Online]. Available: https://docs.amd.com/r/zh-CN/ug1556-power-design-manager

3D-Flow: Flow-based Standard Cell Legalization for 3D ICs

Yuxuan Zhao, Peiyu Liao, Bei Yu
Department of Computer Science and Engineering
The Chinese University of Hong Kong

***Abstract*—The standard-cell placement legalization is a critical step in the physical design. The emerging 3D ICs have brought challenges to traditional legalizers on efficiency and effectiveness. In this work, we present a fast flow-based legalization algorithm, 3D-Flow, that minimizes cell displacement in a 3D solution space. Our legalizer resolves overflowed bins by finding the shortest augmenting path on a 3D grid graph, utilizing an effective branch-and-bound algorithm. Moreover, a post-optimization with a cycle-canceling algorithm is proposed to minimize the maximum displacement. Our approach leverages the global perspective inherent in network flow methods, considering multiple dies in 3D ICs to minimize cell displacement. Experimental results on ICCAD 2022 and 2023 contest benchmarks demonstrate our proposed algorithm achieves up to 13% and 43% less average and maximum cell displacement compared to state-of-the-art legalizers in a similar runtime.**

I. Introduction

Placement legalization is a critical step in the physical synthesis flow. It is typically invoked after global placement and timing optimization, such as gate sizing and buffer insertion, which may compromise the legality of a placement. An ideal legalization algorithm removes all overlap of instances while preserving the quality of the given placement. As technology scaling approaches its physical limits, 3D integrated circuits (3D ICs) [1] have emerged as a viable solution for extending Moore's Law. The vertical stacking of multiple dies in 3D ICs introduces new challenges and optimization opportunities for legalization algorithms.

Existing legalization algorithms can be broadly categorized into two types: *greedy search* and *formal optimization*. Most legalizers [2]–[5] fall into the former category, sequentially moving cells to the nearest free locations. Typically, the sequence of movement is determined by the cells' x-position, arranged in either ascending or descending order. A notable example of this approach is Tetris [2], which has been widely adopted in modern placement tools [6], [7]. Abacus [4] follows a similar procedure as Tetris, but moves the cells already legalized in a row by dynamic programming to minimize the total movement. While the greedy approaches are fast and flexible, they can result in excessive cell displacement, especially in designs with high utilization or blockages caused by macros.

In contrast, formal optimization approaches model placement legalization through mathematical formulations such as network flow [8]–[12] or diffusion [13]. Specifically, the diffusion model [13] employs force-directed methods to globally reposition cells. In flow-based methods, legalization is modeled as a minimum-cost flow problem, where cells are moved from source bins to sink bins, resolving overflow with minimal cell movement. If all cells are of uniform size, the problem can be efficiently solved in polynomial time [8]. However, real-world designs have cells with different sizes, necessitating a flow realization step in these approaches. Cho *et al.* [9] introduce a history-based legalization algorithm that incorporates history data into the cost formulation to avoid unfeasible flows. Inspired by the successive shortest path algorithm [14], BonnPlaceLegal [10] ensures that only flow augmentations, which can be realized exactly by cell movement, are considered. The algorithm iteratively computes the shortest paths from source bins to sink bins using Dijkstra's algorithm. While this approach achieves satisfactory cell displacement results, the iterative calls to Dijkstra's algorithm make it unscalable for large designs. To improve efficiency, Darav *et al.* [11] propose utilizing the breadth-first search (BFS) algorithm to identify multiple candidate paths. Although this algorithm is robust and efficient, it may result in large total cell displacement as it does not consider the cost when selecting the augmenting paths. Innovations are needed to balance the trade-off between solution quality and efficiency in formal optimization approaches.

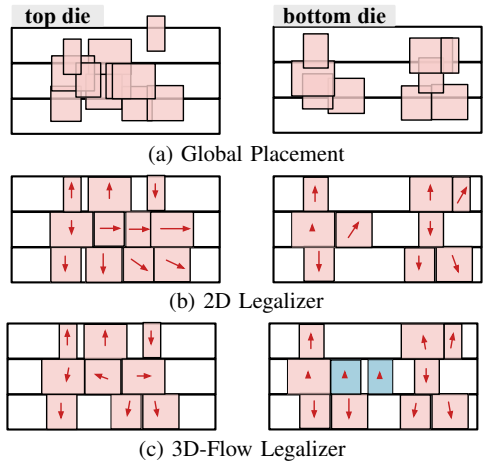


Fig. 1 Motivation for 3D-Flow legalizer. Global placement (a) is given for a 3D IC with two dies. (b) and (c) show legal placements, with red arrows indicating the cell displacement between global and legal placements. By moving two **blue**-colored cells (c) from the top to the bottom die, our 3D-Flow legalizer achieves reduced displacement.

In the context of 3D placement, placers need to determine both cell positions and cell partitioning to optimize design objectives. Pseudo-3D placers [15], [16] separate the partitioning and placement phases. They utilize 2D placement tools to determine cell positions while employing either partitioning-first [15] or partitioning-last [16] flows to perform the die assignment. In contrast, true-3D placers [17]–[20] relax discrete die partitioning constraints and adopt analytical approaches to simultaneously optimize cell partitioning and positions in a 3D solution space. Recent studies demonstrate the superiority of true-3D placers, as they explore a larger solution space. However, legalization algorithms are overlooked in existing true-3D placers. The state-of-the-art (SOTA) true-3D placers [18], [19] greedily assign cells to the nearest die after global placement and then legalize the placement with fixed die assignments using traditional 2D legalizers [2], [4]. However, legalizing the placement in a 3D solution space will produce results with higher quality. Fig. 1 shows an example that by moving cells across the dies, the total displacement and maximum displacement can be significantly reduced at the legalization stage for 3D ICs.

In this work, we present a fast flow-based legalization algorithm for 3D ICs. Our approach leverages the global perspective inherent

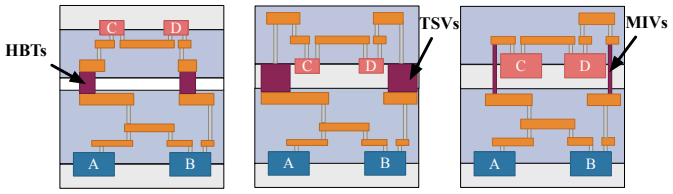


Fig. 2 The typical types of 3D ICs. (a) F2F bonded 3D ICs enable heterogeneous technology integration without space crowding on the silicon layer. (b) TSVs occupy large silicon area, resulting in low integration density. (c) Monolithic 3D ICs require sequential fabrication with low yield.

in network flow methods, considering multiple dies in a 3D solution space to minimize cell movement. Our contributions are summarized as follows.

- We propose the first 3D legalizer, 3D-Flow, to minimize cell displacement in a 3D solution space for 3D ICs using network flow methods.
- We resolve overflowed bins by finding the shortest augmenting path on a 3D grid graph, utilizing an effective branch-and-bound algorithm. A post-optimization with cycle-canceling is proposed to reduce maximum displacement.
- Experimental results on ICCAD 2022 and 2023 contest benchmarks demonstrate 3D-Flow achieves up to 13% and 43% less average and maximum displacement compared to SOTA 2D legalizers in a similar runtime.

II. Preliminaries

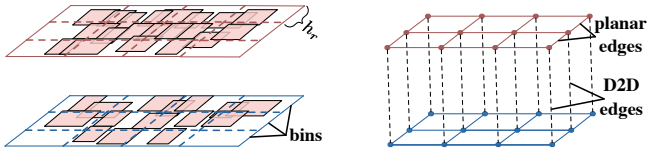
A. 3D ICs

3D integration enables higher transistor density on the silicon and replaces the long 2D interconnects with shorter inter-die connections. There are mainly three types of 3D ICs: through-silicon-via (TSV) based, monolithic, and face-to-face (F2F) bonding. Fig. 2 illustrates the 3D stacking for typical 3D ICs.

TSVs [21] occupy a significant amount of silicon area and introduce large parasitics, limiting TSV-based 3D ICs to a few inter-die connections. While monolithic inter-tier vias (MIVs) exhibit nanoscale sizes and enable fine-grain 3D interconnects, monolithic 3D ICs [22] require sequential fabrication of multiple dies, resulting in low yield and high cost. F2F bonded 3D ICs [23], [24] consist of two prefabricated dies, where the top die is flipped and stacked on top of the bottom die using hybrid bonding terminals on the topmost metal layers. The ease of fabrication and small size of bonding terminals enable a high integration density at a low cost. Additionally, prefabricated dies can adopt different technology nodes to achieve cost-effective performance, making F2F bonding a preferred approach. Recent ICCAD contests [25], [26] have addressed 3D placement for F2F bonded 3D ICs with heterogeneous technology integration. Therefore, we evaluate our 3D-Flow legalizer in F2F bonded settings, but our legalization algorithm is sufficiently general to apply to other types of 3D ICs with more than two dies.

B. Flow-based Legalization Formulation

We focus on standard cell legalization after global placement for true-3D placers, where cells are positioned in a continuous 3D space with their die assignments yet to be determined. The macros have been placed on their corresponding dies without any overlap. Specifically, we adopt the F2F bonded stacking specified in the ICCAD 2022 and 2023 contests [25], [26]. The cells must be placed on either the top or bottom die, aligned to placement rows and sites, and without any overlap.



(a) Initial Bin Assignment (b) 3D Grid Graph

Fig. 3 Flow network construction. (a) Cells are assigned to their nearest bins based on global placement. (b) Each bin is modeled as a vertex. Adjacent bins on the same die are connected by planar edges. And the bins on different dies with planar overlap are connected by die-to-die (D2D) edges.

TABLE I The main notations used in this paper.

Notations	Descriptions
C	the set of standard cells (indexed by c)
(x'_c, y'_c)	initial low-left corner of cell c
(x_c, y_c)	current low-left corner of cell c
w_c	width of cell c (w_c^+ for top die, w_c^- for bottom die)
V	the set of bins (indexed by v)
(x_v, y_v)	low-left corner of bin v
w_v	width of bin v
$\Gamma(v)$	the set of (fractional) cells assigned to bin v
$D_c(v)$	displacement of cell c assigned to bin v

The main notations used in this paper are listed in TABLE I. The top and bottom dies are divided into horizontal placement rows with a row height denoted as h_r. Note that if heterogeneous technology integration is adopted, the row heights may differ for the top die (h_r^+) and the bottom die (h_r^-). For simplicity, we will omit the superscript when there is no ambiguity. The height of a standard cell is equal to the height of a row on its corresponding die. We treat the macros as blockages, hence each row may be segregated into several segments. And each segment is divided into uniform bins with a bin width denoted as w_v. Initially, all the movable cells C are assigned to their nearest dies and then to their nearest bins based on the global placement, as illustrated in Fig. 3(a). Cells are allowed to be fractionally assigned to two bins if the bins are on the same die and horizontally adjacent. Hence, the objects that we move are fractional cells $\gamma = (c_\gamma, \rho_\gamma)$, where $c_\gamma = c$ and $\rho_\gamma \in (0, 1]$, and we have $\sum_{\gamma: c_\gamma = c} \rho_\gamma = 1$ for any cell c. We denote the bin set as V. Adjacent bins on the same die are connected by planar edges, while bins on different dies with planar overlap are connected by die-to-die (D2D) edges. Thus, we construct a 3D grid graph $G(V, E)$, as illustrated in Fig. 3(b).

For a bin v, let $\text{cap}(v) = w_v$ represent the free capacity that can be used by standard cells. Based on the initial bin assignment, let $\Gamma(v)$ be the set of fractional cells assigned to bin v. We can compute the total width of fractional cells as $w_{\Gamma(v)} = \sum_{\gamma \in \Gamma(v)} w_{c_\gamma} \times \rho_\gamma$, where w_{c_γ} represents the width for cell c_γ. Then we can define the supply value for a bin v:

$$\text{sup}(v) = \max\{0, w_{\Gamma(v)} - w_v\}, \tag{1}$$

similarly, we define the demand value for a bin v:

$$\text{dem}(v) = \max\{0, w_v - w_{\Gamma(v)}\}. \tag{2}$$

If $\text{sup}(v) > 0$, the bin v is overflowed and is treated as a source bin. If $\text{dem}(v) > 0$, the bin v is treated as a sink bin. The goal is to find an assignment of cells to bins such that no bin is overflowed while minimizing the average and maximum cell displacement.

III. Proposed 3D-Flow Legalizer

The overall legalization flow of our 3D-Flow legalizer is shown at the end of this section (Algorithm 2). We first give the mathematical model for flow-based legalization in Section III-A. The core part of our legalizer involves finding the shortest augmenting path on the 3D grid graph using a branch-and-bound algorithm (Section III-B). Our legalizer iteratively augments flows along these paths (Section III-C) to resolve overflowed bins. Once we have determined an assignment of cells to bins, we can legalize all cells within their respective rows (Section III-D). Additionally, our legalizer adopts an incremental legalization step with a cycle-canceling algorithm (Section III-E) to reduce the maximum cell displacement.

A. Legalization as a Minimum-Cost Flow Problem

Based on the flow network $G(V, E)$, we can model the legalization as an assignment problem. The goal is to resolve all overflowed bins while minimizing the total cell displacement. Note that fractional cells are not applicable in this discrete assignment formulation. We should assign the complete cells to their nearest bins. Thus, we use $c \in \Gamma(v)$ to denote a cell c in a bin v instead of fractional cells. Let $S = \{v \mid v \in V, \text{sup}(v) > 0\}$ represent the source bins, and $T = \{v \mid v \in V, \text{dem}(v) > 0\}$ represent the sink bins. Let $R = V - S - T$ denote the remaining bins. We can have the following integer linear programming (ILP) formulation:

$$
\begin{aligned}
\min \quad & \sum_{(u,v)\in E} \sum_{c\in\Gamma(u)} \text{cost}_{u,v,c} a_{u,v,c} \\
\text{s.t.} \quad & \sum_{u\in V} \sum_{c\in\Gamma(v)} w_c a_{v,u,c} - \sum_{u\in V} \sum_{c\in\Gamma(u)} w_c a_{u,v,c} \geq \text{sup}(v), \quad \forall v \in S \\
& \sum_{u\in V} \sum_{c\in\Gamma(u)} w_c a_{u,v,c} - \sum_{u\in V} \sum_{c\in\Gamma(v)} w_c a_{v,u,c} \leq \text{dem}(v), \quad \forall v \in T \\
& \sum_{u\in V} \sum_{c\in\Gamma(u)} w_c a_{u,v,c} - \sum_{u\in V} \sum_{c\in\Gamma(v)} w_c a_{v,u,c} = 0, \quad \forall v \in R \\
& a_{u,v,c} \in \{0,1\}, \quad \forall (u,v) \in E, \forall c \in \Gamma(u),
\end{aligned}
\tag{3}
$$

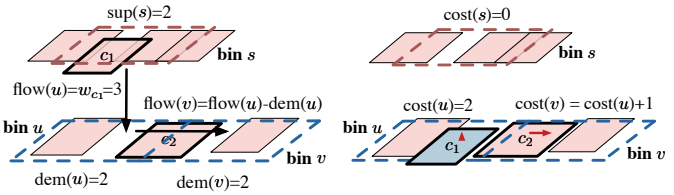
where $a_{u,v,c} = 1$ indicates the cell $c \in \Gamma(u)$ is moved from bin u to bin v; otherwise, the cell remains in bin u. The cost for such movement is calculated as:

$$D_c(v) = |x_c - x'_c| + |y_c - y'_c|, \tag{4}$$

$$\text{cost}_{u,v,c} = D_c(v) - D_c(u). \tag{5}$$

$D_c(v)$ is the estimated displacement when cell c is assigned to bin v. Since bins are aligned to placement rows, we have $y_c = y_v$, y-coordinate of bin v. And x_c is set as the nearest horizontal position within bin v based on the x-coordinate of global placement x'_c. For example, if $x'_c < x_v$, then we have $x_c = x_v$.

According to Equation (3), if w_c is constant for all cells, the optimization problem is a minimum-cost flow problem, which can be solved in $O(|V|\log|V|(|E|+|V|\log|V|))$. However, in practice, we have different cell sizes, resulting in an NP-hard problem. Given the sheer number of cells in modern designs, solving the ILP problem directly is impractical for legalization. One approach is to divide each cell into partial cells with unit width to enable a minimum-cost flow formulation. However, the resulting flow may be unattainable and even misleading [9]. The successive shortest path algorithm for solving minimum-cost flow [14] is appropriate for our application, as we can modify it to ensure that only augmentations that can be exactly realized by cell movements are considered, avoiding the flow realization issue.



(a) Candidate Path (b) Cell Movement

Fig. 4 Path augmentation. (a) Candidate path with corresponding flow. (b) Cell movement with corresponding cost. A candidate path $s-u-v$ for solving the overflow in source bin s is shown in (a). At each step, the cells with minimal cost are selected to move. Since s and u belong to different dies, cell c_1 is moved completely. And u and v are horizontally adjacent, cell c_2 is moved partially to minimize displacement cost.

Algorithm 1 Augmentation with Branch and Bound

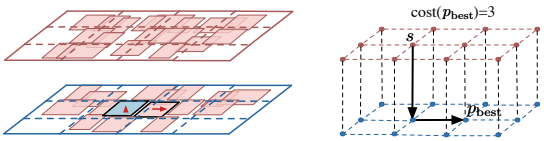
Input: Flow network $G(V, E)$, a supply bin s.
Output: The shortest augmenting path p_{best} resolving overflow.

```
1: flow(s) = sup(s); cost(s) = 0; parent(s) = nullptr;
2: visited(s) = true;
3: Q.insert(s)                          ▷ priority queue Q sorted by cost(·);
4: while Q ≠ ∅ do
5:     u = Q.pop()                      ▷ bin u as source bin;
6:     for each bin v connected to bin u do   ▷ bin v as sink bin
7:         if visited(v) then           ▷ visit more than once
8:             continue;
9:         end if
10:        outFlow, cost = computeFlow(u, v);   ▷ Equation (6)
11:        flow(v) = outFlow; cost(v) = cost(u) + cost;
12:        parent(v) = u; visited(v) = true;
13:        if cost(v) < (1 + α)cost(p_best) then;   ▷ set the bound
14:            if flow(v) ≤ dem(v) then;   ▷ a candidate path
15:                if cost(v) < cost(p_best) then
16:                    p_best = v;      ▷ update the shortest path
17:                end if
18:            else
19:                Q.insert(v);         ▷ insert intermediate path
20:            end if
21:        end if
22:    end for
23: end while
24: return p_best;
```

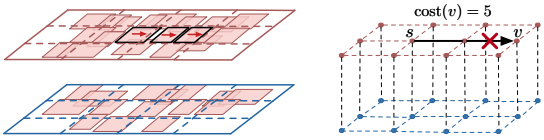
B. Augmentation with Branch-and-Bound

The traditional successive shortest path algorithm [14] begins with a zero flow and iteratively employs Dijkstra's algorithm to identify shortest paths between supply node u and demand node v in the residual graph. However, this approach faces two significant limitations in our application. First, the repeated call of Dijkstra's algorithm becomes computationally expensive for large-scale designs. Second, during iterative optimization, the cost function eq. (5) can yield negative values, indicating that we can move cells toward their initial positions. But Dijkstra's algorithm requires non-negative edge costs. To address these limitations, we propose an enhanced algorithm that combines a modified breadth-first search (BFS) with branch-and-bound techniques to find the shortest augmenting path in the 3D grid graph, achieving high solution quality with better efficiency.

Algorithm 1 presents our path augmentation algorithm. Since our algorithm ensures that each bin is traversed only once (line 7), an n-ary search tree rooted at the supply bin s is constructed during the



(a) Shortest Augmenting Path



(b) Eliminated Branch

Fig. 5 Augmentation with branch and bound. (a) Augmenting path p_{best} with the smallest cost. (b) Eliminated branch with large cost. Moved cells are highlighted with red arrows indicating displacement. The **blue**-colored cell in (a) is from top die. The intermediate paths with large cost are stopped from being explored.

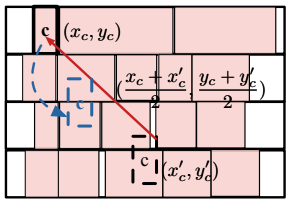
search process. Each branch of the tree represents a candidate path for moving cells to resolve overflow. For each bin v in the search tree, we define parent(v) as the predecessor of v, flow(v) as the required flow that needs to be moved from bin parent(v) to bin v, and cost(v) as the total displacement cost for the path from root s to bin v. For the root bin s, the required flow is set as the supply value sup(s). Unlike the typical BFS algorithm, we prioritize bins with lower displacement costs when extending the search path (line 5).

A step that is critical for the algorithm is the selection of the set $C(u, v)$ of fractional cells to be moved from u to v in line 10, which should resolve overflow with minimal displacement cost. To ensure that no new overflowed bin is created, we compute the $outFlow$ as follows:

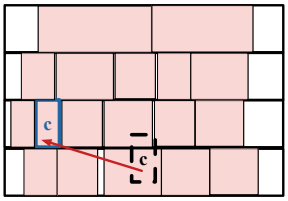
$$outFlow = w_{C(u,v)} \geq \text{flow}(u) - \text{dem}(u), \tag{6}$$

where dem(u) represents the flow that can be sunk by bin u. Thus, $outFlow$ in Equation (6) is the flow that needs to be moved out when flow(u) is moved into bin u. If bins u and v are horizontally adjacent in the same die, we can choose the fractional cell $\gamma \in \Gamma(u)$ with $\rho_\gamma < 1$ to ensure that $w_{C(u,v)} = \text{flow}(u) - \text{dem}(u)$. Specifically, we sort the fractional cells in $\Gamma(u)$ by the cost in Equation (5), which reflects the displacement cost for such movement. The cheapest fractional cells are then selected to reduce their ρ_γ and the path cost is updated accordingly (line 11). On the other hand, if u and v belong to different rows or dies, we can only move cells completely because no cell can be distributed across different segments. Hence, moving γ from u to v means removing all parts of c_γ from their bins (which must be in the same segment as u) and assigning them to v. In these cases, we select the cells such that $w_{C(u,v)} \geq \text{flow}(u) - \text{dem}(u)$ with the lowest displacement cost. If the flow(v) can be completely sunk by bin v, a candidate path for resolving the overflow is found (line 14). An example of one candidate path is illustrated in Fig. 4.

We keep track of the path p_{best} with the smallest cost (line 16). During the search process, any branch with a cost greater than $(1 + \alpha) \times \text{cost}(p_{\text{best}})$ will be eliminated (line 13), where $\alpha \geq 0$, controlling the solution space that our algorithm explores. When $\alpha = 0$, the algorithm becomes a greedy search, including only intermediate paths with costs smaller than the current best cost. Conversely, a large α will cause the algorithm to traverse the entire search tree. Although there are edges with negative costs, such edges are few. Thus, a small $\alpha = 0.1$ can help our algorithm find the shortest augmenting path with great efficiency in practice. Fig. 5 shows an example that a branch with large cost is eliminated.



(a) Negative Cycle



(b) Reduced Displacement

Fig. 6 Post-optimization with cycle-canceling. (a) Negative cycle. (b) Reduced maximum displacement. The black dashed box indicates the initial position. Our algorithm only selects cells with displacement larger than the predifined threshold to move toward their initial positions. Our incremental legalization will reduce the maximum displacement while making minimal perturbation to other cells.

Note that BonnPlaceLegal [10] also adopts a flow-based methodology. However, it utilizes vanilla Dijkstra's algorithm to find the augmenting path on a 2D grid graph, which is not applicable for negative-cost movements. Based on the 3D flow network, our algorithm can move cells freely both intra-die and inter-die to minimize the total displacement with improved computational efficiency.

C. Moving Cells Along Augmenting Path

Once the shortest augmenting path is found by Algorithm 1, the set of fractional cells $C(u, v)$ are moved between the adjacent bins along the path to resolve the overflow. The cell movement is performed by backtracking the path from the leaf bin p_{best} to the root supply bin s. For each current bin v, we apply the same method as in Algorithm 1 to select the set of fractional cells $C(u, v)$ from the parent bin $u = \text{parent}(v)$ with the minimal cost, where $w_{C(u,v)} = \text{flow}(v)$. The procedure repeats until reaching the root supply bin s, thereby resolving the overflow in bin s.

D. Legalization Within a Row

After the flow-based legalization, cells are assigned to bins aligned with placement rows. However, the overlap between cells within a bin may still exist. Many well-known row-based placement algorithms [4], [27], [28] can be used to remove the overlap between cells within a row. We use the `PlaceRow` algorithm in Abacus [4], which efficiently minimizes quadratic cell movement in linear time.

E. Post-Optimization with Cycle-Canceling

At the final step, we propose a post-optimization technique inspired by the cycle-canceling algorithm [29] to minimize maximum displacement. The cycle-canceling algorithm improves the objective function by finding negative cost cycles in the residual graph and augmenting the flow on these cycles. In our application, we construct such negative cycles by moving cells with large displacement toward their initial positions. Specifically, we select the cells with displacement $D_c > threshold$, where $threshold = \max\{5h_r, \frac{D_{\max}}{2}\}$, h_r is the row height and $D_{\max}$ is the current maximum displacement. For each selected cell c, we reposition it at the midpoint between its current and initial positions $(\frac{x_c + x'_c}{2}, \frac{y_c + y'_c}{2})$. This movement may introduce a few overflowed bins, and we perform the flow-based legalization again and remove all the cell overlap. Fig. 6 shows an example of the proposed cycle-canceling algorithm. Our flow-based legalizer enables incremental legalization inherently, therefore, the post-optimization can significantly reduce maximum displacement with minimal perturbation to other cells.

Algorithm 2 3D-Flow Legalizer

```
Divide the layout into bins V;
Assign cells C to their nearest bins;
Connect neighboring bins by edges in a 3D grid graph G;
Q ← {v ∈ V | sup(v) > 0};                    ▷ priority queue sorted by sup(·)
while Q ≠ ∅ do
    s = Q.pop();                              ▷ bin s as source bin
    Call Augmentation path algorithm;         ▷ Algorithm 1
    Move cells;                               ▷ Section III-C
    Update overflowed bins in Q;
end while
for each placement row r do
    PlaceRow(r);                              ▷ Section III-D
end for
Post-optimization with cycle-canceling;       ▷ Section III-E
```

F. Overall Algorithm

Algorithm 2 provides an overview of the proposed 3D-Flow legalizer. We process all supply bins in descending order based on their supply values. For each supply bin s, we identify the augmenting path with minimal cost using Algorithm 1 and adjust cell positions accordingly to resolve overflow. This process iterates until all overflow is eliminated.

The choice of bin width w_v involves a trade-off between result quality and efficiency. For the flow-based legalization, we set the bin width $w_v = 10\bar{w}_c$ to enhance efficiency, where $\bar{w}_c$ is the average width of all cells. During post-optimization, w_v is reduced to $5\bar{w}_c$ for more precise cost estimation, given the limited number of overflowed bins. Our algorithm is flexible enough to handle various constraints. In heterogeneous technology integration, cell movement across die-to-die edges requires updating cell widths. We manage different widths w_c^+ on the top die and w_c^- on the bottom die, ensuring the utilization rate of each die is not exceeded.

$$\text{cost}_{u,v,c} = D_c(v) - D_c(u) + \text{sup}(v) - \text{dem}(v). \tag{7}$$

We set the die-to-die edge cost as shown in Equation (7) to encourage cells to move from the congested die while penalizing unnecessary cross-die movements.

IV. Experimental Results

A. Experimental Setup

To validate our proposed 3D legalizer, we conducted experiments on ICCAD 2022 [25] and 2023 [26] contest benchmarks, which address standard-cell placement and mixed-size placement for F2F bonded 3D ICs, respectively. The detailed benchmark statistics are shown in TABLE II. The global placements for ICCAD 2022 and 2023 contest benchmarks were generated by SOTA true-3D placers [18] and [19], respectively. We implemented the proposed 3D legalizer in C++. All the experiments were performed on a Linux machine with 20-core Intel Xeon Silver 4210R CPU (2.40GHz) and 24GB RAM.

B. Comparison with SOTA Legalizers

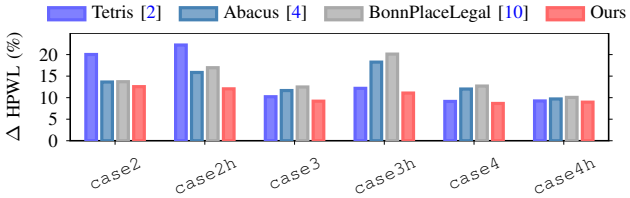
We compared 3D-Flow legalizer with existing SOTA 2D legalizers including Tetris [2], Abacus [4], and BonnPlaceLegal [10]. Tetris and Abacus are the most popular legalization algorithms based on greedy search, which are extensively used in modern global placers [6], [7], [18]–[20]. And BonnPlaceLegal is a competitive flow-based legalizer.

The experimental results are shown in TABLE III and TABLE IV. For ICCAD 2022 contest benchmarks, which contain only standard cells, legalizers have more flexibility to reposition cells without large macros as blockages. Our 3D-Flow legalizer consistently obtained the best results for all the cases, as shown in TABLE III. Compared to Tetris, it achieved 1.61× less average displacement and 2.31× less maximum displacement. Tetris places cells sequentially based on their x-position, resulting in larger displacement for cells placed later. And our legalizer reduced average displacement by 13% and maximum displacement by 54% compared to Abacus. Although Abacus allows movement of already placed cells, they remain on a fixed row, causing significant displacement for later placements. Moreover, 3D-Flow outperformed BonnPlaceLegal by 15% and 43% less average and maximum displacement. While BonnPlaceLegal uses a flow-based method, it limits edge costs to positive values, missing further optimization opportunities. In contrast, our 3D-Flow repositions cells in a 3D solution space, fully utilizing the vertical stacking of 3D ICs. Leveraging the global view in the network flow method, our legalizer allows simultaneous movement of multiple cells across rows and dies to minimize displacement. The post-optimization with cycle-canceling further reduces maximum displacement while maintaining quality.

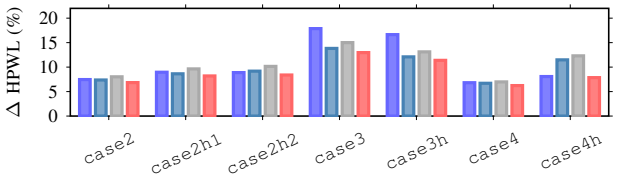
For ICCAD 2023 contest benchmarks, large macros divide placement rows into segments, posing challenges in finding available space near macro boundaries. Despite this, our 3D-Flow legalizer outperformed all SOTA legalizers, as demonstrated in TABLE IV. It achieved 1.46×, 1.08×, and 1.13× less average displacement, and 2.97×, 1.40×, and 1.34× less maximum displacement compared to Tetris, Abacus, and BonnPlaceLegal, respectively.

TABLE II The statistics of ICCAD 2022 [25] and ICCAD 2023 [26] contest benchmarks. h_r^+ and h_r^- represent row height values of the top and bottom die, respectively.

ICCAD 2022	#Cells	#Macros	#Nets	h_r^+	h_r^-
case2	2735	0	2644	176	252
case2h	2735	0	2644	252	252
case3	44764	0	44360	115	115
case3h	44764	0	44360	92	115
case4	220845	0	220071	92	115
case4h	220845	0	220071	103	115
ICCAD 2023	**#Cells**	**#Macros**	**#Nets**	h_r^+	h_r^-
case2	13901	6	19547	33	33
case2h1	13901	6	19547	33	48
case2h2	13901	6	19547	33	48
case3	124231	34	164429	33	48
case3h	124231	34	164429	36	48
case4	740211	32	758860	92	115
case4h	740211	32	758860	55	69



(a) Δ HPWL (%) increase on ICCAD 2022 benchmarks [25].



(b) Δ HPWL (%) on ICCAD 2023 benchmarks [26].

Fig. 7 HPWL increase (%) of global placement and legal placement for different methods. (a) ICCAD 2022 contest benchmarks. (b) ICCAD 2023 contest benchmarks.

TABLE III Comparison on the legalization results of our 3D-Flow legalizer with existing SOTA legalizers for ICCAD 2022 benchmarks. **RT** (s) stands for the total runtime including file IO. The average displacement and maximum displacement are normalized by the row height.

ICCAD 2022	Tetris [2]			Abacus [4]			BonnPlaceLegal [10]			Ours		
	Avg. Disp.	Max. Disp.	RT	Avg. Disp.	Max. Disp.	RT	Avg. Disp.	Max. Disp.	RT	Avg. Disp.	Max. Disp.	RT
case2	0.801	7.27	0.23	0.522	3.05	0.24	0.511	2.91	0.26	**0.503**	**2.77**	0.27
case2h	1.091	5.77	0.22	0.593	4.34	0.24	0.610	4.80	0.24	**0.511**	**2.97**	0.25
case3	0.929	13.40	1.15	0.705	5.81	1.35	0.727	6.13	2.70	**0.660**	**4.53**	1.54
case3h	1.162	8.30	1.19	0.938	9.47	1.44	0.986	7.95	6.28	**0.750**	**4.61**	1.73
case4	1.775	12.83	6.79	1.383	16.81	10.98	1.428	15.13	72.93	**1.174**	**11.25**	9.69
case4h	1.622	27.59	6.75	1.161	14.96	9.96	1.186	11.94	53.68	**1.100**	**8.14**	10.36
Average	1.613	2.31	**0.76**	1.125	1.54	0.95	1.153	1.43	3.34	**1.000**	**1.00**	1.00

TABLE IV Comparison on the legalization results of our 3D-Flow legalizer with existing SOTA legalizers for ICCAD 2023 benchmarks. **RT** (s) stands for the total runtime including file IO. The average displacement and maximum displacement are normalized by the row height.

ICCAD 2023	Tetris [2]			Abacus [4]			BonnPlaceLegal [10]			Ours		
	Avg. Disp.	Max. Disp.	RT	Avg. Disp.	Max. Disp.	RT	Avg. Disp.	Max. Disp.	RT	Avg. Disp.	Max. Disp.	RT
case2	3.029	13.27	0.50	2.199	11.33	0.61	2.308	11.50	2.49	**2.109**	**9.09**	0.64
case2h1	3.609	50.28	0.57	2.509	35.28	0.68	2.668	36.85	4.80	**2.433**	**27.03**	0.74
case2h2	3.709	46.29	0.55	2.673	39.26	0.71	2.828	34.11	5.19	**2.518**	**27.73**	0.74
case3	3.612	195.31	3.07	2.475	136.25	4.52	2.615	131.96	59.83	**2.442**	**70.82**	4.46
case3h	3.041	141.31	2.89	2.266	119.31	4.21	2.370	109.11	69.99	**2.204**	**99.05**	5.74
case4	1.349	127.69	22.86	1.000	17.32	37.79	1.023	15.32	264.23	**0.920**	**13.77**	39.13
case4h	1.709	128.19	23.75	1.432	78.55	47.02	1.492	77.71	315.26	**1.130**	**54.31**	37.73
Average	1.461	2.97	**0.69**	1.076	1.40	1.00	1.127	1.34	8.89	**1.000**	**1.00**	1.00

TABLE V Comparison on the legalization results for our 3D-Flow legalizer with and without die-to-die (D2D) cell movement on ICCAD 2023 contest benchmarks. #Move stands for the number of cells moved across dies.

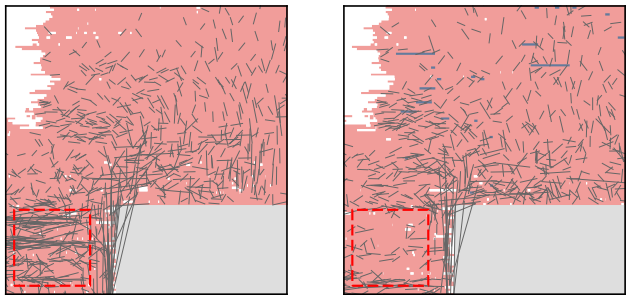
ICCAD 2023	w/o. D2D		Ours		
	Avg. Disp.	Max. Disp.	Avg. Disp.	Max. Disp.	#Move
case2	2.197	11.33	**2.109**	**9.09**	35
case2h1	2.506	35.28	**2.433**	**27.03**	46
case2h2	2.711	32.10	**2.518**	**27.73**	54
case3	2.582	98.98	**2.442**	**70.82**	493
case3h	2.229	107.84	**2.204**	**99.05**	373
case4	0.931	15.32	**0.920**	**13.77**	2479
case4h	1.358	54.33	**1.130**	**54.31**	2081
Average	1.068	1.19	**1.000**	**1.00**	-

The HPWL increase results for ICCAD 2022 and 2023 contest benchmarks are shown in Fig. 7. Although HPWL increase may not always align with displacement results, our 3D-Flow consistently achieved the best outcomes, with 1.29×, 1.30×, and 1.38× less increase for ICCAD 2022 contest benchmarks, and 1.17×, 1.13×, and 1.22× less increase for ICCAD 2023 contest benchmarks, compared to Tetris, Abacus, and BonnPlaceLegal, respectively.

Regarding runtime, Tetris is the fastest due to its simple procedure, being 1.32× and 1.45× faster than our method on ICCAD 2022 and 2023 contest benchmarks. And 3D-Flow achieved better results with similar runtime compared to Abacus. However, BonnPlaceLegal showed poor scalability in large cases. Leveraging our efficient branch-and-bound algorithm, 3D-Flow achieved 3.34× and 8.89× speedup over BonnPlaceLegal on ICCAD 2022 and 2023 contest benchmarks, respectively.

C. Ablation Study on Die-to-Die Cell Movement

To validate the effectiveness of die-to-die movement in our 3D legalizer, we performed experiments on ICCAD 2023 contest benchmarks for our 3D-Flow legalizer with and without die-to-die cell movement. The experimental results are shown TABLE V. In the last column, we list the number of cells moved across dies. By moving only a few cells vertically, our legalizer finds white space for other cells to minimize displacement in a global view. Our 3D-Flow legalizer outperformed its 2D counterpart by 6.8% and 19% less average and maximum displacement. The displacement of different settings for case3 is visualized in Fig. 8.



(a) w/o. D2D cell movement (b) Our 3D-Flow Legalizer

Fig. 8 Displacement visualization for the top die of case3 in ICCAD 2023 contest benchmarks. (a) Without die-to-die (D2D) cell movement. (b)Using our 3D legalizer. The gray block represents the macro. The **blue**-colored cells in (b) are from the bottom die. Black lines indicate cell displacement.

V. Conclusion

This paper introduces 3D-Flow, the first 3D legalizer designed to minimize cell displacement for 3D ICs using network flow methods. Leveraging a novel branch-and-bound algorithm, 3D-Flow efficiently identifies the shortest augmenting path on a 3D grid graph to iteratively resolve overflowed bins. By fully utilizing the vertical stacking of 3D ICs, our legalizer finds legal positions for cells with minimal displacement in a global view. Additionally, a post-optimization with a cycle-canceling algorithm further reduces maximum displacement while preserving result quality. Experimental results on ICCAD 2022 and 2023 benchmarks show that our 3D legalizer achieves up to 13% and 43% less average and maximum displacement, respectively, compared to state-of-the-art legalizers, within a similar runtime.

Acknowledgements

The project is supported in part by Research Grants Council of Hong Kong SAR (No. RFS2425-4S02 and No. CUHK14211824), and the MIND project (MINDXZ202404).

REFERENCES

[1] G. Murali and S. K. Lim, "Heterogeneous 3D ICs: Current status and future directions for physical design technologies," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2021, pp. 146–151.

[2] D. Hill, "Method and system for high speed detailed placement of cells within an integrated circuit design," Apr. 9 2002, US Patent 6,370,673.

[3] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2007, pp. 135–140.

[4] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: fast legalization of standard cell circuits with minimal movement," in *ACM International Symposium on Physical Design (ISPD)*, 2008, pp. 47–53.

[5] W.-K. Chow, J. Kuang, X. He, W. Cai, and E. F. Y. Young, "Cell density-driven detailed placement with displacement constraint," in *ACM International Symposium on Physical Design (ISPD)*, 2014, pp. 3–10.

[6] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 12, pp. 1914–1927, 2014.

[7] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[8] U. Brenner and J. Vygen, "Legalizing a placement with minimum total movement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 23, no. 12, pp. 1597–1613, 2004.

[9] M. Cho, H. Ren, H. Xiang, and R. Puri, "History-based vlsi legalization using network flow," in *ACM/IEEE Design Automation Conference (DAC)*, 2010, pp. 286–291.

[10] U. Brenner, "Bonnplace legalization: Minimizing movement by iterative augmentation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no. 8, pp. 1215–1227, 2013.

[11] N. K. Darav, I. S. Bustany, A. Kennings, and L. Behjat, "A fast, robust network flow-based standard-cell legalization method for minimizing maximum movement," in *ACM International Symposium on Physical Design (ISPD)*, 2017, pp. 141–148.

[12] J. Monteiro, M. Johann, and L. Behjat, "An optimized cost flow algorithm to spread cells in detailed placement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 3, pp. 1–16, 2019.

[13] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia, "Diffusion-based placement migration," in *ACM/IEEE Design Automation Conference (DAC)*, 2005, pp. 515–520.

[14] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by successive approximation," *Mathematics of Operations Research*, vol. 15, no. 3, pp. 430–466, 1990.

[15] K. Chang, S. Sinha, B. Cline, R. Southerland, M. Doherty, G. Yeric, and S. K. Lim, "Cascade2D: A design-aware partitioning approach to monolithic 3D IC with 2D commercial tools," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.

[16] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Shrunk-2-D: A physical design methodology to build commercial-quality monolithic 3-D ICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 10, pp. 1716–1724, 2017.

[17] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "TSV-aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no. 4, pp. 497–509, 2013.

[18] P. Liao, Y. Zhao, D. Guo, Y. Lin, and B. Yu, "Analytical die-to-die 3d placement with bistratal wirelength model and gpu acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[19] Y. Zhao, P. Liao, S. Liu, J. Jiang, Y. Lin, and B. Yu, "Analytical heterogeneous die-to-die 3d placement with macros," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[20] Y.-J. Chen, C.-H. Hsieh, P.-H. Su, S.-H. Chen, and Y.-W. Chang, "Mixed-size 3d analytical placement with heterogeneous technology nodes," in *ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[21] M. Motoyoshi, "Through-silicon via (TSV)," *Proceedings of the IEEE*, vol. 97, no. 1, pp. 43–48, 2009.

[22] S. S. K. Pentapati and S. K. Lim, "Heterogeneous monolithic 3d ics: Eda solutions, and power, performance, cost tradeoffs," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 925–930.

[23] C. Premachandran, J. Lau, L. Xie, A. Khairyanto, K. Chen, M. E. P. Pa, M. Chew, and W. K. Choi, "A novel, wafer-level stacking method for low-chip yield and non-uniform, chip-size wafers for mems and 3d sip applications," in *IEEE Electronic Components and Technology Conference*. IEEE, 2008, pp. 314–318.

[24] S. Liu, J. Jiang, Z. He, Z. Wang, Y. Lin, B. Yu, and M. Wong, "Routing-aware legal hybrid bonding terminal assignment for 3d face-to-face stacked ics," in *ACM International Symposium on Physical Design (ISPD)*, 2024, pp. 75–82.

[25] K.-S. Hu, I.-J. Lin, Y.-H. Huang, H.-Y. Chi, Y.-H. Wu, and C.-F. C. Shen, "2022 ICCAD CAD Contest Problem B: 3D placement with D2D vertical connections," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–5.

[26] K.-S. Hu, H.-Y. Chi, I.-J. Lin, Y.-H. Wu, W.-H. Chen, and Y.-T. Hsieh, "2023 ICCAD CAD Contest Problem B: 3D placement with macros," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[27] A. B. Kahng, P. Tucker, and A. Zelikovsky, "Optimization of linear placements for wirelength minimization with free sites," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE, 1999, pp. 241–244.

[28] A. B. Kahng, I. L. Markov, and S. Reda, "On legalization of row-based placements," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2004, pp. 214–219.

[29] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," *Journal of the ACM (JACM)*, vol. 36, no. 4, pp. 873–886, 1989.

A Post-Implementation Performance Prediction Method with HLS Optimization Directives

Jingyu Zhu, Yan Ding*, Lu Xiao, Kenli Li, Chubo Liu, and Zheng Xiao*
College of Information Science and Engineering, Hunan University, China
{zhujingyu, ding, dalu, lkl, liuchubo, zxiao}@hnu.edu.cn

***Abstract*—High-Level Synthesis (HLS) offers various optimization directives that enable designers to flexibly adjust hardware microarchitecture. However, existing HLS performance prediction methods typically rely on control data flow graphs generated (CDFG) from original HLS C/C++, which struggle to capture the complex interactions between directives and the resulting hardware resource reuse issues. To address these issues, this paper proposes a post-implementation performance prediction method tailored for directive-optimized circuits design, which utilizes a Graph Builder to integrate directive optimization and resource reuse information into the graph representation. In addition, the performance prediction model, which integrates a TransformerConv-based graph neural network (GNN) and an aggregation pool module, effectively captures key features related to post-implementation performance. Experimental results show that our method can reduce the prediction error of critical path delay (CP), power, and resource utilization to 3.87%∼8.08%, significantly outperforming existing state-of-the-art methods. It also demonstrates excellent generalization on unseen kernels, providing a more effective and accurate performance prediction tool for HLS.**

***Index Terms*—High-Level Synthesis, Directive-Optimized, Resource Reuse, Performance Prediction Model.**

I. Introduction

With the rapid development of 5G, autonomous driving, and large model technologies, hardware accelerator are playing a crucial role in enhancing computational performance and efficiency [1]. However, hardware programming typically requires extensive hardware knowledge and expert experience, along with significant effort to write Register-transfer-level(RTL) code, making the process both time-consuming and error-prone. To improve hardware development efficiency, High-Level Synthesis (HLS) technology has emerged as a solution.

HLS enables designers to create RTL hardware designs at a higher level of abstraction using programming languages such as C/C++, and to optimize micro-architecture configurations through various optimization directives [2]. However, the interactions between optimization directives are complex, and simply increasing the number of directives does not necessarily lead to better design outcomes. As a result, designers must carefully balance and select the optimal combination of directives from a vast design space, which introduces new challenges to design optimization. **First**, as the number of optimization directives increases, the design space expands exponentially, making it increasingly difficult to find the optimal design. **Second**, using HLS tools to evaluate the performance of hardware implementations is highly time-consuming, as the process involves layout, routing, and other post-implementation steps. This time-consuming evaluation process further slows down the progress of design space exploration. Hence, the introduction of a fast and accurate performance prediction proxy model as a substitute for HLS tools becomes crucial.

Existing performance prediction models can be categorized into analytical models and learning-based models. Analysis models such as Lin-analyzer [3] and Comba [4] rely on extensive hardware knowledge and an understanding of regularized data flow designs. However, traditional analysis models often require substantial time to obtain necessary parameters, such as instruction latency, and fails to capture the heuristic optimization rules within HLS tools.

To address these limitations, researchers in [5] proposed a learning-based model for performance prediction. Recent work [6] has further advanced this direction by using control data flow graphs (CDFG) in combination with GNNs to predict design quality. GNN-DS [7] and HARP [8] directly insert directive nodes into the graph representation to improve performance prediction. However, this coarse-grained insertion approach has limitations: when applying the unroll directive, it may treat all expanded operations as a single node, making it challenging to capture the fine-grained impacts on execution order and resource utilization. In contrast, they [9] directly incorporates directive effects within a single for loop and uses a hierarchical GNN network to predict overall quality.

Nevertheless, existing methods, whether analysis or learning models, have yet to fully capture the complex effects of directive interactions and resulting resource reuse. Specifically, when *unroll* and *array partition* directives are applied together, parallelism and memory bandwidth are expected to improve. However, when multiple operations share the same hardware resource, this reuse can alter the intended scheduling order, impacting overall performance. Therefore, effectively incorporating directive interaction effects and resource reuse information into the graph representation, a more accurate reflection of operation scheduling and resource allocation in circuit design can be achieved, providing GNN models with richer design information and significantly enhancing the accuracy and reliability of performance prediction.

To address the above issues, this paper proposes a post-implementation performance prediction method tailored for directive-optimized circuit design. This approach not only

*Corresponding author.

achieves precise performance prediction but also demonstrates excellent generalization capability. Our main contributions are as follows:

1) We propose a Graph Builder that constructs a directive-optimized graph by consider the effects of directives and hardware resource reuse, aligning more closely with the actual operation scheduling and resource allocation in hardware circuits. To our knowledge, this is the first approach in HLS performance prediction that introduces hardware resource reuse to simulate circuit characteristics, enabling more accurate performance prediction.
2) Based on the constructed directive-optimized graph, we propose a post-implementation performance prediction model that fully considers the impact of edge features on node information. This model integrates TransformerConv and aggregation pool modules to capture graph-level features related to performance metrics, while incorporating global information to enhance prediction accuracy and generalization capability.
3) Experimental results demonstrate that the proposed method outperforms state-of-the-art models across five performance prediction metrics; Specifically, the MAPE for LUT, FF, CP, and Power, and the MAE for DSP, are reduced to 8.08%, 3.87%, 4.34%, 6.49%, and 0.11%, respectively. Furthermore, it exhibits strong generalization capabilities on unseen kernels. Ablation studies further indicate that incorporating hardware resource reuse dependencies reduces prediction error.

The remainder of this paper is organized as follows: Section 2 introduces some preliminary concepts. Section 3 details the proposed method, including the directive-optimized graph construction, node and edge feature encoding, and the prediction model. Section 4 presents an extensive experimental analysis. Section 5 presents the conclusions.

II. PRELIMINARIES

This section introduces the core foundational knowledge involved in this study, focusing on how programs are transformed into graph representations and the role of directives in HLS hardware design optimization. Additionally, we will discuss the principles of Graph Neural Networks and their application in hardware performance prediction. These concepts form the basis for the subsequent our approach and experiments.

A. Program-to-Graph and Directives

In HLS tools, such as Vitis HLS, the source program is first compiled by an LLVM-based [10] front-end to generate an intermediate representation (IR), and then a Control Data Flow Graph (CDFG) is constructed based on the relationships between nodes and blocks. This graph clearly represents the dependencies and execution order of various operations within the program. The graph structure based on LLVM IR makes the transformation from high-level languages to hardware designs more efficient, enabling designers to intuitively understand the execution logic of the program and analyze its hardware implementation details.

Furthermore, hardware design optimization involves the application of optimization directives (e.g., unroll, pipeline, array partition), which directly impact resource allocation and scheduling. To accurately capture the effects of these directives, they must be integrated into the graph structure to reflect actual resource allocation and scheduling. The directives involved in this paper are shown in Table I.

TABLE I: Configuration of directives.

Pragma	**Configuration**
Loop unrolling	Unrolling factors
Loop pipelining	Enabled/Disabled
Loop flattening	Yes/No
Array partitioning-type	Block/Cyclic/Complete
Array partitioning-factor	Partitioning factors
Function inlining	Yes/No

B. Graph Neural Networks

Graph Neural Networks [11] are a class of deep learning models specifically designed to process graph-structured data. By iterative propagation and aggregation of node information, GNNs learn feature representations of nodes and edges within the graph. Unlike traditional neural networks, GNNs can capture the topological structure of the graph and complex dependencies between nodes, making them well-suited for graph-related tasks. In the context of HLS, where the CDFG is generated from the intermediate representation, GNNs can effectively learn the dependencies between control flow and data flow, capturing the interactions between operations. Additionally, GNNs exhibit strong generalization capabilities, maintaining high accuracy in performance prediction across various design choices and optimization strategies. This is crucial for Design Space Exploration (DSE), as GNNs enable designers to quickly evaluate the performance of a large number of design points, accelerating the iterative optimization process and providing robust support for hardware microarchitecture optimization.

In summary, the application of GNNs can address the challenges of optimization directives in complex hardware designs, enhancing the accuracy and efficiency of design quality prediction and offering advanced solutions for hardware design optimization.

III. METHODOLOGY

Fig. 1 illustrates the overall framework of our proposed post-implementation performance prediction method, which is divided into three stages: dataset generation, model training, and inference.

In the dataset generation stage, we randomly sample candidate optimization directives to determine the specific optimization strategy applied to the designs. Based on the IR generated by the HLS front end, the Graph Builder constructs a directive-optimized graph that incorporates resource reuse and directive

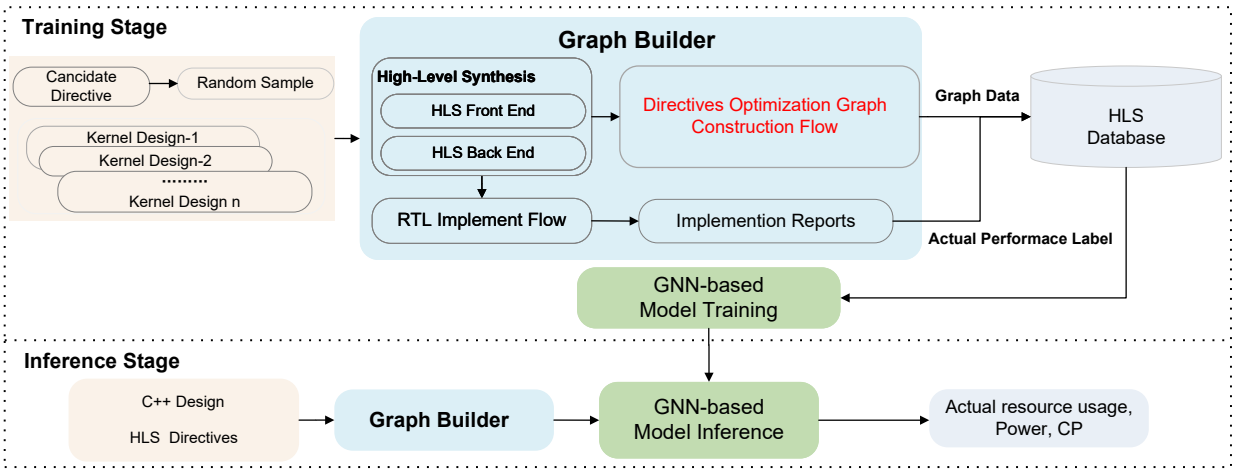


Fig. 1: Framework overview of the proposed method.

optimization effects, providing a more accurate simulation of resource allocation and operation scheduling in hardware circuits. This setup facilitates the use of GNN's powerful graph information learning capabilities for performance prediction. Subsequently, the C-to-bitstream design flow is completed to extract post-implementation performance metrics (LUT, FF, CP, Power, DSP). Using this information, we generate a directive-optimized graph representation corresponding to the optimization directives, along with true performance labels, to build a HLS performance prediction database.

During the training stage, we use the generated HLS database to train a model that integrates GNN and aggregation pool modules. Finally, in the inference stage, for a given test design, the directive-optimized graph representation generated by the Graph Builder is input into the trained model, enabling the prediction of post-implementation performance results without complete the post-implementation process.

A. *Directives Optimization Graph Construction Flow*

We use the Graph Builder to construct a graph representation of the design, which simulates the actual behavior of the hardware circuit. This graph representation incorporates control data flow information, the effects of various optimization directives, and hardware resource reuse details. Fig 2 illustrates our graph construction flow.

1) **Explicit Representation of Directive Optimization**
The CDFG generated from LLVM IR (as shown in Fig.2(a)) provides a clear representation of the program's control structures and data dependencies, aiding designers in analyzing program behavior. However, optimization directives provided by HLS tools can alter the microarchitecture of the hardware circuit design. Specifically, applying the unroll directive duplicates the execution logic within loops to enhance parallelism, while the array partition directive increases memory bandwidth to improve throughput, thereby impacting the overall performance. Consequently, the interactions between directives can significantly change the data flow execution patterns of the program, making accurate performance prediction essential.
To intuitively reflect the impact of directives on hardware design, we constructed a graph representation based on the frontend IR generated by HLS tools and the design principles of optimization directives. This graph representation effectively illustrates the effects of optimization directives on the source program, providing a clear graph structure for GNN-based performance prediction and facilitating the learning of directive interaction features (as shown in Fig.2(b)).
2) **Impact of hardware resource reuse**
The optimized data flow graph effectively captures the execution characteristics of the program but often overlooks the impact of resource reuse caused by directive interactions. To more accurately model the resource allocation and operation scheduling in hardware circuits, we extend the optimization graph representation constructed in step (1) by incorporating resource dependency information derived from the backend of HLS tools. This enhancement improves the reliability of post-implementation performance prediction (as shown in Fig.2(c)).
More specifically, in Fig.2(c), the application of the unroll and array partition directives duplicates the execution logic and partitions array B into two segments, thereby improving parallelism and memory bandwidth. However, due to hardware resource reuse constraints, the two multiplication operations (mul1 and mul2) must share a single DSP resource. Consequently, mul2 must wait for mul1 to release the multiplier resource (e.g., DSP) before execution. To address this, a resource reuse dependency is introduced between these two multiplication operations, ensuring that mul1 and mul2 share the same DSP resource and that mul2 is scheduled to execute after mul1. This optimized graph representation more accurately reflects the resource allocation and operation scheduling in hardware circuits, enabling a more realistic simulation of hardware execution behavior.

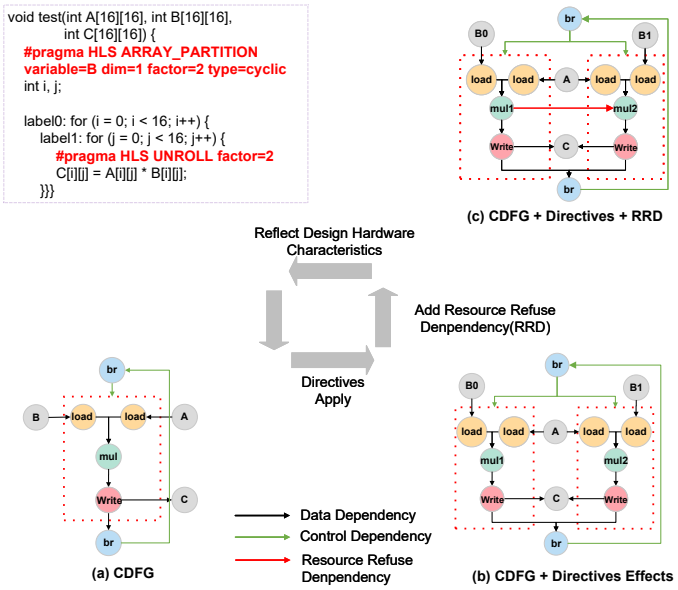


Fig. 2: Directive-optimization Graph Representation.

B. *Graph Feature Extraction and Annotation*

The extraction of node and edge features from the directives optimization graph constructed by Graph Builder is crucial for model training (see Table II).

Firstly, node features are derived from the front-end of the HLS tool, extracting key information related to performance prediction, including node type, opcode type, latency, and resource usage (such as LUT, DSP, FF, etc.). Additionally, each node includes a "Reuse" flag, indicating whether the node reuses hardware resources. Node type and opcode type are encoded using one-hot vectors, while other features (such as latency and resource usage) are represented using numerical values. These features assist the GNN model in understanding the function and resource requirements of each node.

Edge features primarily describe the dependencies between nodes, including data dependencies, control dependencies, and resource reuse dependencies. In particular, resource reuse dependencies describe the operation scheduling and resource sharing in circuit design, aligning more closely with the characteristics of actual hardware execution. Comprehensive feature extraction and precise encoding enable the GNN model to gain deeper insights into the details of hardware design, helping it more effectively analyze and predict the performance of designs optimized by directives.

TABLE II: Node and edge features.

Category	Description	Values
Node Features	Type	Instruction, Variable, etc.
	Opcode Type	Load, Mul, etc.
	Delay	Float values
	LUT	Integer values
	DSP	Integer values
	FF	Integer values
	Reuse	0, 1
Edge Feature	Type	Control, Data, Call, Resource Reuse

C. *GNN-Based Prediction Approach*

Mainstream GNN models (such as GAT [12] and GraphSAGE [13]) typically focus on aggregating node features while often overlooking edge feature learning. However, in the directive-optimized graph constructed by Graph Builder, edge relationships contain rich circuit characteristics, including hardware resource reuse and the interactions between directive. To address this limitation, our performance prediction model (as shown in Fig.3) employs TransformerConv [14] as a foundational module for node information aggregation. TransformerConv introduces a self-attention mechanism similar to that in Transformers, enabling dynamic fusion of edge features between nodes, which allows for more effective handling of the graph data generated by Graph Builder. While learning node features, TransformerConv can also fully capture data dependencies, directive interactions, and resource reuse, thus extracting critical features related to performance metrics.

The learning of node features is illustrated by Equation (1),

$$\mathbf{x}_i' = \mathbf{W}_1\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}(\mathbf{W}_2\mathbf{x}_j + \mathbf{W}_5\mathbf{e}_{ij}) \tag{1}$$

where the attention coefficient $\alpha_{i,j}$ is calculated as follows:

$$\alpha_{i,j} = \text{softmax}\left(\frac{(\mathbf{W}_3\mathbf{x}_i)^\top(\mathbf{W}_4\mathbf{x}_j + \mathbf{W}_5\mathbf{e}_{ij})}{\sqrt{d}}\right) \tag{2}$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, W_5, \mathbf{W}_5$ are learnable weight parameters, and $\mathbf{x}_i$ and $\mathbf{e}_{ij}$ represent the feature embeddings of node i and the edge between nodes i and j, respectively. To further enhance the model's ability to learn graph-level features, we introduce an aggregation pool module [15] after three layers of TransformerConv to obtain a graph-level representation. Additionally, global information is incorporated into the graph-level representation, further enriching the information expression for circuit design and significantly improving the model's prediction accuracy and generalization capability. Finally, the graph-level and global features are concatenated and fed into a multi-layer perceptron (MLP) layer to predict performance metrics.

In summary, by integrating the TransformerConv model and aggregation pool, incorporating global information, we capture critical features related to performance prediction, thereby enhancing the performance prediction capability for circuit designs. Our prediction model framework is shown in Fig.3.

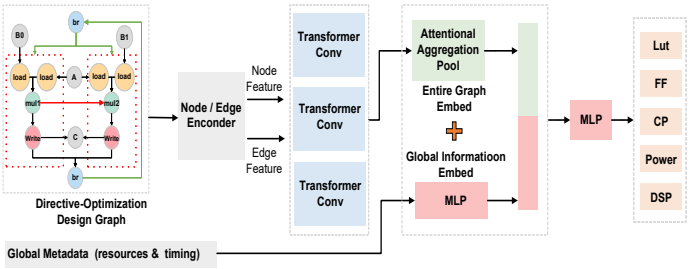


Fig. 3: The architecture of predictive model.

IV. EXPERIMENT

A. *Experimental Setup*

Dataset. We selected 19 representative kernel designs from the commonly used MachSuite [16] and PolyBench [17] benchmark suites. These kernels cover various types of computational complexity, including typical algorithms such as matrix multiplication, triangular solving, dynamic programming, and Jacobi iteration.

For each kernel design, we randomly generated 1,000 different directive configuration combinations, synthesized using Vitis HLS 2023.2 and implemented with Vivado to obtain actual hardware performance metrics, including resource usage (e.g., LUT, DSP, FF), critical path delay (CP), and power consumption. Finally, we combined each HLS kernel's generated directive-optimization graph representation with corresponding performance labels, constructing an HLS dataset to comprehensively evaluate the effectiveness of our method.

More specifically, these 19 kernels generate a total of 19,000 design points. Among them, design points from 15 kernels are used for training and testing, while the remaining 4 kernels are reserved to evaluate the model's generalization capability on unseen kernel designs. The specific details of the constructed HLS dataset are shown in Table III.

TABLE III: Statistics of databases which consist of 19 kernels.

Version	Kernels	Range	LUT	FF	CP	Power	DSP
Vitis HLS 2023.2	19	min	19	26	1.48	0.24	0
		max	138942	246368	10	2.97	1428

Baseline. To validate the effectiveness of our method, we compared it with four baseline methods: PNA [18], IronMan-Pro [6], GNN-DSE [7], and HGP [19] represent kernel characteristics through data flow graphs, while GNN-DSE directly inserts directive nodes into the graph representation. In our experiments, we primarily focus on prediction accuracy and generalization capability, using Mean Absolute Percentage Error (MAPE) to evaluate LUT, FF, CP, and Power metrics, and Mean Absolute Error (MAE) for the DSP metric. Additionally, we implemented the Graph Builder in Python to construct the directive-optimized graph representation and perform feature extraction and encoding. Both our method and the baseline GNN methods were implemented using Python and PyTorch Geometric.

B. *Model Evaluation*

In the above experimental setup, we selected 15,000 design points (from 15 kernel designs), with 80% used for training and 20% for testing, to evaluate the performance of all models.

The performance of all models is shown in Fig. 4, the proposed model outperforms baseline models (PNA, IronMan-Pro, GNN-DSE, and HGP) across five evaluation metrics, achieving the lowest MAPE (MAE) values. Compared to PNA and IronMan-Pro, the proposed approach demonstrates significant improvements in all performance prediction metrics, primarily due to its effective capture of node features and edge dependencies in the directive optimization graph. Although GNN-DSE and HGP reduce performance prediction errors to some extent, the proposed method still surpasses them, further demonstrating its ability to effectively learn hardware resource reuse, directive interaction information, and correlations between performance metrics.

Overall, our model reduces the prediction errors for LUT, FF, CP, and Power to 8.08%, 3,87%, 4.34%, and 6.49%, respectively. And achieves an MAE of 0.11 for DSP, which is significantly lower than that of other models, this low error indicates a high consistency between predicted and actual values, further validating the model's high accuracy and stability.

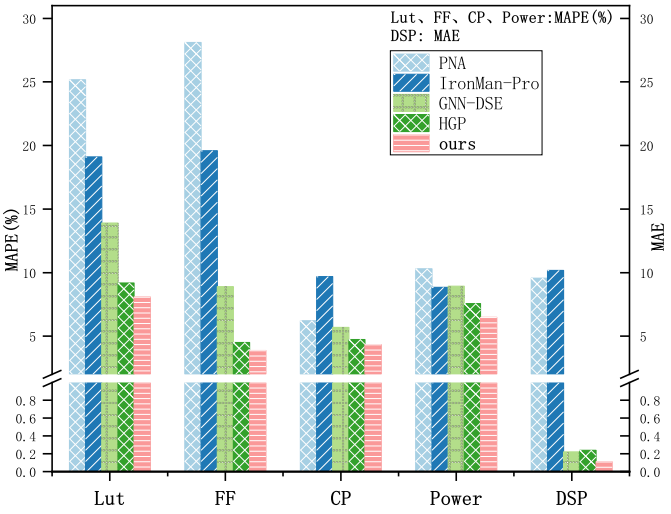


Fig. 4: Performance comparison.

C. *Effectiveness of Resource Reuse*

In actual RTL circuits, resource reuse is closely related to operation scheduling and resource allocation, making it essential to consider resource reuse information for reducing model prediction errors. To assess the impact of resource reuse information, we compared the model's performance with (W/) and without (W/O) this information. As shown in TableIV, incorporating resource reuse information significantly reduces prediction errors across five metrics. Specifically, the MAPE for LUT, FF, CP, and Power decreased from 9.41%, 4.52%, 4.54%, and 7.25% to 8.08%, 3.87%, 4.34%, and 6.49%, respectively, while the MAE for DSP dropped from 0.20 to 0.11. These results indicate that incorporating resource reuse information enables the model to more accurately capture the operation scheduling and resource allocation characteristics of hardware circuits, effectively reducing prediction errors.

TABLE IV: Performance comparison of resource reuse.

Method	Configuration	MAPE (%)				MAE
		Lut	FF	CP	Power	DSP
Ours	W/O Resource Reuse	9.41	4.52	4.54	7.25	0.20
	W/ Resource Reuse	8.08	3.87	4.34	6.49	0.11

TABLE V: Generalization performance comparison of different models on unseen Kernels. GNN-DSE is abbreviated as GD in the table.

	MAPE (%)																MAE			
	Lut				FF				CP				Power				DSP			
Kernels	PNA	GD	HGP	**ours**	PNA	GD	HGP	**ours**	PNA	GD	HGP	**ours**	PNA	GD	HGP	**ours**	PNA	GD	HGP	**ours**
viterbi	71.95	46.67	18.76	**18.33**	57.5	20.38	9.06	**6.91**	10.74	16.38	9.01	**6.51**	28.96	16.94	37.03	**8.97**	20.51	4.39	3.67	**3.21**
trmm	49.59	23.23	22.94	**17.96**	39.69	10.21	12.26	**8.87**	12.62	7.11	**6.92**	8.89	13.71	10.23	15.51	**10.21**	12.89	0.31	3.17	**0.09**
syrk	53.80	23.71	30.32	**20.96**	64.67	14.4	13.18	**9.44**	9.19	7.98	**6.51**	7.60	31.83	17.91	22.53	**14.31**	32.74	0.21	11.31	**0.08**
2mm	27.61	17.09	14.72	**12.53**	39.85	7.37	4.72	**3.69**	6.02	6.84	6.92	**5.84**	7.88	4.51	7.24	**4.11**	6.36	0.09	1.65	**0.02**
Mean	50.74	27.68	21.69	**17.45**	45.68	13.09	9.81	**7.23**	9.64	9.57	7.34	**7.21**	20.60	12.40	20.58	**9.40**	18.13	1.25	4.95	**0.85**

D. Generalization on Unseen Kernel

To evaluate the model's generalization capability, we conducted experiments on four unseen kernel designs (viterbi, trmm, syrk, and 2mm). As shown in TableV, the proposed model outperforms other baseline models (PNA, GNN-DSE, and HGP) on unseen kernel designs, achieving lower MAPE (MAE) across five metrics. Specifically, the average MAPE of our model for LUT, FF, CP, and Power decreased to 17.45%, 7.23%, 7.21%, and 9.40%, respectively, while the MAE for DSP was significantly reduced to 0.85, much lower than other models. The experimental results validate the strong generalization capability of the proposed method on unseen kernel designs, further demonstrating the effectiveness of combining the directive optimization graph representation with the performance prediction model architecture. This approach enables a more comprehensive modeling of the commonalities and differences across various designs, highlighting its robustness and adaptability.

V. Conclusion

In this paper, we propose a novel post-implementation performance prediction method for directive-optimized designs. This approach involves the development of a Graph Builder, which constructs a directive optimization graph that integrates both directive optimization effects and resource reuse information. By combining an ensemble GNN model with an aggregation module, we extract key features relevant to performance prediction, such as resource allocation and operation scheduling, to achieve precise performance estimation. Experimental results demonstrate that our method significantly outperforms the baseline approaches across five metrics and exhibits strong generalization ability on unseen kernels.

Acknowledgement

This work was supported in part by National Natural Science Foundation of China (Nos. 62472155, 62202154), the science and technology innovation Program of Hunan Province (No.2024RC3076), and the Fundamental Research Funds for the Central Universities.

References

[1] Biagio Peccerillo, Mirco Mannino, Andrea Mondelli, and Sandro Bartolini. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture*, 129:102561, 2022.

[2] *Vitis High-Level Synthesis User Guide (UG1399)*, 2022. Retrieved from Xilinx Documentation: https://docs.xilinx.com/r/2022.1-English/ug1399-vitis-hls.

[3] Guanwen Zhong, Alok Prakash, Yun Liang, Tulika Mitra, and Smail Niar. Lin-analyzer: A high-level performance analysis tool for fpga-based accelerators. In *DAC*, pages 1–6, 2016.

[4] Jieru Zhao, Liang Feng, Sharad Sinha, Wei Zhang, Yun Liang, and Bingsheng He. Comba: A comprehensive model-based analysis framework for high level synthesis of real applications. In *ICCAD*, pages 430–437. IEEE, 2017.

[5] Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline FY Young, and Zhiru Zhang. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *FCCM*, pages 129–132. IEEE, 2018.

[6] Nan Wu, Yuan Xie, and Cong Hao. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of teh 2021 on Great Lakes Symposium on VLSI*, pages 39–44, 2021.

[7] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. Automated accelerator optimization aided by graph neural networks. in 2022 59th acm. In *DAC*, 2022.

[8] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. Robust gnn-based representation learning for hls. In *ICCAD*, pages 1–9. IEEE, 2023.

[9] Mingzhe Gao, Jieru Zhao, Zhe Lin, and Minyi Guo. Hierarchical source-to-post-route qor prediction in high-level synthesis with gnns. In *DATE*, pages 1–6. IEEE, 2024.

[10] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International symposium on code generation and optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004.

[11] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[12] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[14] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[15] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *ICML*, pages 3835–3845. PMLR, 2019.

[16] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks. Machsuite: Benchmarks for accelerator design and customized architectures. In *IISWC*, pages 110–119. IEEE, 2014.

[17] T. Yuki and L.-N. Pouchet. PolyBench/C. [Online]. Available: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.

[18] Nan Wu, Hang Yang, Yuan Xie, Pan Li, and Cong Hao. High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing. In *DAC*, pages 49–54, 2022.

[19] Huizhen Kuang, Xianfeng Cao, Jingyuan Li, and Lingli Wang. Hgbo-dse: Hierarchical gnn and bayesian optimization based hls design space exploration. In *ICFPT*, pages 106–114. IEEE, 2023.

Libra: A Hybrid-Sparse Attention Accelerator Featuring Multi-Level Workload Balance

Faxian Sun*, Runzhou Zhang, Zhenyu Liu, Heng Liao, Zhinan Qin, Jianli Chen, Jun Yu and Kun Wang†
State Key Lab of Integrated Chips & Systems, and School of Microelectronics, Fudan University, Shanghai, China
*23212020018@m.fudan.edu.cn, †kun.wang@ieee.org

***Abstract*—Transformers have delivered exceptional performance and are widely used across various natural language processing (NLP) tasks, owing to their powerful attention mechanism. However, the high computational complexity and substantial memory usage pose significant challenges to inference efficiency. Numerous quantization and value-level sparsification methods have been proposed to overcome these challenges. Since higher sparsity leads to greater acceleration efficiency, leveraging both value-level and bit-level sparsity (hybrid sparsity) can effectively exploit the acceleration potential of the attention mechanism. However, increased sparsity exacerbates load imbalance across compute units, potentially limiting the extent of acceleration benefits. To fully exploit the acceleration potential of hybrid sparsity, we propose Libra, an attention accelerator developed through algorithm-hardware co-design. At the algorithm level, we design the bit-group-based algorithm consisting of filtered bit-group sparsification (FBS) and dynamic bit-group quantization (DBQ) to maximize the utilization of sparsity in attention. FBS imposes structured sparsity on weights, while DBQ introduces dynamic sparsification during the computation of activations. At the hardware level, we design task pool to achieve multi-level workload balance, effectively mitigating the load imbalance among compute units induced by hybrid sparsity. Additionally, different stages in DBQ can be executed in parallel, with each stage operating at distinct bit-widths. To support this, we design an adaptive bit-width architecture that enables simultaneous computations at varying bit-widths. Our experiments demonstrate that, compared to state-of-the-art (SOTA) attention accelerators, Libra achieves up to $1.49\times \sim 5.89\times$ speedup and $2.65\times \sim 10.82\times$ enhancement in energy efficiency.**

I. Introduction

In recent years, Transformers [3] have exhibited remarkable performance in Natural Language Processing (NLP) [4, 10, 6, 7, 8, 5, 15], establishing themselves as the prevailing architecture for various tasks. This success is largely attributable to their innovative attention mechanism. The emergence of large pre-trained language models, such as GPT [8, 10], has further entrenched the dominance of Transformer architectures in the field. However, as these models grow in size and complexity, Transformers face increasingly significant computational challenges. The attention mechanism, in particular, imposes substantial computational costs and results in considerable memory footprints, thereby restricting the deployment of Transformer models in resource-constrained devices and real-time applications.

Attention mechanism analyzes the relationships between all tokens in input sequence to evaluate their relative significance through attention scores. However, as the input sequence length grows, the computational complexity and memory cost of attention scores increase quadratically. As a result, attention computation becomes the primary bottleneck in deploying Transformers. To address the high computational redundancy inherent in attention calculation, sparse attention is an appealing approach. Sparse attention reduces the computational cost of dense attention by skipping attention calculation for tokens with low scores, thereby eliminating redundancy in the process. Sanger [14] mitigates some of this redundancy through threshold filtering, setting low-relevance scores to zero. However, it focuses solely on activation computation, neglecting the significant redundancy present in weights. By leveraging sparsity in both weights and activations, the overall redundancy in attention computation can be more effectively reduced.

†Corresponding author

Existing Transformer accelerators [14, 17, 20] leverage value-level sparsity to skip computations involving zero-value correlations, effectively reducing the computational load and achieving acceleration. However, as shown in Fig. 1, even among non-zero values, numerous zero bits remain. This naturally suggests leveraging bit-level sparsity to further reduce inference latency. Computing an N bits data element by only one multiplier unit requires N cycles without accounting for bit-level sparsity. However, with bit-serial architecture (BSA), the number of required cycles matches the number of significant bits, which is always less than or equal to N. However, previous attempts to accelerate using BSA were tailored for Convolutional Neural Networks (CNNs) and are not hardware-friendly for Transformers, making direct adaptation infeasible. This poses significant challenges to our efforts to implement both value-level and bit-level sparsity (hybrid-sparsity) in attention.

In bit-serial computation, workload distribution across compute units is often imbalanced due to the uneven distribution of bit sparsity. As stated in Fig. 1, compute units with lighter workloads may experience extended stalling, waiting for those with heavier workloads to complete. This issue is particularly evident when employing bit-level sparsity in activations. The root of the problem lies in fact that data channels remain unchanged after initial scheduling until all computations are complete, leading to stalled computational resources without further rescheduling. By employing structured sparsity to control distribution of effective bits, bit-level sparsity in weights can mitigate workload imbalance.

To tackle the insufficient utilization of sparsity in attention computation and the workload imbalance caused by hybrid sparsity, we propose Libra, a hybrid-sparse attention accelerator that achieves workload balance through bit-group, offering a solution to make Transformer acceleration more efficient. Our main contributions can be summarized as follows:

- We propose the bit-group-based algorithm consisting of filtered bit-group sparsification (FBS) and dynamic bit-group quantization (DBQ), applying structured sparsity to weights and dynamic sparsity to activations, thereby maximally eliminating redundancy in attention computation.
- An adaptive bit-width hardware architecture is developed to leverage the structural characteristics of the bit-group-based algorithm, allowing different bit-width computation stages to execute in parallel and improving hardware utilization.
- A multi-level workload balance scheme centered around the task pool for workload distribution is introduced. By decomposing workloads into fine-grained tasks and dynamically scheduling them, the scheme achieves near-optimal performance when exploiting hybrid-sparsity.
- Extensive experiments are conducted to evaluate the performance of our accelerator. Libra achieves up to $1.49\times \sim 5.89\times$ speedup and $2.65\times \sim 10.82\times$ enhancement energy efficiency compared to SOTA attention accelerators with negligible accuracy loss.

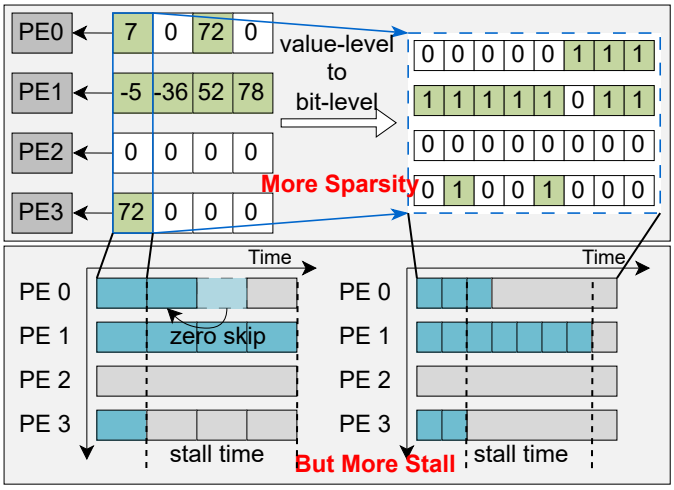


Fig. 1: Increased workload imbalance issue when leveraging bit-level sparsity.

II. BACKGROUND AND MOTIVATION

A. Attention Mechanism

The attention mechanism is the core operation within the Transformer models. During the computation of attention, the embedded input sequence is initially processed through linear layer to generate the Query (Q), Key (K), and Value (V) matrices. After that, the score matrix is derived by computing the dot product between Q and the K^T. Subsequently, the generated score matrix is normalized using non-linear functions to produce the probability matrix P which is then multiplied with the V. The result matrix is applied to linear functions and derived through residual connection. After the final layer normalization, we can get the attention result.

B. Sparse Accelerator

Various sparsity and quantization methods have been employed to accelerate attention computations. Sanger [14] leverages redundancy in the attention mechanism to enhance value-level sparsity. DTQAtten [17] and DTATrans [16] implement token-wise mixed-precision quantization. DEQ [20] further advances this by introducing element-wise mixed-precision quantization. These methods have achieved remarkable performance. However, they only optimize at the value level, neglecting the huge potential for acceleration at the bit level.

Existing researches on bit sparsity focus on accelerating CNNs using BSA. Laconic [9] processes both weights and activations as bit sequences, while Adas [18] dynamically selects the side with fewer effective Booth terms for sequence expansion. However, the irregular distribution of bit sparsity in weights and activation can result in load imbalance during computation, where some processing elements (PEs) are active while others remain idle. While Adas uses inner-join units to alleviate this issue, it does not fully resolve the problem for the entire PE array. To address the issue of load imbalance, Bit-Balance [19] proposes a channel-wise bit-sparse quantization method and uses clustering techniques to balance workloads across PEs.

Despite these advances, the aforementioned bit-sparsity methods primarily rely on patterned distributions of significant bits in weights, which are less applicable to the dynamically generated activations in Transformers. BSViT [21] introduces a dense-sparse hybrid dataflow method to achieve workload balance in PEs but primarily leverages dynamic patch quantization in Vision Transformers (ViT) [11] and lacks a comprehensive focus on the unique characteristics and requirements of the attention mechanism in Transformers.

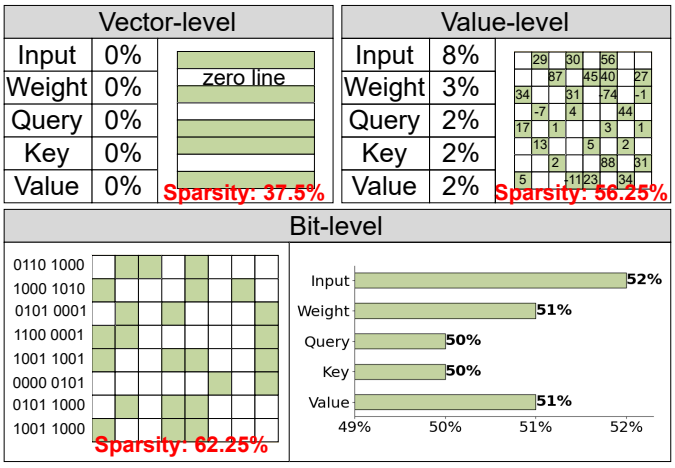


Fig. 2: Preliminary experimental results on the vector-level, value-level and bit-level sparsity in the BERT model and GLUE dataset.

C. Motivation

The primary motivation for our work is based on two key observations. On the one hand, as shown in the Fig. 2, Transformers exhibit extremely high bit-level sparsity, approximately 25× greater than value-level sparsity without applying quantization or sparsity techniques. In particular, the bit-level sparsity in P can reach as high as 96%. Therefore, leveraging hybrid sparsity will further enhance the acceleration effects by adding bit-level sparsity on top of value sparsity. Additionally, due to the redundant nature of the attention mechanism, combining hybrid sparsity with attention features holds significant potential for further acceleration.

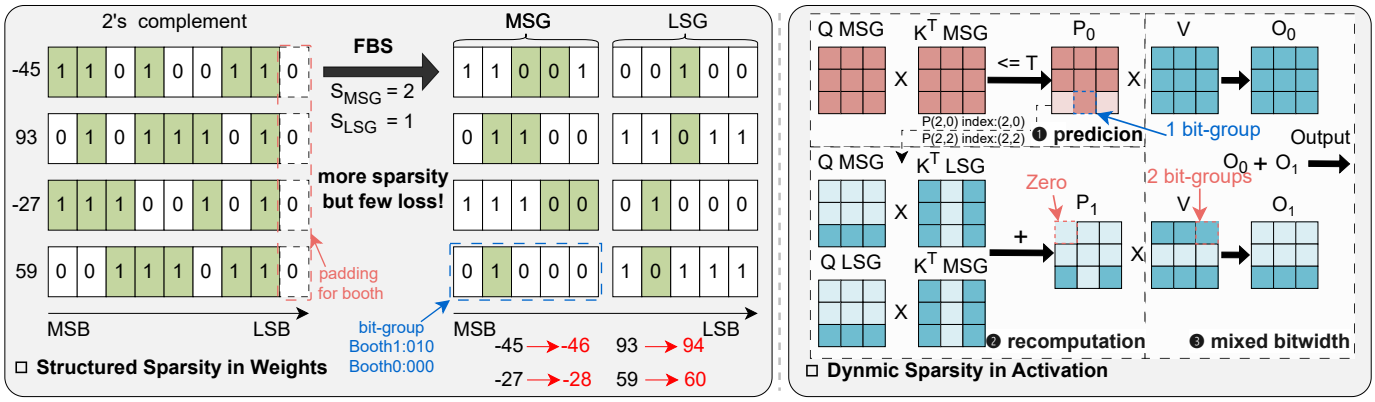
On the other hand, as illustrated in Fig. 1, the irregular distribution patterns of zero values and zero bits increase the likelihood of compute units idling, which further exacerbates workload imbalance. The acceleration benefits brought by increased sparsity may not always offset the losses caused by load imbalance. Therefore, maintaining balanced workloads while leveraging hybrid sparsity remains an unavoidable challenge. For instance, Bit-Balance [19] uses a clustering method to group workloads with similar characteristics into the same channels, facilitating inter-channel load balance through scheduling. Adas [18] achieves load balance within PEs through inner-join units. However, when leveraging hybrid sparsity, single-level load balance methods are insufficient to address these challenges. This motivates the design of our multi-level workload balance strategy.

III. BIT-GROUP-BASED ALGORITHM

In this section, we introduce our algorithmic optimizations based on bit-group, extending from weight FBS to activation DBQ. BSA performs a significant amount of unnecessary computations due to the presence of negative numbers [9]. To eliminate these redundancies, we incorporate Booth encoding into our bit-level operations. Furthermore, to fully leverage the advantages of Booth encoding, our algorithm is structured around bit-group, each containing 5 bits that can be encoded into two Booth codes.

A. Filtered Bit-group Sparsification

We propose the FBS scheme to reduce redundancy in weight representations and decrease hardware computational overhead. For integers encoded with Booth encoding, only bit sequences of '000' and '111' are encoded as '0'. Therefore, increasing the continuity of bit positions in the weights can enhance the sparsity of weights after Booth encoding.



(a) Filtered Bit-group Sparsification (b) Dynamic Bit-group Quantization

Fig. 3: An overview of bit-group-based algorithm. (a) Libra takes filtered bit-group sparsification (FBS) to balance weight distribution with negligibe accuracy loss. (b) Libra employs dynamic bit-group quantization (DBQ) to introduce dynamic sparsity in activations.

Algorithm 1: Filtered Bit-Group Sparsification

Input: Weight tensor $\mathbf{W}$, error threshold ϵ
Output: Compensated sparsified weight tensor $\mathbf{W}_{\text{comp}}$
$\mathbf{W}_q \leftarrow \text{Quantize}(\mathbf{W},\ 8\ \text{bits})$ ▷ Step 1
  Extract high bits (**MSG**) and low bits (**LSG**):
  MSG $\leftarrow$ high 5 bits of $\mathbf{W}_q$;
  LSG $\leftarrow$ low 4 bits of $\mathbf{W}_q$;
  LSG $\leftarrow$ left-shift **LSG** by 1 bit;
Sparsify **MSG** and **LSG** under constraints: ▷ Step 2
  $\text{MSG}_{\text{sparse}} \leftarrow \text{Sparsify}(\textbf{MSG},\ S = 2)$;
  $\text{LSG}_{\text{sparse}} \leftarrow \text{Sparsify}(\textbf{LSG},\ S = 1)$;
Reconstruct sparsified weights: ▷ Step 3
  $\mathbf{W}_{\text{sparse}} \leftarrow \text{Combine}(\text{MSG}_{\text{sparse}},\ \text{LSG}_{\text{sparse}})$;
Compute error: $\Delta\mathbf{W} \leftarrow \mathbf{W}_q - \mathbf{W}_{\text{sparse}}$ ▷ Step 4
while $\|\Delta\mathbf{W}\|_2 > \epsilon$ **do**
  Adjust $\text{MSG}_{\text{sparse}}$, $\text{LSG}_{\text{sparse}}$ to minimize $\|\Delta\mathbf{W}\|_2$ under constraints;
  Reconstruct $\mathbf{W}_{\text{sparse}}$;
  $\Delta\mathbf{W} \leftarrow \mathbf{W}_q - \mathbf{W}_{\text{sparse}}$;
$\mathbf{W}_{\text{comp}} \leftarrow \mathbf{W}_{\text{sparse}} + \text{CompensateError}(\Delta\mathbf{W})$;

To better preserve the numerical value of the weights, each quantized weight can be represented as $B = -B_{n-1}2^{n-1} + \sum_{i=0}^{n-2} B_i * 2^i$. Numerical values are composed of the sum of multiple powers of two. Therefore, the weights can be denoted as follows,

$$B = \begin{cases} \sum_{i=m}^{M} B_i * 2^i, & B \geq 0 \\ -\sum_{i=m}^{M} B_i * 2^i - 1, & B < 0 \end{cases}, \quad (1)$$

where $M = \min(N-1, m+s-1)$, s represents the length of the continuous range of effective bits, and $m \in \{0, 1, 2, \ldots, N-1\}$. As shown in Fig. 3 and Algorithm 1, the quantized weights are first divided into two groups. The first group, called Most Significant Group (MSG), includes bits from the 7th to the 3rd position. The second group, called Least Significant Group (LSG), includes bits from the 3rd to the 0th position, with an additional zero appended at the end according to Booth encoding rules.

Next, we assign a fixed maximum number of significant bits, S, to each bit group. For example, $S = 2$ for MSG and $S = 1$ for LSG. For positive integers, the significant bits are the non-zero bits, but for negative integers, the significant bits are the zero bits. Finally, we optimize the weights of the current layer according to the constraints defined by our scheme. If the difference between the sparsified weights and the original weights exceeds a given threshold ϵ, we introduce appropriate compensations and re-sparsify them iteratively until the specified constraints are satisfied.

B. Dynamic Bit-group Quantization

In Transformers, the correlation between each pair of tokens is initially calculated as $P = Q \cdot K^T$. Due to the softmax function, less significant correlations P approach zero, making them irrelevant in the subsequent $P \cdot V$ computations. Therefore, we can predict the value of P using lower-cost computations, assign lower bit-width precision to less important values, and perform high-precision recalculations for important values. This approach minimizes computational complexity while maintaining accuracy.

In our design, we introduce a Dynamic Bit-group Quantization (DBQ) scheme, which can dynamically allocate data bit-widths without requiring a redesign of the network architecture. Since i-BERT [13] can maintain accuracy comparable to full precision using Int8, our design also adopts Int8 as the base precision. As shown in Fig. 3, the DBS scheme consists of two main stages.

The first stage is the prediction stage. To be compatible with the DBQ scheme, the MSG of Q and K are multiplied to generate a prediction result P, which is represented by a single bit-group. Each element of P is then compared to a predefined threshold T. The threshold T is determined by a ratio parameter r and the specific task. If $P_{i,j} \leq T$, the corresponding correlation is considered unimportant and can be represented with a single bit-group. Otherwise, a high-precision recalculation is required to avoid accuracy loss. To facilitate locating important elements, we retain the row and column indices for future use.

The second stage is the recomputation stage. Using the previously stored indices, we can locate the important elements $P_{i,j}$, retrieve the ith token of Q and the jth token of K, read their MSG and LSG, and then perform vector-vector multiplication to obtain $P_{i,j}$. This recalculation process is repeated for all important elements. Now, we have a matrix P with two different precision, which needs to be multiplied by the value matrix V. Considering hardware implementation specifics, we split this into two groups of computations: one for the full-precision p multiplied by V, producing a partial output matrix O_0. The other group, based on the previously stored indices, retrieves the high-precision $P_{i,j}$ and the jth token of V, multiplying them to produce another part of the output matrix O_1. The final output O is obtained by summing the two matrices.

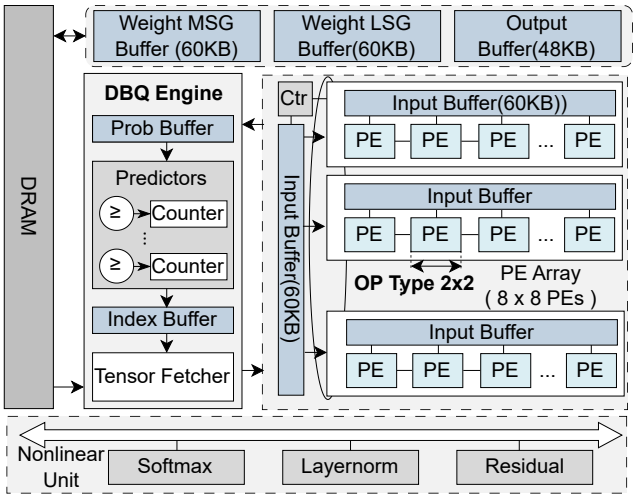


Fig. 4: Overview of Libra hardware architecture.

IV. LIBRA HARDWARE ARCHITECTURE

A. Architecture Overview

To fully leverage the advantages of FBS and DBQ, we designed Libra, a hybrid-sparse attention accelerator closely tailored to the algorithm. The key architectural highlights include: 1) an adaptive bit-width PE based on the bit-group multiplier unit (BMU), which accommodates the varying bit-width requirements at different computation stages. 2) a task pool, which maximizes bit sparsity utilization, effectively balances workloads across multi-levels. In this section, we first present the overall architecture of Libra. Then we discuss how the task pool within the BMU reduces unnecessary computations by exploiting bit sparsity. Finally, we detail the PE constructed from BMU, highlighting its ability to leverage hybrid sparsity and implement multi-level workload balance strategies. The overall architecture of Libra, as shown in Fig. 4, includes input buffers, weight buffers, an output buffer, an 8×8 decoupled PE array composed of adaptive bit-width PEs, a global accumulator, nonlinear units, a DBQ engine, and a top-level controller.

B. Bit-group Multiplier Unit

As shown in Fig. 6, a single PE unit consists of four BMU. Each BMU is composed of an encoder, a task pool, four bit-group compute cores, and decoder. The input to each BMU is bit-groups. As shown in Fig .5, after being encoded by the encoder, tasks are generated and stored in the task pool for computation, with the output handled by the accumulation unit. In the task generation phase, our hardware unit eliminates unnecessary 0-Booth computations, taking advantage of bit sparsity.

1) **Encoder**. We ensure that each BMU receives data represented by a single bit-group, which allows us to implement our encoder with very low overhead. The encoder receives bit-group in each cycle, producing necessary task pairs that are sent to the task pool. The bit-group can be encoded into two Booth-encoded values according to the following formulas: $Booth_1 = 4 * (-2b_4 + b_3 + b_2)$, $Booth_0 = 1 * (-2b_2 + b_1 + b_0)$. The values of b_i range from {2, 1, 0, -1, -2}, so $Booth_i$ has a range of $[-8, 8]$, and can be shown as $[-2^3, 2^3]$. So we can decode these into a sign bit, a valid bit and a 3-bit shift term. When performing multiplication, only the exponent terms need to be added. Finally, the decoder produces the final output.

During the task generation stage, we use valid bit to compress unnecessary computations. For example, as shown in Fig. 5, for A and B, with valid bits of [1, 0] and [1, 1], respectively, if the Booth_0

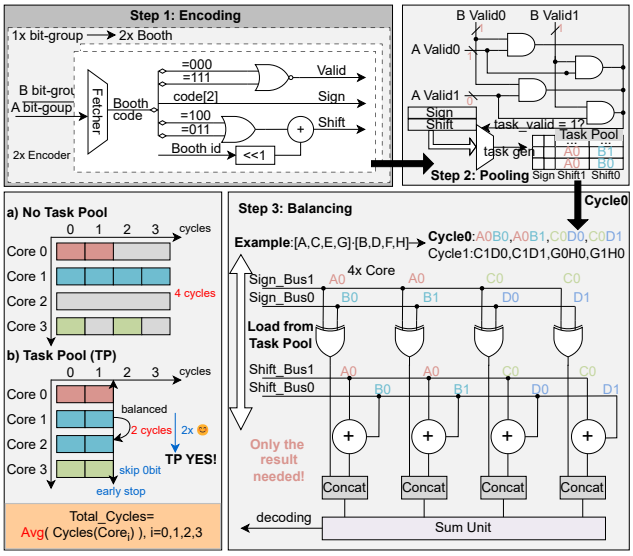


Fig. 5: Illustration of workload balancing in BMU via task pool.

of A is 1, two tasks $[A_0, B_1]$ and $[A_0, B_0]$ will be generated. This reduces the required tasks by 50% compared to the original four tasks.

2) **Task Pool**. The task pool is primarily used to temporarily store tasks, enabling fine-grained scheduling both within and between BMU to fully utilize sparsity. Since each BMU contains four compute cores, the task pool must maintain enough tasks to ensure all cores remaining active, to prevent idle periods. Simultaneously, since up to 4 tasks can be generated in a single decode cycle, we designed the task pool of each BMU to have a depth of 8, striking a balance between functionality and minimizing overhead.

For each BMU, the computation involves multiplying and summing two vectors. As we only need the correct final result rather than element-wise result, we can use the task pool to issue tasks out-of-order, dynamically assigning them to compute cores that have finished their previous tasks. This avoids load imbalance caused by the irregular distribution of bit sparsity within the BMU.

3) **Compute Core**. The input to the compute core is a task. Due to the decoder design, we can use an adder to achieve the same functionality as a multiplier. We perform an XOR on the sign bits to obtain the sign of the partial product, while the exponent terms are added to determine the final shift amount. Since we only need the sum of all partial products, there is no need for immediate decoding. The sum unit accumulates the results from all cores based on the sign bits, and once all computations are complete, the decoder provides the final output.

C. BMU-based Adaptive PE

In the computation process, we encounter three different bit-width operations: 2×2, 2×1, and 1×1, where '1' represents a single bit-group (3 bits) and '2' represents 2 bit-groups (5 bits). To avoid wasting area by designing separate PE units for each bit-width, we designed an adaptive bit-width PE unit based on BMU. By combining BMUs in different configurations, we support three different bit-width operations required by our algorithm.

As shown in the Fig. 6, we can combine 2 BMU into one 2×1 PE. Similarly, 4 BMU can be further combined into a 2×2 PE. Taking 2×2 computation as an example, there is some sparsity at the value level. Thus, before feeding data into the PE, we perform bit-mask preprocessing to eliminate unnecessary computations at the value level. Bit-level sparsity utilization is handled by the task pool,

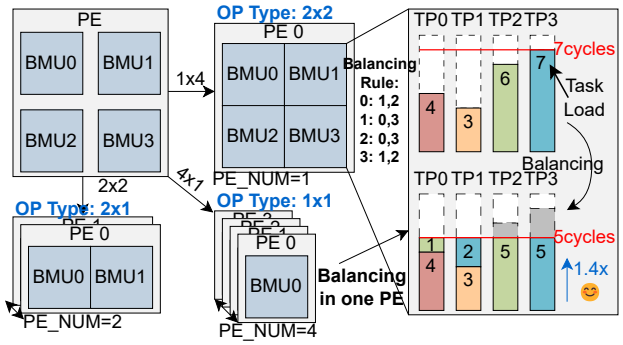


Fig. 6: Cases of adaptive bit-width in PE. Workload balancing within PE for 2x2 OP type.

as described in the previous section. Due to the irregularity of the hybrid sparsity distribution, we found that maintaining workload balance among BMUs within a PE and across different PEs is difficult through pre-allocation alone. Therefore, we implement a multi-level workload balance strategy using the task pool from the BMUs. Each PE contains 4 task pools, which are arranged in a loop to enable data sharing between 4 BMUs. To further reduce the overhead caused by task scheduling, tasks are prioritized within the same PE. As shown in Fig. 6, when BMU0 completes its computation, it loads task from the task pool of the adjacent BMU at the tail end, performs the calculation, and temporarily stores the accumulated result in a dynamic register. Once all computations are completed, the result is returned to the corresponding BMU for summation, allowing for any subsequent operations such as shifting.

Similarly, load imbalance can occur across PEs. We set up a stack in each PE's input channel to temporarily store input data loaded onto the chip. A monitor is added to stack to track the number of value pairs expected to be computed by each PE in real time. Within the same PE group, stacks are shared. When one PE finishes its computation, it will prioritize reading from the stack of the PE with the most remaining tasks, thereby balancing the workload within the same PE group. Considering that in the $P \cdot V$ computation stage, different PE groups handle different types of computations, and cross-PE group scheduling would incur significant overhead, outweighing the potential benefits. Therefore, we do not consider scheduling across PE groups.

V. EVALUATION

A. Experimental Setup

Benchmarks. To validate the computing performance of Libra, we select the Bert-base model and fine-tuned it on eight GLUE text classification tasks. Our evaluation criteria encompass F1 scores for MRPC and QQP, Pearson correlation for STS-B, and Matthews correlation for CoLA. Accuracy on the validation set is reported for remaining tasks. Subsequently, we evaluate the model's inference capabilities, building them with the open-source Huggingface [12] and PyTorch [2] libraries. For all models, we perform five rounds of fine-tuning using the AdamW optimizer with a learning rate of 2e-6 and cosine scheduling.

Hardware implementation. We implement the Libra architecture in Verilog HDL and synthesize it by Synopsys Design Compiler at 500MHz under TSMC 28nm HPC+ technology. The power consumption of SRAM-based on-chip systems is obtained by CACTI 7 [1].

B. Evaluation on Libra Algorithm

1) Weight FBS. To evaluate the effectiveness of the proposed FBS for weights, we test the Booth sparsity of weights after applying FBS and compare the results with a baseline without FBS. Fig. 7 shows the Booth sparsity across different datasets. We apply '2' and '1' constraints to MSG and LSG, respectively. Compared to the baseline, Booth sparsity in weights after FBS increase by an average of 15.92%. Given that, without FBS, the computational reduction in Bit-Serial Architecture depends on bit sparsity, we compare the Booth sparsity of weights after FBS with the bit sparsity of baseline weights, observing an average increase of 13.41%. While significantly enhancing sparsity, the accuracy loss due to the FBS algorithm remains negligible. This is because FBS leverages the characteristics of Booth encoding to filter out a large number of redundant consecutive '1's in negative values. Additionally, the introduction of bit-group reduces the impact on larger values and allows for more aggressive sparsification of less influential values.

TABLE I Accuracy Results of Different Approaches on Different Tasks

TASK	QNLI	MRPC	MNLI	QQP
Baseline	91.1	89.8	84.7	87.9
Sanger	91.0	88.7	84.8	87.9
DTQAtten	90.5	89.7	85.0	75.5
DEQ	90.8	89.6	85.2	87.7
Libra	**91.1**	**88.6**	**83.6**	**88.1**
TASK	**SST-2**	**CoLA**	**RTE**	**STS-B**
Baseline	92.8	55.7	63.2	88.5
Sanger	93.1	57.2	66.4	88.6
DTQAtten	92.2	54.3	63.0	-
DEQ	92.5	58.3	62.4	88.4
Libra	**91.9**	**57.8**	**65.7**	**88.7**

TABLE II Area and Power of Different Modules

Module	Area[mm^2]	Power[mW]
PE Array	0.247	62.53
DBQ Engine	0.004	0.102
Nonlinear - Softmax	0.13	5.63
Nonlinear - Others	0.072	1.085
SRAM Buffer	0.478	268.9
Others	0.199	38.41
Total	**1.13**	**376.66**

2) Activation DBQ. To investigate the effective utilization of activation sparsity potential by DBQ, we evaluate the impact of the ratio parameter r on accuracy loss and the associated reduction in computation across multiple datasets. Fig. 8 illustrates the percentage of computations for $Q \cdot K^T$ that can directly leverage the predicted results under different values of r. And Table III illustrates the accuracy loss. Notably, for QQP at r = 0.8, the proportion of computations that directly used predicted results reached 93.3%, with no accuracy loss. This observation underscores that within attention computation, a substantial portion of calculations are redundant, indicating that only a small subset of token pairs exhibiting high correlation significantly affect accuracy.

Sanger and DTQAtten directly eliminate computations related to token pairs with low correlation, resulting in wasted computational overhead for prediction. We argue that for token pairs with low correlation, the information provided by the prediction is sufficient to replace the result from full-precision computation without causing significant accuracy loss. When the dynamic sparsity ratio is sufficiently high, even the eliminated small values still contain a substantial amount of information. Furthermore, the computational overhead of predicting correlation increases with the input length. Therefore, DBQ reuses the predicted results, allowing fewer high-precision calculations while maintaining the same accuracy loss as Sanger and DTQAtten.

3) Put it Together. Using the optimal configuration, we integrate FBS and DBQ, presenting the final accuracy and corresponding

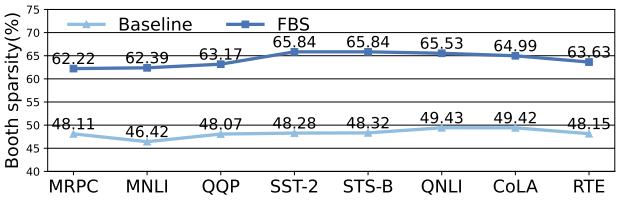


Fig. 7: Average Booth sparsity of linear layer weights across different tasks on Glue.

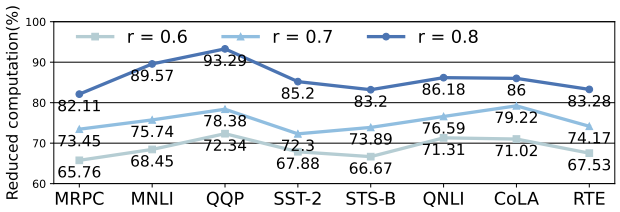


Fig. 8: Reduced computation of different r settings of DBQ.

TABLE III Accuracy Loss of Different r Settings of DBQ.

r	MRPC	MNLI	QQP	SST-2	STS-B	QNLI	CoLA	RTE
0.6	1.01%	0.89%	-6.93%	-0.71%	-3.39%	1.48%	-0.35%	-0.23%
0.7	1.14%	1.12%	-4.31%	-0.53%	-3.35%	1.4%	-0.24%	-0.14%
0.8	1.35%	1.01%	-3.75%	-0.17%	-3.96%	1.25%	-0.17%	-0.04%

speed-up for each dataset in the Table I. The average accuracy loss for the Libra sparse scheme remains under 1%.

C. Evaluation on Libra Hardware

Libra is implemented in TSMC 28nm technology, operating at 500MHz. The detailed area and power breakdown of Libra are shown in Table II. Our accelerator features an 8×8 processing array, with 408 KB of SRAM allocated for weights, inputs, temporary buffers, and their corresponding buffers. Our accelerator have a total area of $1.13mm^2$ and total power of $376.66mW$.

We contact extensive experiments on glue datasets to compare our accelerator with SOTA Transformer accelerators in terms of speedup and energy efficiency. A cycle-accurate simulator is developed to collect performance statistics. Fig. 10 presents the normalized performance with other accelerators. Experimental results show that comparing with Sanger, DTQAtten, DEQ, Libra has achieved $5.89\times$, $3.35\times$, $1.49\times$ normalized performance gain, respectively. As shown in Fig. 11, our accelerator achieves $10.82\times$, $7.41\times$, and $2.65\times$ enhancement in energy efficiency compared with Sanger, DTQAtten, and DEQ, respectively. The improvements are driven by three key factors. First, our algorithm exploits hybrid sparsity, reducing redundant computations caused by zero values and bits. And then, Libra significantly decreases the computational load in the $Q \cdot K^T$ operation, a critical step in attention mechanisms, resulting in enhanced overall efficiency. Lastly, the multi-level workload balance strategy improves the utilization of computational resources and reduces energy consumption.

Fig. 9 illustrates the performance under different task pool depths (0, 4, 8, and 16) across various tasks. We observe that as the depth increases, performance also improves. This trend can be attributed to the enhanced capacity of the task pool to manage workloads, thereby reducing the likelihood of load imbalance. However, increasing the depth also results in greater area and power consumption, as well as increased complexity in scheduling logic. Considering the trade-offs between performance and additional overhead, we set the task pool depth to 8 for optimal balance.

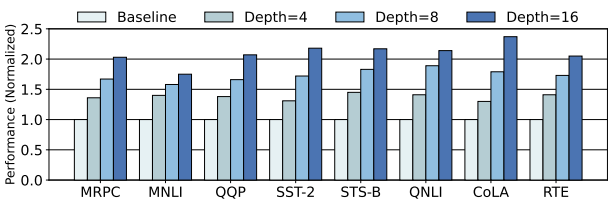


Fig. 9: Performance analysis with different task pool depths. All results are normalized to Baseline (depth=0).

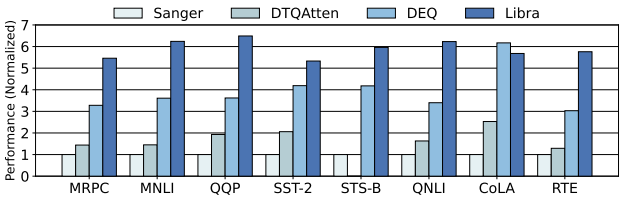


Fig. 10: Comparison of performance between Libra and other accelerators. All results are normalized to Sanger.

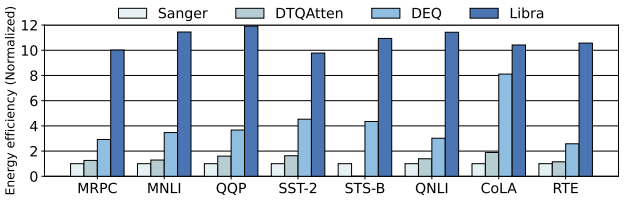


Fig. 11: Comparison of energy efficiency between Libra and other accelerators. All results are normalized to Sanger.

To validate the effectiveness of our multi-level workload balance strategy, we evaluated Libra's performance under same conditions without enabling the multi-level load balance strategy across various tasks. When the task pool depth is 0, the multi-level workload balancing strategy is not enabled. As shown in the Fig. 9, without multi-level workload balance, Libra's performance decreased by an average of $1.73\times$, indicating the strategy's effectiveness. The multi-level workload balance strategy allows Libra to achieve local workload balance within BMUs and between PEs, thereby reducing the stalling of compute units.

VI. CONCLUSION

This paper presents Libra, a software-hardware co-design that leverages hybrid sparsity to accelerate attention-based models. The bit-group-based algorithm applies sparsification to weights, and dynamically adjusting the number of bit-groups for different elements by exploiting the redundancy in the attention mechanism, resulting in a significant reduction in computation. To fully harness the potential of the hybrid sparsity and algorithm, we design a matching adaptive bit-width architecture, with multi-level optimizations to address load imbalance caused by hybrid sparsity. Experimental results demonstrate that Libra can achieve up to $1.49\times \sim 5.89\times$ speedup and $2.65\times \sim 10.82\times$ enhancement in energy efficiency.

ACKNOWLEDGEMENTS

This work was financially supported in part by National Key Research and Development Program of China under Grant 2021YFA1003602, in part by Project of Shanghai EC under Grant 24KXZNA12, and in part by National Natural Science Foundation of China (NSFC) Research Project 92373207. The computations in this research were performed using the CFFF platform of Fudan University.

REFERENCES

[1] Rajeev Balasubramonian et al. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14.2 (2017).

[2] Adam Paszke et al. Automatic differentiation in pytorch (2017).

[3] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[4] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] Alec Radford. Improving language understanding by generative pre-training (2018).

[6] Z Lan. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[7] Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[8] Alec Radford et al. Language models are unsupervised multitask learners. *OpenAI blog* 1.8 (2019).

[9] Sayeh Sharify et al. Laconic deep learning inference acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*. 2019.

[10] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[11] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[12] Thomas Wolf et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 2020.

[13] Sehoon Kim et al. I-bert: Integer-only bert quantization. In *International conference on machine learning*. PMLR. 2021.

[14] Liqiang Lu et al. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021.

[15] Shaden Smith et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990* (2022).

[16] Tao Yang et al. DTATrans: Leveraging dynamic token-based quantization with accuracy compensation mechanism for efficient transformer architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.2 (2022).

[17] Tao Yang et al. Dtqatten: Leveraging dynamic token-based quantization for efficient attention architecture. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022.

[18] Xiaolong Lin et al. AdaS: A Fast and Energy-Efficient CNN Accelerator Exploiting Bit-Sparsity. en. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA, 2023.

[19] Wenhao Sun et al. Bit-balance: Model-hardware co-design for accelerating nns by exploiting bit-level sparsity. *IEEE Transactions on Computers* (2023).

[20] Xuhang Wang et al. Deq: Dynamic element-wise quantization for efficient attention architecture. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE. 2023.

[21] Gang Wang et al. BSViT: A Bit-Serial Vision Transformer Accelerator Exploiting Dynamic Patch and Weight Bit-Group Quantization. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2024).

A High-Precision and Low-Cost Approximate Transform Accelerator for Video Coding

Zhijian Hao[1], Jiaming Liu[2], Chenlong He[2], Qi Zheng[2], Shushi Chen[2], Jinchang Xu[3], Yue Hao[1], Xiao Yan[*4] and Xiaohua Ma[*1]
[1]State Key Discipline Laboratory of Wide Band-Gap Semiconductor Technology, School of Microelectronics, Xidian University
[2]State Key Laboratory of Integrated Chips and Systems, Fudan University, [3]School of Computer Science, Peking University,
[4]Electronics Engineering Department, Xi'an University of Posts and Telecommunications
[*]Corresponding authors: Xiao Yan {xiao_yanxd@163.com} and Xiaohua Ma {xhma@xidian.edu.cn}

***Abstract*—The introduction of multiple transform types in the Versatile Video Coding (VVC) standard has yielded notable encoding gains but also imposed considerable computational burdens. Existing transform circuits of different types are typically implemented separately due to their independence, leading to substantial hardware overhead. To address this, we explore the relationship between Discrete Cosine Transform Type-2 (DCT2) and Discrete Sine Transform Type-7 (DST7) matrices and reveal a prominent diagonal aggregation phenomenon in their transfer matrix. Based on this insight, the least-squares method is applied to optimize the transfer matrix sparsity, achieving a high-precision, low-cost approximate conversion from DCT2 to DST7. Furthermore, we optimize DCT2 computation by proposing an elaborate matrix decomposition approach that allows a lightweight shift-adder unit to efficiently generate all required product terms across varying sizes. Leveraging these algorithmic optimizations, we implement a highly reusable and area-efficient approximate transform accelerator that supports sizes from 4 to 32 points and accommodates three types in VVC. Experimental results demonstrate that the proposed accelerator achieves over 44% reduction in circuit resource consumption with negligible BD-BR performance loss of just 0.53%, maintaining processing capabilities up to 8K@57 fps.**

***Index Terms*—Video coding, transform accelerator, approximate architecture, sparsity optimization, area-efficient**

I. Introduction

Transform coding has been widely integrated in video coding due to its efficient spatial decorrelation capability. Cooperating with quantization and entropy coding, it significantly improves compression rates while maintaining slight quality loss. It was proved in [1] that the Discrete Cosine Transform Type-2 (DCT2) can optimally approximate the Karhunen-Loève (K-L) transform and is also more computationally efficient, resulting in its widespread adoption in H.264 and H.265. Later, Han et al. [2] demonstrated that for images modeled by a first-order Gaussian-Markov process, the Discrete Sine Transform Type-7 (DST7) provides better decorrelation than DCT2. Therefore, the latest coding standards have introduced DST7 alongside DCT2 for more comprehensive decorrelation.

The introduction of multiple transform types provides significant coding benefits, yielding a 2.10% coding gain, but it also substantially increases computational complexity. In terms of hardware resource consumption, supporting multiple transform types requires instantiating additional transform circuits, which incurs high hardware costs. To mitigate these costs, several approaches have been proposed, grouped into three main categories. The first category focuses on **independent optimizations for DCT2**. Early studies include the Butterfly algorithm, which reduces multipliers by leveraging symmetry. Ahmed et al. [3] further reduced multiplications by decomposing DCT matrices into sparse sub-matrices. Additionally, multiplier-less techniques [4]–[6] have been developed to optimize multiplication. Recent DCT2 hardware optimizations focus on improving reusability. Hao et al. [7] proposed using smaller transform matrices as subsets of larger ones to improve reusability. Zheng et al. [8] and Hao et al. [9] introduced data rearrangement schemes that allow even-odd decomposition circuits to be reused across different transform sizes.

The second category focuses on the **independent optimization of DST7**. Notably, Discrete Cosine Transform Type-8 (DCT8), another transform type used in VVC, exhibits significant symmetry with DST7, allowing the optimization of both transformations to be reduced to the optimization of DST7. Garrido et al. [10]–[15] explored various approaches for designing efficient DST architectures on FPGA, including utilizing DSP resources for multiplication and replacing multiplication with shift-add operations. Fan et al. [16], [17] applied the RAG-n algorithm to minimize adder count, significantly reducing resource usage.

The third category focuses on the **joint optimization of both types**. These methods typically employ approximate computation strategies to reduce resource overhead. In [18], the matrix elements are modified to multiples of 2^3 to reduce multiplier size, though suffering unneglectable performance loss. Ahmed et al. [19] proposed a method to convert results of inverse DCT2 to DST7 with good coding performance. However, it requires an inverse transform in the forward transform pipeline, making it impractical for hardware implementation.

Previous studies have predominantly focused on optimizing DCT2 or DST7 architectures in isolation, or on joint optimization strategies that are impractical for real-world applications, often failing to effectively balance performance retention with hardware constraints. In this paper, we achieve a high-precision and low-cost approximate transform accelerator designed through algorithm-hardware co-optimizations. The main contributions are summarized as follows:

- **High-precision and low-cost approximate scheme for DST7.** Our analysis reveals an efficient transfer relationship between DST7 and DCT2. Leveraging this insight,

we define a transfer matrix and further optimize its sparsity with the least squares method, facilitating a high-precision result conversion from DCT2 to DST7. Notably, 79.3% multipliers can be saved for the 32-point case.

- **Matrix decomposition-based calculation scheme for DCT2.** Building upon the butterfly algorithm, we further reduce the remaining complexity by proposing an improved computation scheme for DCT2, enabling a lightweight shift-adder unit (SAU) to generate all product terms across varying size.
- **Highly-pipelined and area-efficient approximate transform architecture.** With above algorithmic optimizations, we develop a pipelined architecture that integrates multiple resource-reuse and data-parallelism strategies. Compared to related works, the proposed architecture achieves a reduction of over 44% in resource costs and delivers the highest throughput.

II. Background and Motivation

A. Background of transform

The transform scheme employed in VVC integrates three primary transform types: DCT2, DST7, and DCT8. By exploiting the separability, the two-dimensional (2D) transform is decomposed into two sequential 1D transforms, effectively reducing both storage and computational costs. Therefore, optimization efforts for transform architectures mainly focus on 1D transforms. The 1D DCT2 is defined as (1),

$$Y(k) = \sqrt{\frac{2}{N}}\varepsilon_k \sum_{n=0}^{N-1} x(n) cos\left[\frac{k(2n+1)\pi}{2N}\right],\ \varepsilon_k = \begin{cases} \sqrt{\frac{2}{N}}, k=0 \\ 1 \quad\quad , k\neq 0 \end{cases} \tag{1}$$

where k is the index of transform coefficients. $Y(k)$ is the transformed outputs, $x(n)$ is the input data, and N is the transform size. It can be expressed in matrix form as follows:

$$Y = A \times X^T, \tag{2}$$

where A is the transform matrix. In practical video coding, by scaling and rounding the A matrix while preserving its orthogonality, the integer transform matrix T can be obtained. Therefore, the integer 1D transform can be conducted as (3):

$$Y = T \times X^T. \tag{3}$$

The transformation process is inherently a computation-intensive matrix multiplication. The T varies with transform types. Thus, the optimization of the transform architecture relies on refining the corresponding matrix multiplications.

B. Motivation

The integration of multiple transform types intensifies the industry's desire for a unified architecture that supports all three transform types with the resource cost of a single type. However, the independence of DCT2 and DST7 matrices poses a huge challenge as the general multipliers-based architecture leads to heavy resource overhead. While DST7 and DCT8 share the same circuit, an efficient solution for merging DCT2 and DST7 into a joint design remains unresolved. Previous works targeting joint optimization are not efficient enough to balance coding performance with hardware friendliness, rendering them impractical for most real-world applications.

For optimizing DCT2 and DST7 jointly, we conduct an in-depth analysis of their computation processes and observe a prominent diagonal aggregation feature in the transfer matrix, motivating us to explore its sparse optimization for high-precision and cost-effective result conversion from DCT2 to DST7. Besides, we observe that after applying the butterfly transform to DCT2, the remaining matrix multiplications exhibit a beneficial feature, providing further opportunities to reduce the computational resource requirements for DCT2.

III. Hardware-Oriented Algorithmic Optimizations

A. Algorithmic Optimizations for DST7

To establish the relationship between the DCT2 and DST7, we begin with their corresponding transform processes:

$$Y_2 = T_2 \times X^T, \qquad Y_7 = T_7 \times X^T, \tag{4}$$

By transposing both sides of these equations, we obtain:

$$Y_2^T = X \times T_2^T, \qquad Y_7^T = X \times T_7^T. \tag{5}$$

Further, a transfer matrix TF is defined to reflect the conversion between the results of DCT2 and DST7:

$$Y_2^T \times TF = Y_7^T \iff X \times T_2^T \times TF = X \times T_7^T, \tag{6}$$

simplifying it as:

$$T_2^T \times TF = T_7^T. \tag{7}$$

Considering that T_2 is an orthogonal matrix, we have $T_2^{-1} = \frac{T_2^T}{2^n}$, where 2^n is introduced for integer transform. Besides, the inverse matrix of T_2^T is $\frac{T_2}{2^n}$. Substitute it into (7) and TF can be derived as (9):

$$\frac{1}{2^n} \times T_2 \times T_2^T \times TF = \frac{1}{2^n} \times T_2 \times T_7^T, \tag{8}$$

$$TF = \frac{1}{2^n} \times T_2 \times T_7^T. \tag{9}$$

Fig. 1a presents the scaling and rounding case of the transfer matrix TF, which converts the transform results from DCT2 to DST7. It is evident that the elements of TF exhibit a pronounced aggregation pattern, with the larger magnitudes concentrated along the diagonal and values farther from the diagonal approaching zero. Smaller values, representing lower significance, can be discarded to reduce computational costs while preserving transfer accuracy.

The previous observation inspires the sparse optimization of TF. This optimization problem can be formulated as a constrained optimization problem, where the objective is to minimize the error defined in (10) between the original DST7 matrix T_7 and its approximation $\widetilde{T_7}$ (11) computed from the sparse transfer matrix $\widetilde{TF}$, subject to the constraint of limiting the number of non-zero values in each column of $\widetilde{TF}$. This can be expressed mathematically as follows:

$$E(\widetilde{TF}) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (T_7(i,j) - \widetilde{T_7}(i,j))^2 \tag{10}$$

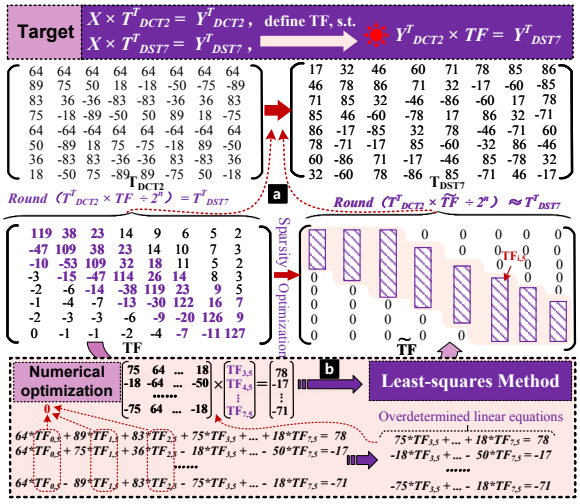


Fig. 1. The algorithmic optimizations for DST7. (a) The introduction of transfer matrix and the motivation of sparsity optimization. (b) Sparsity optimization of $TF_{i,5}$ with the least-squares method

$$\widetilde{T_7^T} = round\ (T_2^T \times \widetilde{TF} \div 2^n); \tag{11}$$

$$\begin{aligned} &\textbf{minimize}\ E(\widetilde{TF}), \\ &\textbf{subject to}\ \widetilde{TF}_{i,j} = 0,\ |i-j| > (r-1)/2, \\ &\qquad \widetilde{TF}_{i,j} \in \mathbb{Z} \cap [-2^n+1, 2^n], \end{aligned} \tag{12}$$

where r is the number of retained elements in each column of $\widetilde{TF}$. For example, when $r = 5$, as illustrated in Fig. 1a, only the diagonal and two neighboring elements on each side are preserved. Besides, note that the error $E(\widetilde{TF})$ is accumulated independently from every column, therefore, the optimization problem can be further decomposed into individual column-wise optimization tasks, which effectively reduces the optimization complexity. In this paper, we employ the least squares method to solve the constraint optimization for each column. Fig. 1b presents the optimization process for the sixth column $TF_{i,5}$. It is evident that, for $r = 5$, the system of equations becomes overdetermined. By formulating this system of equations as a matrix equation, it can be efficiently solved using MATLAB to obtain the least squares solution.

Following that, we conduct a group of experiments to determine the optimal variable r and the results are illustrated in Fig. 2. **For performance evaluation**, five video sequences with varying resolutions were randomly selected as test vectors, and the results of $r = 5$, $r = 7$, and $r = 9$ are depicted in Fig. 2a. As r increases, the Bjøntegaard Delta-Bit Rate (BD-BR) [20] loss decreases across all sequences. It is worth noting that the loss reduction between $r = 7$ and $r = 9$ is less significant than that between $r = 5$ and $r = 7$. Specifically, For *basketballDrill*, the performance loss decreases by 0.24% when r increases from 5 to 7, while the reduction from 7 to 9 is only 0.06%. Additionally, **the multiplier usage ratio** for the approximation to the exact transform is presented in Fig. 2b, where the multiplier usage increases approximately linearly with r. Taking performance and multiplier usage into consideration, $r = 7$ is selected as the optimal parameter.

Fig. 2c depicts the approximation of the first eight basis vectors of the 16-point DST7 when $r = 7$. The results clearly

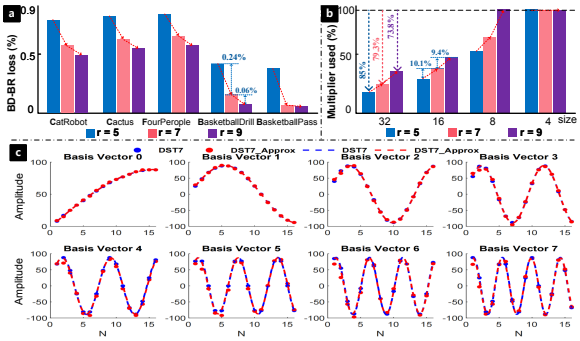


Fig. 2. (a) The Bd-BR loss of five sequences under different r. (b) The multiplier usage ratio of four sizes under different r. (c) Performance of approximation when $r = 7$.

indicate that the $\widetilde{T_7}$ derived from (11) closely matches the original matrix, underscoring the efficacy of the proposed method in balancing precision and computational complexity.

B. Algorithmic Optimizations for DCT2

The algorithmic optimizations for DCT2 are inspired by an efficient hardware design of DST7. The DST7 transform matrix exhibits a beneficial feature: its row/column coefficients are derived from a unified set, differing only in position and sign. This feature enables an efficient SAU-based matrix multiplication, where N identical SAUs generate all product terms, which are then accumulated through an adder tree, significantly reducing computational resources, as shown in Fig. 3a. However, Fig. 3b reflects that this feature doesn't appear in the DCT2 transform matrix, making direct optimization impractical. Encouragingly, after applying the classic butterfly algorithm to DCT2, its sub-matrices possess this desirable feature. As shown in Fig. 3b, the 8-point DCT2 with the butterfly algorithm is presented, which merges inputs with matching coefficients to produce the E and O vectors. These vectors are subsequently processed by sub-matrices to derive the final transformed outputs. The presence of this mentioned feature in the sub-matrices allows for further optimization of the remaining calculations.

It is worth noting that directly optimizing the sub-matrix computations with SAU-based matrix multiplication still results in substantial resource consumption. This stems from the diversity in sub-matrix elements across various sizes, requiring distinct SAUs to be instantiated for each size. To enhance SAU reuse across multiple sizes, we introduce a matrix decomposition-based calculation scheme for DCT2, as depicted in Fig. 3.c, defined as follows:

$$T = Q \times s + R, \tag{13}$$

here, s represents a scaling factor with a power-of-two value (2^n), determined by balancing the complexities of Q and R. As the s increases, the complexity of Q decreases and that of R increases. For each entry in $T(i,j)$, its decomposition equation is given by (14):

$$\begin{cases} T(i,j) = & Q(i,j) * s + R(i,j), \\ |R(i,j)| \leq & s/2. \end{cases} \tag{14}$$

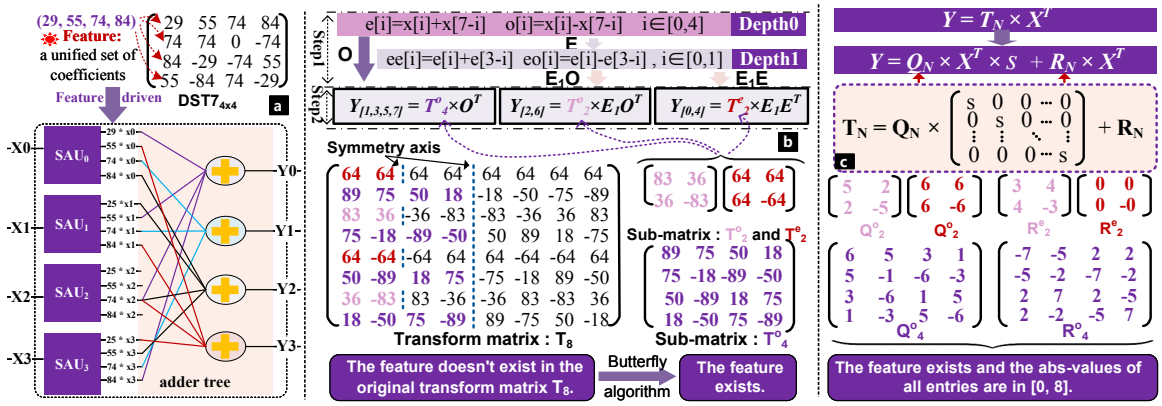


Fig. 3. (a) The beneficial feature and the SAU-based matrix multiplication. (b) The butterfly algorithm and sub-matrix of DCT2. (c) The proposed calculation scheme based on matrix decomposition.

Given that the maximum entry of the DST7 and DCT2 matrix is 90. We can acquire the range of absolute value in Q and R matrices presented in Table I. Clearly, when s is equal to 16, these two matrices share a similar level of complexity. In this case, the optimized calculation scheme for DCT2 can be described as (15):

$$Y^T = Q \times X^T \times s + R \times X^T, \tag{15}$$

where the X is the inputs, and the computationally intensive matrix operation is carried out by two simpler matrix multiplications and shift operations. The limited range of Q and R enables a lightweight SAU circuit with output $\{1x, 2x, 3x, 4x, 5x, 6x, 7x, 8x\}$ to generate all required product terms, improving the reusability dramatically.

IV. The Hardware Transform Accelerator Implementation

A. Overall Architecture Design

We implement a highly parallel, extensively reusable, and deeply pipelined transform accelerator in Fig. 4, capable of processing 32 samples per cycle. For transform sizes below 32 points, multiple input sets are combined to create a 32-point input. This approach processes 8 groups of 4-point inputs, 4 groups of 8-point inputs, 2 groups of 16-point inputs, or a single group of 32-point inputs in each cycle. Such parallel processing achieves the highest level of concurrency among existing research.

The proposed accelerator features a deeply pipelined architecture with six stages: even-odd decomposition, lightweight SAU calculation, selector tree, adder tree, DCT2 result integration, and transfer calculation. The initial five stages focus on DCT2, whereas the last stage handles DST7. The following sections provide a detailed description of the data flow within each pipeline stage, highlighting the strategies for parallelism and reusability.

TABLE I
The Range of Absolute Value in Q and R Matrices

Scale factor s	4	8	16	32	64
abs Q range	[0,23]	[0,11]	[0,6]	[0,3]	[0,1]
abs R range	[0,2]	[0,4]	[0,8]	[0,16]	[0,32]

B. Even-Odd Decomposition Stage

Stage0 is the even-odd decomposition stage, breaking down inputs into E and O vectors via the butterfly algorithm, as shown in Fig. 3b. For this purpose, a custom decomposition circuit was designed, specifically for processing 8 groups of 4-point inputs, as illustrated in Fig. 4b.

Reusability. Applying this decomposition circuit to other input sizes directly leads to data mismatches. For instance, with a set of 32-point input, the correct $e[0]$ should be derived by adding X_1 and X_{31}, but the circuit would erroneously compute it as the sum of X_1 and X_3, leading to a malfunction. To ensure functional correctness and enable reuse of the decomposition circuit across varying input sizes, we propose a data rearrangement scheme depicted in Fig. 5, where X_{31} is repositioned to the location of X_3 accordingly. This strategy eliminates data mismatches and allows a single decomposition circuit to support four input sizes, effectively reducing the adder resource consumption by 75%.

C. Lightweight SAU Calculation Stage

Stage1 is the SAU computation stage, generating all the product terms involved in the matrix multiplications.

Reusability. As shown in Table I, the absolute value range of all matrix elements is confined within [0, 8]. Therefore, we design a lightweight SAU embedding with only three adders to efficiently generate the product terms, as shown in Fig. 4a. The reuse strategy of this pipeline stage is twofold: first, the SAU can be reused across different transform sizes; second, there is no need to instantiate separate SAU circuits for the Q and R sub-matrices, as they share a similar value range.

Parallelism. The accelerator instantiates 32 lightweight SAUs in total, which operate in parallel to generate the product terms corresponding to the 32 inputs.

D. Selector Tree

Stage2 is the selector tree stage, which routes the product terms produced by the SAUs. The appropriate SAU outputs are selected based on the matrix elements and forwarded to the next stage via Adder-tree Selectors (ATS), as depicted in Fig. 4c. Each ATS includes multiple selectors, controlled by the *size* signal.

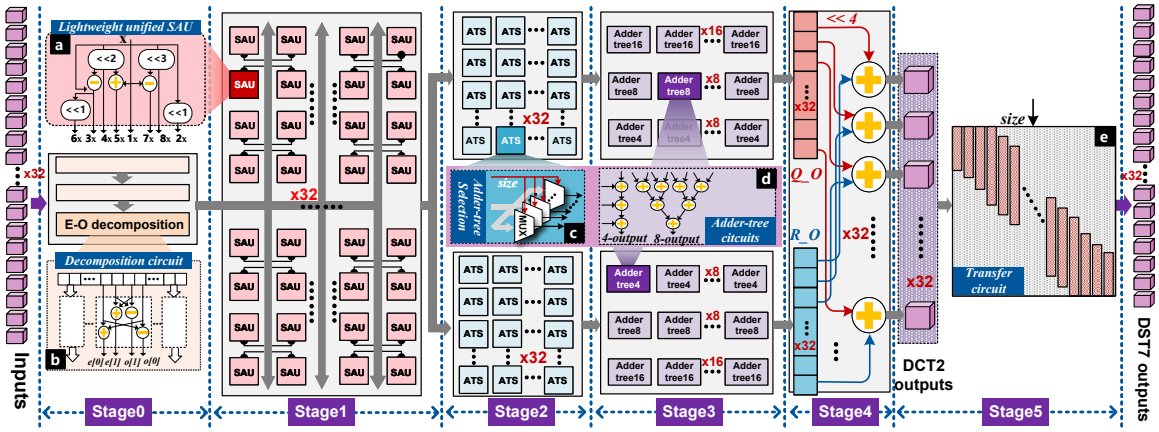


Fig. 4. The proposed architecture. (a) Lightweight unified SAU. (b) Decomposition circuit. (c) Adder tree selection. (d) Adder-tree circuit. (e) Transfer circuit.

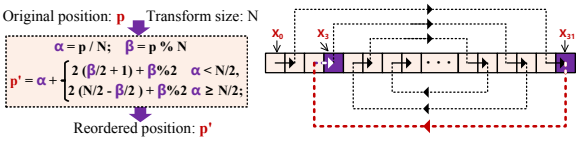


Fig. 5. The proposed data rearrangement scheme.

Parallelism. In this stage, the selection process for the sub-matrices Q and R is performed in parallel. Each of them requires 32 ATS units, resulting in a total of 64 ATS units working concurrently.

E. Adder Tree and DCT2 Result Integration Stages

Stage3 is the adder tree stage, where the selected product terms are accumulated.

Parallelism. This stage employs three types of adder trees: adder-tree4, adder-tree8, and adder-tree16, as depicted in Fig. 4d for efficiently accumulating 4, 8, and 16 inputs, respectively. In the original DCT2, each output is obtained by accumulating 32 product terms, which would typically demanding 32 adder-tree32. However, the butterfly algorithm reduces the number of multiplications significantly. For the 32-point case, after the even-odd decomposition, the actual sub-matrices involved in the computation are O, EO, EEO, and EEE, with dimensions of 16×16, 8×8, 4×4 and 4×4, respectively. Consequently, instead of 32 adder-tree32, only 16 adder-tree16, 8 adder-tree8, and 8 adder-tree4 are needed. Meanwhile, the sub-matrices Q and R are processed concurrently, the number of adder-tree circuits is doubled. Thus, the proposed accelerator requires 32 adder-tree16, 16 adder-tree8, and 16 adder-tree4 for parallel accumulation.

Stage4 is the DCT2 result integration stage, where the final DCT2 results are computed using the outputs of sub-matrices Q and R, as defined in (15).

F. Transfer Calculation Stage

Stage5 is the transfer calculation stage,where the DCT2 results are approximated to DST7 results through sparse matrix multiplication, as defined in (6). This stage employs 212 multipliers, significantly fewer than $32 \times 32 = 1024$ multipliers needed for precise DST7 calculation. This approximation strategy achieves a 79.3% reduction in multiplier consumption.

TABLE II
COMPARISON OF THE ALGORITHMS

	DCT2	TCSVT [19]	ICIP [21]	ISCAS [18]	Ours
Performance	Evaluating metric: **BD-BR (%)**				
Hardware Friendly	✓	×	✓	✓	✓
A1 & A2	1.36	0.20	1.13	1.79	0.69
B	1.54	0.14	1.32	1.16	0.74
C	2.97	0.06	0.96	0.93	0.23
D	2.37	-	0.99	0.96	0.35
E	2.24	0.18	0.95	1.25	0.65
Average	**2.10**	**0.15**	**1.07**	**1.22**	**0.53**

V. EVALUATION

A. Experimental Setup

Performance evaluations were carried out using the VVC reference software VTM17.0 under all-intra configurations, employing BD-BR as the evaluation metric. The accelerator was designed in Verilog HDL and synthesized using Global Foundry's 28nm standard cell library to measure area, frequency, and power consumption metrics.

B. Coding performance comparison

To validate the effectiveness of the proposed algorithm, a comparison with other approximate transformation algorithms was conducted, as shown in Table II. When the transformation type in VVC is restricted to DCT2, coding performance drops by 2.10% relative to the use of multiple transform types. Significantly, applying the proposed approximation method within the multiple transform framework minimizes this loss, reducing it to just 0.53%. This ensures that most of the coding benefits of the original approach are maintained.

Approximation algorithms in existing research can be divided into two types. **Neglecting hardware constraints** and **Hardware-friendly algorithms**. The first type often overlooks practical hardware limitations in algorithm design. For example, [19] proposes a high-precision private approximation architecture. Despite its high accuracy, it necessitates embedding an inverse transform circuit in the forward pipeline and modifying the decoder, which doubles the circuit area and renders it unsuitable for most embedded applications. The second type, including [18], [21], and the proposed algorithm, is tailored to be hardware-friendly, allowing easy

TABLE III
COMPARISON OF DIFFERENT TRANSFORM ARCHITECTURES

Design	TCSVT [22]	TCSVT [23]	Ours	TCSVT [16]	ISCAS [18]	Ours	TCE [24]	TVLSI [8]	ICASSP [25]	Ours
Technology(nm)	90	90	28	65	65	28	90	65	28	28
Frequency (MHz)	250	187	769	384	333	769	345	476	600	769
Parallelism	8	16	32	32	2∼8	32	8	32	2	32
Gate Count	163K	88K	132K	275.4K	30.7K	51.9K	136.4K	329K	96.8K	184.4K
Power (mW)	33.72	17.04	42	68.6	*NA*	16.5	*NA*	95.2	*NA*	58.7
NGC	20.375	5.5	**4.125**	8.61	3.838 ∼ 15.35	**1.628**	7.05	10.281	48.4	**5.763**
NP	0.135	0.091	**0.054**	0.179	-	**0.021**	-	0.2	-	**0.076**
Sizes	4∼32	4∼32	4∼32	4∼32	4∼32	4∼32	4,8	4∼32	4∼64	4∼32
Types	DCT2	DCT2	DCT2	DCT8 DST7	DCT8 DST7	DCT8 DST7	DCT2/5/8 DST1/7	DCT2 DST7/DCT8	DCT2 DST7/DCT8	DCT2 DST7/DCT8

integration into the forward transform pipeline without decoder modifications.While [18] and [21] are hardware-efficient, they result in coding performance losses of over 1%. In comparison, our method cuts this loss in half, highlighting its enhanced approximation accuracy.

C. Hardware Implementation Comparison

To demonstrate the hardware superiority of the proposed accelerator, a comparison with existing works is presented in Table III, highlighting resource requirements, throughput, and power metrics.

Resource. With a parallelism of 32 samples per cycle and a gate count of 184.4K, the proposed accelerator significantly outperforms existing designs. Compared to [8], which requires 329K gates for the same parallelism, our design reduces resource usage by 44%. Moreover, while [16] supports only two transform types at the same throughput with 275.4K gates, our accelerator achieves a 33% reduction in gate count despite supporting more transform types, thanks to advanced algorithmic optimizations.

Since different designs have varying specifications, the level of data parallelism they support may differ. To facilitate a fairer comparison, we introduce the Normalized equivalent Gate Count (NGC), defined in (16), as a metric for evaluating design efficiency. Then, we provide a comparison of the NGC for DCT2 and DST7/DCT8 across various works.

$$NGC = Gate\ Count\ /\ Parallelism\ . \tag{16}$$

For DCT2, our accelerator achieves a 32-point data parallelism with only 132.5K equivalent gates, resulting in the NGC of 4.125. The NGC of works [22] and [23] in the comparison list are 20.375 and 5.5, respectively. Compared to them, our accelerator reduces the normalized resources for DCT2 by **77.9%** and **25%**, respectively.

For DST7/DCT8, the computational resources in our work are primarily allocated to the transfer calculation stage, requiring 51.9K gates with an NGC of 1.628. Two designs, [16] and [18], are compared. The design in [16] achieves an NGC of 8.61 with 275.4K gates for 32-point parallelism. Compared to this, our design reduces normalized resources by 81.1%. For work [18], which supports variable throughput with a maximum parallelism of 8 points and an NGC ranging from 3.838 to 15.35, our accelerator achieves a normalized resource reduction of **57.6%** to **89.4%.**

The designs in [24], [8], and [25] support three transform types in VVC, with NGC values of 17.05, 10.281, and 48.4, respectively. The proposed design, which also supports three transform types with a constant and maximum parallelism of 32-point, achieves an NGC of 5.763. This represents a reduction in normalized resources of **66.2%**, **44%**, and **88%** compared to these works.

Throughput. Beyond area efficiency, the proposed architecture also delivers the highest processing speed among all designs. It achieves superior parallelism and operating frequency compared to existing works, making it capable of supporting real-time encoding for **8K@57fps.**

Power. The power efficiency of the proposed accelerator is evaluated using frequency-normalized dynamic power (NP) as defined in [22]. Table III indicates that, compared to the works reporting power consumption, the proposed accelerator achieves NP savings of over **40.7%**, **88.3%**, and **62%** for DCT2, DST7, and three types, respectively.

VI. CONCLUSION

This paper presents a high-precision, cost-effective approximate transform accelerator, supporting three transform types and sizes ranging from 4 to 32 points. Following the principle of algorithm-hardware co-optimization, an approximate algorithm based on a sparse transfer matrix is proposed for DST7, and a matrix decomposition-based algorithm is proposed for DCT2. Furthermore, based on these two algorithms, a deeply pipelined, area-efficient, and highly parallel accelerator is designed. Evaluation results demonstrate that the proposed design offers significant advantages in resource, throughput, and power consumption.

ACKNOWLEDGMENT

This work is supported by the China Postdoctoral Science Foundation under Grant Number 2024M762536 and National Natural Science Foundation of China under Grant 62304179. Corresponding authors: Xiao Yan and Xiaohua Ma.

REFERENCES

[1] R. Clarke, "Relation between the karhunen loève and cosine transforms," *IEE Proceedings F (Communications, Radar and Signal Processing)*, vol. 128, pp. 359–360(1), November 1981.

[2] J. Han, A. Saxena, V. Melkote, and K. Rose, "Jointly optimized spatial prediction and block transform for video and image coding," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 1874–1884, 2012.

[3] A. Ahmed, M. U. Shahid, and A. u. Rehman, "N point dct vlsi architecture for emerging hevc standard," *VLSI Design*, vol. 2012, no. 1, p. 752024, 2012.

[4] W. Zhao, T. Onoye, and T. Song, "High-performance multiplierless transform architecture for hevc," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 1668–1671.

[5] M. Masera, G. Masera, and M. Martina, "An area-efficient variable-size fixed-point dct architecture for hevc encoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 1, pp. 232–242, 2020.

[6] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, "Efficient integer dct architectures for hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 1, pp. 168–178, 2014.

[7] Z. Hao, F. Xu, G. Xiang, P. Zhang, X. Zeng, and Y. Fan, "A multiplier-less transform architecture with the diagonal data mapping transpose memory for the avs3 standard," in *2021 IEEE 14th International Conference on ASIC (ASICON)*, 2021, pp. 1–4.

[8] Z. Hao, H. Sun, G. Xiang, P. Zhang, X. Zeng, and Y. Fan, "A reconfigurable multiple transform selection architecture for vvc," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 5, pp. 658–669, 2023.

[9] M. Zheng, J. Zheng, Z. Chen, L. Wu, X. Yang, and N. Ling, "A reconfigurable architecture for discrete cosine transform in video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 3, pp. 810–821, 2020.

[10] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, and C. Sanz, "A high performance fpga-based architecture for the future video coding adaptive multiple core transform," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 1, pp. 53–60, 2018.

[11] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, C. Sanz, and P. Paz, "An fpga-based architecture for the versatile video coding multiple transform selection core," *IEEE Access*, vol. 8, pp. 81 887–81 903, 2020.

[12] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, and C. Sanz, "A 2-d multiple transform processor for the versatile video coding standard," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 274–283, 2019.

[13] A. Kammoun, S. Ben Jdidia, F. Belghith, W. Hamidouche, J. F. Nezan, and N. Masmoudi, "An optimized hardware implementation of 4-point adaptive multiple transform design for post-hevc," in *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, 2018, pp. 1–6.

[14] A. Kammoun, W. Hamidouche, F. Belghith, J.-F. Nezan, and N. Masmoudi, "Hardware design and implementation of adaptive multiple transforms for the versatile video coding standard," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 4, pp. 424–432, 2018.

[15] Z. Zhang, X. Zhao, X. Li, L. Li, Y. Luo, S. Liu, and Z. Li, "Fast dst-vii/dct-viii with dual implementation support for versatile video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 1, pp. 355–371, 2021.

[16] Y. Fan, Y. Zeng, H. Sun, J. Katto, and X. Zeng, "A pipelined 2d transform architecture supporting mixed block sizes for the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 9, pp. 3289–3295, 2020.

[17] F. Yibo, K. Jiro, S. Heming, Z. Xiaoyang, and Z. Yixuan, "A minimal adder-oriented 1d dst-vii/dct-viii hardware implementation for vvc standard," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, 2019, pp. 176–180.

[18] Y. Zeng, H. Sun, J. Katto, and Y. Fan, "Approximated reconfigurable transform architecture for vvc," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[19] A. Kammoun, W. Hamidouche, P. Philippe, O. Déforges, F. Belghith, N. Masmoudi, and J.-F. Nezan, "Forward-inverse 2d hardware implementation of approximate transform core for the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 11, pp. 4340–4354, 2020.

[20] G. Bjøntegaard, "Calculation of average psnr differences between rd-curves," 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:61598325

[21] L. Chang, Z. Liu, X. Ji, and D. Wang, "Coding sensitive based approximation algorithm for power efficient vbs-dct vlsi design in hevc hardwired intra encoder," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3001–3005.

[22] M. Masera, M. Martina, and G. Masera, "Adaptive approximated dct architectures for hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 12, pp. 2714–2725, 2017.

[23] M. Masera, G. Masera, and M. Martina, "An area-efficient variable-size fixed-point dct architecture for hevc encoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 1, pp. 232–242, 2020.

[24] A. C. Mert, E. Kalali, and I. Hamzaoglu, "High performance 2d transform hardware for future video coding," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 2, pp. 117–125, 2017.

[25] I. Farhat, W. Hamidouche, A. Grill, D. Menard, and O. Déforges, "Lightweight hardware implementation of vvc transform block for asic decoder," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 1663–1667.

Intermittent Systems at Small Scale: Execution Model and Design Guidelines

Youngbin Kim and Yoojin Lim
yb.kim@etri.re.kr, yoojin.lim@etri.re.kr
Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea

***Abstract*—Intermittent systems require software support to execute tasks amid frequent power failures. In designing such techniques, software designers rely on execution models that abstract hardware-level operations. In this paper, we propose an execution model that more accurately describes emerging intermittent systems with small energy storage. Our evaluation shows show that systems designed based on the traditional models can be up to 5.62x less power-efficient than expected and may result in unsafe checkpoint operations. Our design guidelines enhance the performance of existing static and dynamic checkpoint techniques by 3.04x and 2.85x on average, respectively.**

***Index Terms*—Intermittent Computing, Batteryless System.**

I. Introduction

Batteryless systems are emerging as a promising future platform of Internet-of-Things (IoT) devices. These systems adopt a small capacitor as an energy storage and operate by harvesting power from environmental sources. This setup effectively addresses challenges associated with traditional battery-based systems, such as need for human intervention for recharging or replacement [1] and harmful environmental impacts [2]. They are also known as intermittent systems, since the computation happens intermittently during short periods only when there exist sufficient power to compute.

Intermittent systems require software supports to sustain long-running executions across power failures. During operation, volatile data (e.g., registers or SRAM data) must be saved to Non-Volatile Memory (NVM) through a process called checkpointing. When power is restored, this saved state is recovered to allow operations to resume the execution from the last checkpoint (recovery). In designing these state retention techniques, software designers rely on an *execution model* that abstracts hardware-level operations and represents behavior of intermittent systems necessary for software design.

Fig. 1 illustrates such execution model commonly adopted in the literature [3]–[10]. As energy accumulates, the voltage of the capacitor gradually increases. Once the voltage reaches the power-on threshold V_h, the collected power is supplied to the system. The system begins operation at this point, and execution is halted when the capacitor voltage reaches the power-off threshold V_l. Software designers aim to leverage this execution model to implement intermittent systems at minimal cost (e.g., by executing checkpoints just before reaching the power-off threshold [4], [5], [11]–[13]).

This work was supported by IITP grant funded by the Korea government (MSIT) (No.2021-0-00360 and RS-2024-00438551).

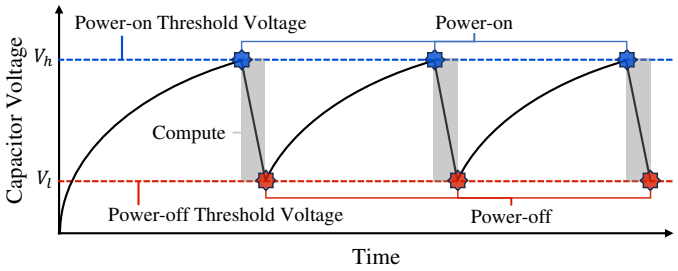


Fig. 1: Traditional execution model of intermittent systems.

In the meantime, recent research on intermittent systems is increasingly exploring shorter operation times by using smaller capacitors. Operating on small capacitors is generally desirable, as it reduces device volume and enhances the responsiveness by enabling the system to wake up more frequently [5], [14], [15]. As a result, recent studies have targeted operation times in the range of tens of milliseconds [1], [13], [16]–[20] or even microseconds [16], [17]. However, as energy storage sizes decrease, the traditional execution model is failing to provide an accurate abstraction of actual execution behavior. The major source of this discrepancy is the buffering effects of the system's inherent capacitance, mostly coming from its decoupling capacitors. This factor has been overlooked in the traditional model, as the inherent capacitance was considered negligible compared to the main energy storage.

Decoupling capacitors are on-board capacitors that act as energy buffers. They are mandatory since the buffered energy prevent transient voltage drop when the system suddenly draws a large current, such as during checkpointing (Sec. II-A). However, their buffering effects also introduce discrepancies between the execution model and actual behavior. For example, during power-on, they rapidly charge from the energy storage, making capacitor voltage an unreliable estimate of available energy. This buffered energy also allows the system operate for a while at sub-normal voltages after the power supply is stopped. Additionally, between power cycles, decoupling capacitors discharge due to the resistance of the system, considerably lowering the power efficiency. In systems with smaller capacitors, these effects dominate the behaviors that are modeled in the traditional execution model. Consequently, highly efficient checkpoint techniques according to the traditional model may introduce substantial power overhead and even correctness issues in small-scale systems.

In this paper, we propose a new execution model for

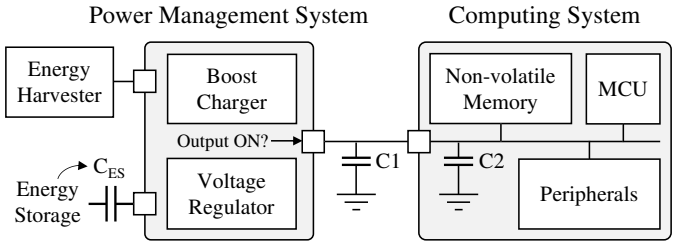


Fig. 2: A typical hardware setup of intermittent systems.

intermittent systems which accounts for the buffering effects of decoupling capacitors. In Sec. II, we demonstrate that understanding this model is critical for software designers: intermittent systems designed upon the traditional model can be up to 5.62x more energy-inefficient than expected and may fail to predict power-off timings accurately, leading to unsafe checkpointing. In Sec. III, we present design guidelines to implement efficient intermittent systems with small energy storages, leveraging insights from our model. Our guidelines include designs which improve end-to-end execution latencies by 3.04x in static and 2.86x in dynamic checkpointing schemes on average, without incurring additional overhead.

II. Detailed Intermittent Execution Model

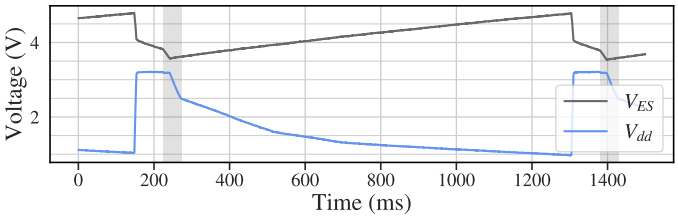
In this section, we describe our execution model and its implications for software design. Sec. II-A introduces the target architecture and the reference system used for evaluations. Sec. II-B presents our execution model, derived from key observations obtained through experimental results. In the following three sections, we discuss how this model affects both the power efficiency and correctness of software design. Finally, in Sec. II-F, we evaluate the effectiveness of our model across systems with various architectural configurations.

A. Target Architecture and Reference System

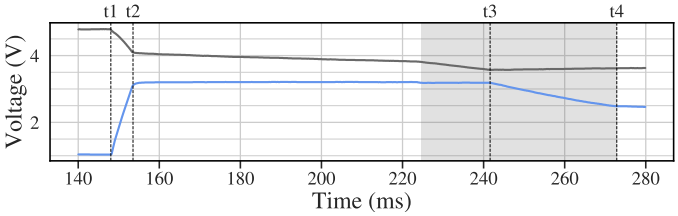
A typical intermittent system consists of two main components: a power management system and a computing system, as shown in Fig. 2. The power management system collects incoming energy into storage (C_{ES}) and supplies a stable-voltage current to the computing system. The computing system equips NVMs along with an MCU and peripherals, and utilize the NVMs for state retention between power failures.

This setup includes two notable decoupling capacitors that affect the execution model of intermittent systems. The first one (C1) is located in the power management system, required by voltage regulators to ensure stable operation. The second capacitor (C2) is part of the computing system and used to stabilize the operating voltage against sudden current draw.

Recent studies have increasingly explored 32-bit architectures for computing systems [17], [19], [21]–[29], as emerging applications on intermittent systems, such as Deep Neural Networks (DNNs) [7], [27]–[36], demand greater computational capabilities [14], [37]. In this context, we employ a custom-built board featuring a 32-bit ARM Cortex-M33 processor (STM32L5, operating at 16Mhz) with 512KB of Ferroelectric RAM (FRAM, Infineon FM22L16) as our reference system.



(a) Voltage traces for one power cycle.



(b) Voltage traces of the first execution cycle.

Fig. 3: Voltages trace of energy storage (V_{ES}) and V_{dd}.

For the power management system, we use a TI BQ25570-based board configured with V_h = 4.9V and V_l = 3.4V. We empirically select 22uF and 220uF capacitors for C1 and C2, respectively, as smaller capacitors fail to provide a reliable voltage for checkpoint and recovery. Sec. II-F evaluates the generality of our model across different architectures, such as systems with different NVM (e.g., Magnetic RAM, MRAM) and a 16-bit core (e.g., MSP430).

B. Execution Model

To derive a general execution model with the effects of decoupling capacitors, we first present a sample measurement from our reference system. In this paper, we denote the voltage of the energy storage C_{ES} as V_{ES} and the MCU operating voltage as V_{dd}. To achieve an operation time of 50 ms under 1.5mA current supply, we use a 470uF capacitor for C_{ES}. Fig. 3a illustrates the voltage traces of V_{ES} and V_{dd} over a single power cycle. Note that V_{dd} is maintained by decoupling capacitors once the power supply from the power management system stops. The shaded areas represent the periods that system executes the application code.

Fig. 3b presents the first execution cycle in more detail. It reveals several differences between the traditional execution model and the actual operation. Among them, we highlight three key observations that affect software design decisions:

- **O1**: The capacitor voltage (V_{ES}) drops rapidly to charge decoupling capacitors when the system wakes up ($t1$–$t2$).
- **O2**: System operates at sub-normal voltage using decoupling capacitors, even after power supply stops ($t3$–$t4$).
- **O3**: Decoupling capacitors discharge while the system is powered off (after $t4$, as shown in Fig. 3a).

Fig. 4 illustrates our detailed execution model, incorporating these key observations. When V_{ES} reaches V_h, the voltage experience a rapid drop due to the buffering effects (①), instead of gradual decline. After initialization (②), the system begins execution at normal operating voltage (③), 3.3V for example. When the voltage hits V_l, the power supply stops

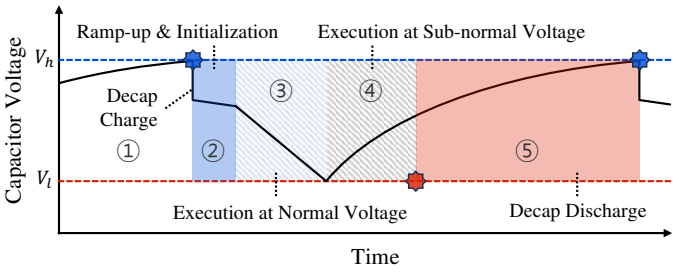


Fig. 4: Detailed execution model of intermittent systems.

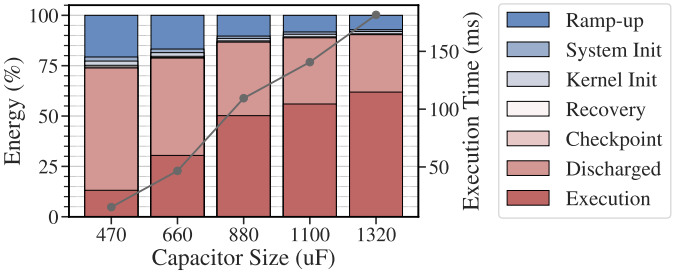


Fig. 5: Distribution of energy consumed in a power cycle.

but system now starts to operate using the buffered energy (④). Since the voltage of the decoupling capacitors decreases as they discharge, the system executes at sub-normal voltage until it reaches the voltage level it cannot operate (e.g., 2.5V in Fig. 3). Finally, until the next power-on event, the remaining energy in decoupling capacitors continues to discharge (⑤).

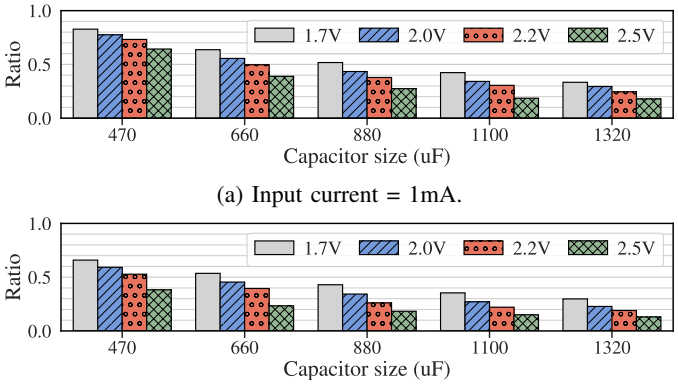
When designing intermittent systems, particularly those utilizing small capacitors, understanding the effects described by this model is critical. In the following sections, we discuss impacts of our model to software design in more detail.

C. Impact on Power Efficiency

The traditional model implies that the energy consumed between V_h and V_l is entirely used in the computing system. However, our model reveals that considerable energy is used for charging the decoupling capacitors (**O1**) and dissipated during power-off durations (**O3**). This indicates that much smaller energy may be used for the useful computation compared to the designer's expectation.

Fig. 5 shows the distribution of the energy consumption for each stage of operation within one power cycle, averaged over 50 executions, where 1mA of input current is provided at 1.9V. The x-axis represents the size of C_{ES} and the line in the secondary axis represents the average operation times for application code. The checkpoint is executed by the interrupt from the power management system [4], [11], [20], [38], [39], which is generated when V_{ES} reaches V_l (3.4V). Note that this is the most efficient point for checkpoint execution according to the traditional model (i.e., just before the poweroff).

The results shows that significant energy is wasted in the decoupling capacitors. For example, in 470uF case, 60.7% of the energy is lost during the power-off duration (denoted as *Discharged*), leaving only 13.1% of the energy for computation. While the ratio of *Discharged* decreases with larger C_{ES},



(a) Input current = 1mA.

(b) Input current = 3mA.

Fig. 6: Ratio of sub-voltage operations in total execution time.

it remains substantial; for example, in the 1320uF case, 28.5% of energy is discharged, which is still non-negligible.

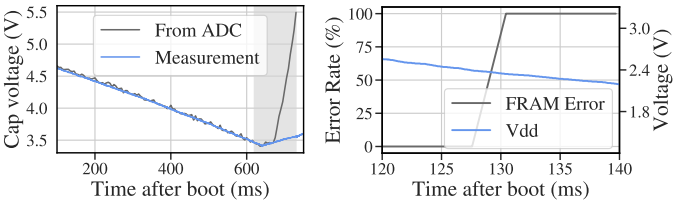
This is because the discharging behavior can be modeled as an RC-discharging circuit (i.e., $q = CVe^{-\frac{t}{RC}}$), which exhibits an exponential discharge rate. Indeed, 50% of the energy is discharged within the first 161 ms in our measurements. Since recharging C_{ES} takes 2.13 secs even in 470uF configuration, most of the buffered energy is lost before the next power-on, regardless of the size of C_{ES}. As a result, the energy loss ratio due to discharging is larger with smaller capacitors.

Another important observation is the error introduced by the traditional model. The traditional model expects both the energies, *Execution* and *Discharged*, are used for computation. This introduces huge errors, up to 5.62x in 470uF setup, for example. In the same context, the traditional model predicts that using a 470uF capacitor for C_{ES} instead of a 1320uF would result in only 1.22x overhead in energy efficiency, while the actual difference is 4.71x. This can significantly mislead system designers when they select capacitor sizes by considering tradeoffs between overall efficiency and reactiveness. In Sec. III, we explore strategies to minimize the inefficiencies caused by discharging when designing software techniques.

D. Impact on Predicting Power Failures

According to the traditional model, system states should be saved to NVM before V_{ES} reaches V_l, as the system is expected to halt at this point. On the other hand, our model shows that the system may continue operating using the energy stored in the decoupling capacitors (**O2**). Since modern MCUs can operate across a wide range of supply voltages (e.g., 1.7V to 3.6V in STM32L5 and MSP430), the computing system operates until the voltage of decoupling capacitors drops to the minimum operating level. This makes V_{ES} not a reliable indicator for the imminent power-off.

Fig. 6 presents the ratio of the times executed under sub-normal voltages to the total execution times, averaged over 30 measurements. The x-axis represents the sizes of C_{ES} and the colors indicate the voltage levels at which the system stops operation. We evaluate a range of stop voltages from 1.7V to 2.5V since not all components in the computing system



(a) Analog-Digital Converter. (b) External FRAM.

Fig. 7: Incorrect operations at sub-normal voltages.

may function at the lowest voltage level (Sec. II-E). Also, we examine two cases with input currents of 1mA (Fig. 6a) and 3mA (Fig. 6b), to assess the impact of input power.

The figure shows that a significant portion of MCU operation occurs at sub-normal voltages. For example, when 470uF capacitor is used at 1mA input current (Fig. 6a), 82.8% of computation takes place *after* the power-off threshold. This ratio decreases as the system stops earlier (reducing sub-voltage operation time) or the input current increases (extending operation time at normal voltage). However, at least 13.0% of computations are operated at sub-normal voltages even in highly optimistic configurations (1320uF in Fig. 6b).

These values can be directly translated to the inefficiencies of the systems based on the traditional model. For example, in the case of 470uF with a input current of 1mA, systems executing checkpoint at V_l may operate 16.3 ms. However, the system could operate for an additional 29.4 ms if the checkpoint can be delayed until 2.5V. At the next power-on, the decoupling capacitors discharge to similar voltage levels in both cases, as discussed in Sec. II-C. As a result, failing to utilize the buffered energy at sub-normal voltages introduces significant power inefficiency. In Sec. III-B, we validate this aspect and propose methods to fully utilize the buffered energy.

E. Impact of Sub-normal Voltage Execution

The traditional model leads software designers to assume that the system is executed under a stable voltage. However, a significant portion of execution may happen after the power-off threshold at sub-normal voltages (**O3**), as discussed in Sec. II-D. Being aware of this is crucial to software designers since analog components and peripherals may function differently at sub-normal voltages. Two relevant examples are Analog-Digital Converters (ADCs) and external NVMs.

ADCs are commonly used to determine when to execute a checkpoint by reading V_{ES}. It quantizes the input analog voltage into discrete 2^n values, ranging from 0 to the given reference voltage, where n is a resolution. Using smaller reference voltage increases sensitivity of ADC at the cost of reduced representation range.

Fig. 7a shows the behavior of ADCs, where the execution in sub-normal voltages are depicted in gray. Because STM32L5 uses V_{dd} as a reference voltage, accessing the ADC under sub-normal voltages produces inconsistent results. As shown in the figure, the ADC returns values higher than the actual measurements since its representation range decreases as V_{dd} drops. Consequently, ADC may mislead the system into overestimating the energy in C_{ES} during sub-normal voltage executions, potentially leading to checkpoint miss and a loss of progress for the entire power cycle.

TABLE I: Architectures for generality evaluation

	Core	Core Freq.	Capacitance (uF)			Current	Memory
			C1	C2	C_{ES}		
A1	STM32L5	16MHz	22	220	1,320	3mA	MRAM (off-chip)
A2	MSP430FR5994	8MHz	22	10	40	100uA	FRAM (on-chip)

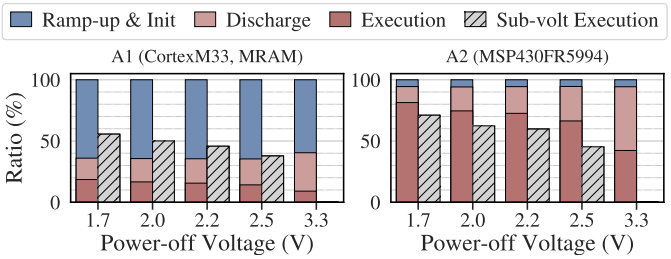


Fig. 8: Energy breakdown and the ratio of sub-voltage operations in different architectures.

Also, intermittent systems typically designed to operate with peripherals such as sensors [5], [40]–[42], wireless communication modules [6], [43], [44] or external NVMs [6], [23]–[25], which have their own minimum operating voltage requirements. Fig. 7b illustrates the error rate of FRAM in the reference system at different voltages, showing FRAM cannot operate reliably below 2.4V. Since the system continues operating until it reaches the lowest MCU operation voltage (e.g., 1.7V), software designers must ensure that peripherals are accessed only at safe voltage levels. Failing to do so can result in corrupted sensor data or unsafe checkpointing.

F. Sensitivity to Architectural Designs

To evaluate the generality of the proposed model, we assess it across two additional architectural setups. Table I shows the detailed parameters of the target architectures. A1 shares the same configuration as the reference system but equips MRAM (Everspin MR5A16ACYS35), which is gaining attention as a next generation NVM [6], [14], [17], [23], instead of FRAM. Second target is MSP430 equipped with on-chip FRAM, a widely adopted 16-bit platform in intermittent system research. For both systems, the architectural parameters are configured to achieve an operation time of approximately 50 ms.

Fig. 8 shows the results for different power-off voltages. The bars on the left illustrate the energy breakdown in a single power cycle, and the bars on the right represent the ratio of the sub-normal voltage executions. The most noticeable difference is ratio of energy consumed during the *Ramp-up & Init* stage. While A1 consumes 63.4% power at this stage on average, only 5.6% of energy is consumed in A2. This is because A1 is configured with an external MRAM, which exhibits significantly higher leakage current, even compared to the FRAM used in the reference system. In contrast, A2 is equipped with on-chip FRAM, which has much lower leakage.

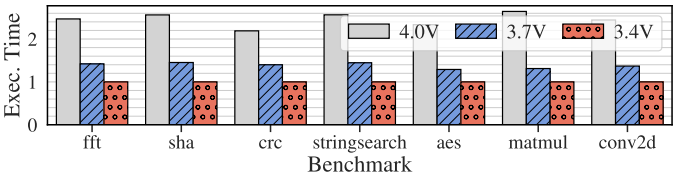


Fig. 9: Execution times across various checkpoint voltages, normalized to the 3.4V configuration.

Despite these differences, both architectures exhibit high sub-normal voltage execution rates, up to 55.5% in A1 and 70.1% in A2. In addition, discharged energy takes considerable portion in both A1 (31.4%) and A2 (52.0%) at 3.3V power-off voltage configuration, which represents the techniques based on the traditional model that halt immediately at V_{ES}. In summary, the evaluation demonstrates that the modeled buffering effects are general and their impacts are significant across different system architectures.

III. Design Guidelines

Based on the insights from our model, we propose design guidelines to implement efficient intermittent systems. The effectiveness of these guidelines is evaluated using seven benchmarks on the reference system used in Sec. II. We ported five benchmarks from miBench [45] benchmark suite and implemented two computation kernels (*matmul* and *conv2d*) commonly used in the evaluation of intermittent systems in the literature [11], [23], [24], [46], [47].

We evaluate two popular existing checkpointing schemes: *static* and *dynamic*. In *static*, checkpoint triggers are inserted at the back-edge of every loop in the program during compilation [3], [24], [25], [48]. At runtime, checkpoint triggers examine V_{ES} and execute checkpoint only when it is below a predefined threshold. In contrast, *dynamic* [4], [11], [20], [38], [39] does not modify the original program code. Instead, it executes checkpoints via interrupts from the power management system, generated when V_{ES} reaches V_l. These schemes are considered since most checkpoint techniques utilize V_{ES} either by actively polling it (as in *static*) or by receiving external signals generated based on V_{ES} levels (as in *dynamic*). All the evaluations are conducted with 470uF energy storage and 1mA of input current at 1.9V, unless otherwise stated.

A. Delaying Checkpoint Executions

The first design practice we propose is to delay checkpoint executions until the last possible moment. While this practice is generally regarded as desirable in existing works [3], [49], it has not been recognized as a critical property. Under the traditional execution model, early checkpoint execution is often considered acceptable as it makes the system wake up sooner, incurring only minor costs for initialization and recovery. For example, some approaches have explored proactive power-offs based on the program's worst-case execution time [1], [16], [50]. In contrast, our model reveals that significant energy is wasted each time the system powers off (Sec. II-C).

We evaluate the impact of delaying checkpoint executions in *dynamic*, by varying the interrupt voltage. A 1100uF capacitor is used for C_{ES}. Fig. 9 presents the average execution times of the benchmarks over 30 runs, normalized to the 3.4V configuration. The results show that executing checkpoints earlier is significantly inefficient as opposed to existing expectations: by 1.38x in 3.7V, and 2.45x in 4.0V setups, on average. Moreover, the overhead is consistent across all benchmarks since early checkpoint executions directly reduce the energy available for the computing system. Consequently, for maximum power efficiency, checkpoint techniques should be able to minimize the margin between the checkpoint execution and the power-off. Achieving this fundamentally depends on accurately predicting imminent power failures, which is the focus of the next section.

B. Using V_{dd} with a Reference Voltage for Checkpoint Signals

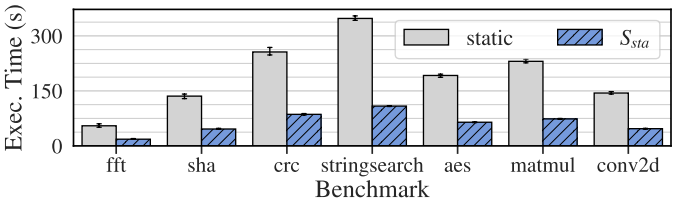
Sec. II-D demonstrates that V_{ES} is not a reliable estimate for the system's remaining execution time and that low V_{dd} is the direct cause of power-off. Based on this insight, we propose using V_{dd} to more accurately detect the imminent power failures, as in works without power management system (Sec. IV). We present two efficient implementations, S_{sta} and S_{dyn}, to accurately detect the imminent power-off events in approaches similar to *static* and *dynamic*, respectively.

Meanwhile, when designing techniques using V_{dd}, designers should account for the behaviors of analog components at sub-normal voltages (Sec. II-E). For consistent operation of ADCs, we adopt a voltage source with a known value of V_{ref}. In STM32L5 and MSP430, an internal reference voltage source of 1.2V is available; alternatively, an external voltage reference (e.g., TI LVM431 [51]) can be used. Note that V_{ref} should be lower than the minimal operating voltage of MCU (e.g., 1.7V) as V_{ref} is generated by regulating V_{dd}.

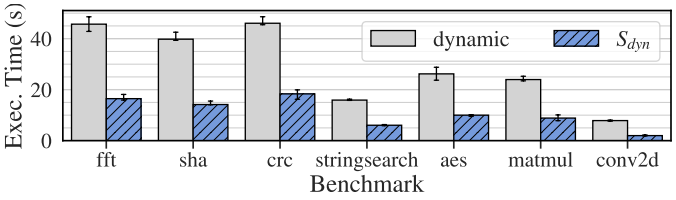
S_{sta} is designed for techniques similar to *static*, which query whether to execute a checkpoint at checkpoint triggers. Since directly reading V_{dd} is infeasible (i.e., V_{dd} itself is a reference voltage), S_{sta} reads V_{ref} instead. This results in the same value of $\lfloor V_{ref}/V_{dd} \cdot 2^n \rfloor$ when operating on normal voltage, where n is the ADC resolution. On the other hand, during sub-normal voltage executions, this value increases as V_{dd} decreases, as discussed in Sec. II-E. As a result, given that the target threshold voltage for checkpoint execution is V_{th}, software designers can compare the ADC value against $\lfloor V_{ref}/V_{th} \cdot 2^n \rfloor$ to determine whether to execute a checkpoint.

On the other hand, S_{dyn} utilizes an on-chip comparator, which is available in most modern MCUs including STM32L5 and MSP430. As V_{ref} is always lower than V_{dd}, we use a voltage divider consisting of two resistors, $R1$ and $R2$, to scale V_{dd} and compare it with V_{ref}. Specifically, we configure $R1$ and $R2$ to satisfy $\frac{R2}{R1+R2} \cdot V_{th} = V_{ref}$, so the comparator generates an interrupt when V_{dd} reaches the threshold V_{th}.

Fig. 10 compares the average execution times of the benchmarks over 30 iterations between traditional systems and the proposed setups. Fig. 10a and Fig. 10b illustrates the results of S_{sta} and S_{dyn}, respectively. The whiskers indicate the minimum and maximum execution times for each benchmark. The results show significant improvements in execution times for both systems, with average gain of 3.04x in S_{sta} and 2.85x



(a) Static checkpointing with S_{sta}.



(b) Dynamic checkpointing with S_{dyn}.

Fig. 10: Impact of precise checkpoint timings to the end-to-end execution times.

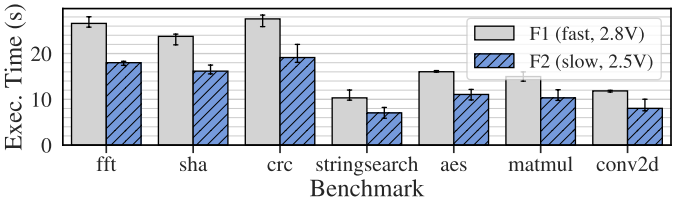


Fig. 11: Impact of peripheral operating voltage.

in S_{dyn}. While the effectiveness of checkpoint schemes varies depending on application characteristics, our setups evenly enhance performance across all benchmarks. This underscores the importance of accurately detecting power-off events for efficient intermittent system operation.

Another advantage of the proposed setups is their simplicity and practical applicability. Since the both setups only modify the method to detect imminent power failures and leave the checkpoint algorithms unchanged, it is straightforward to apply them in existing techniques. Furthermore, the proposed setups can reduce the system complexity, as they eliminate the need for communication between the energy storage system and the computing system (e.g., interrupt or access to V_{ES}).

C. On Selecting Hardware Components

Our model also helps designers in selecting efficient hardware components across various parameters. For example, it reveals that operating voltage of peripherals (e.g., external NVMs) is a critical design consideration (Sec. II-E), often more important than other factors such as latency.

To evaluate this tradeoff, we simulate two FRAM configurations, F1 and F2, in our reference system. F1 represents a slower setup capable of operating down to 2.5V. This is achieved by doubling the software-configurable wait time for FRAM accesses. F2 is set to have the lowest access latency but requires the system stop operating at 2.8V.

Fig. 11 presents the execution times of the benchmarks for the two configurations in S_{dyn}, averaged over 30 runs. Despite its doubled latency, F1 completes the workloads 1.46x faster on average, with consistent improvements across all benchmarks. These results imply that using slower FRAM that operates until 1.8V (e.g., [52]) could considerably improve the performance of our reference system. This example highlights that operating voltage, often overlooked in the traditional model, should be considered a critical design parameter.

Finally, our model highlights advantages of using smaller decoupling capacitors. Larger buffers not only increases the ratio of sub-normal voltage operations but also raise the amount of discharged energy during power-offs. Indeed, in our reference system with C_{ES} = 1100uF, we observe that completing benchmarks takes 1.18x and 1.36x longer on average, when 440uF and 660uF capacitors are used as C2, respectively, compared to our setup with a 220uF capacitor.

IV. Related Work

This work can be compared to existing modeling-based approaches that estimate the timing or efficiency of intermittent systems [7]–[10], [22], [53]. The primary focus of these works is identifying the most efficient design configurations (e.g., capacitor size, input power or checkpoint techniques [22]) for a given application. Zhan et al. [54] especially examined the trade-offs between capacitor sizes and forward progress. However, these works assume that the entire energy discharged from the capacitor is utilized by computing system, overlooking the buffering effects addressed in this work. Furthermore, our work proposes several practical guidelines to improve the efficiency of existing techniques with minimal efforts.

In some works that do not have a dedicated power management system and directly supply unregulated power to the computing system [13], [38], [39], [55], [56], V_{dd} has been used as a checkpoint signal. This is natural in these works since the voltage of the energy storage is always identical to V_{dd}. In contrast, our work demonstrates that accounting for sub-normal voltage operations is also critical in systems with regulated power supplies, which represent the majority of intermittent system setups. Also, we address the impacts of sub-normal voltage executions on the correctness and efficiency of software designs, along with suggestions to exploit such impacts for improved system performance.

V. Conclusion

As recent intermittent systems target smaller energy storages and shorter operation times, the traditional execution model for intermittent systems is failing to accurately represent actual system behaviors. In this paper, we propose a new execution model that incorporates the buffering effects from the system's inherent capacitance, which is a primary source of the discrepancies of the traditional model. Our model reveals that systems designed upon the traditional model can be up to 5.62x power-inefficient than expected and may result in unsafe checkpoint executions. Based on the insights from our model, we propose several design guidelines, including setups to improve performance of existing static and dynamic checkpoint techniques by 3.04x and 2.85x on average, respectively.

References

[1] J. Choi *et al.*, "Compiler-Directed High-Performance Intermittent Computation with Power Failure Immunity," RTAS '22, May 2022.

[2] S. Ahmed *et al.*, "The Internet of Batteryless Things," *Communications of the ACM*, vol. 67, pp. 64–73, Mar. 2024.

[3] B. Ransford *et al.*, "Mementos: System support for long-running computation on RFID-scale devices," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 159–170, Mar. 2011.

[4] H. Jayakumar *et al.*, "QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers," VLSID '14', pp. 330–335, Jan. 2014.

[5] K. Maeng and B. Lucia, "Adaptive low-overhead scheduling for periodic and reactive intermittent execution," PLDI '20', ACM, June 2020.

[6] J. de Winkel *et al.*, "Intermittently-powered bluetooth that works," MobiSys '22, pp. 287–301, ACM, June 2022.

[7] X. Hou *et al.*, "A Tale of Two Domains: Exploring Efficient Architecture Design for Truly Autonomous Things," ISCA '24, June 2024.

[8] F. Erata *et al.*, "ETAP: Energy-aware Timing Analysis of Intermittent Programs," *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 23:1–23:31, Jan. 2023.

[9] F. Ghasemi *et al.*, "PES: An Energy and Throughput Model for Energy Harvesting IoT Systems," ISPASS '23, pp. 13–23, Apr. 2023.

[10] J. San Miguel *et al.*, "The EH Model: Early Design Space Exploration of Intermittent Processor Architectures," MICRO '18, Oct. 2018.

[11] K. Maeng and B. Lucia, "Supporting peripherals in intermittent systems with just-in-time checkpoints," PLDI '19, ACM, June 2019.

[12] V. Kortbeek *et al.*, "BFree: Enabling Battery-free Sensor Prototyping with Python," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, pp. 135:1–135:39, Dec. 2020.

[13] A. J. Neto *et al.*, "DiCA: A Hardware-Software Co-Design for Differential Check-Pointing in Intermittently Powered Devices," ICCAD '23, pp. 1–9, Oct. 2023.

[14] A. Bakar *et al.*, "Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-Free Computing," SenSys '23, pp. 207–221, ACM, Jan. 2023.

[15] A. Alsubhi *et al.*, "Stash: Flexible Energy Storage for Intermittent Sensors," *ACM Trans. Embed. Comput. Syst.*, vol. 23, pp. 18:1–18:23, Mar. 2024.

[16] H. Reymond *et al.*, "SCHEMATIC: Compile-Time Checkpoint Placement and Memory Allocation for Intermittent Systems," CGO '24, pp. 258–269, Mar. 2024.

[17] Y. Wu *et al.*, "IntOS: Persistent Embedded Operating System and Language Support for Multi-threaded Intermittent Computing," OSDI '24, pp. 425–443, 2024.

[18] E. Yildiz *et al.*, "Efficient and Safe I/O Operations for Intermittent Systems," in *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 63–78, ACM, May 2023.

[19] V. Kortbeek *et al.*, "WARio: Efficient Code Generation for Intermittent Computing," p. 15, 2022.

[20] V. Kortbeek *et al.*, "Time-sensitive Intermittent Computing Meets Legacy Software," ASPLOS '20, pp. 85–99, ACM, Mar. 2020.

[21] H.-X. Shih *et al.*, "Intermittent Edge Computing for Green Agricultural Automation," IoTDI '24, pp. 219–220, May 2024.

[22] Y. Kim and H. Kim, "Rapid Hardware/Software Design Space Exploration for Efficient Intermittent Systems," ISLPED '24, pp. 1–6, ACM, Sept. 2024.

[23] K. Akhunov *et al.*, "Enabling Efficient Intermittent Computing on Brand New Microcontrollers via Tracking Programmable Voltage Thresholds," ENSsys '23, pp. 16–22, ACM, Nov. 2023.

[24] Y. Kim *et al.*, "LACT: Liveness-Aware Checkpointing to reduce checkpoint overheads in intermittent systems," *Journal of Systems Architecture*, vol. 153, p. 103213, Aug. 2024.

[25] Y. Kim *et al.*, "Liveness-Aware Checkpointing of Arrays for Efficient Intermittent Computing," DATE '23, pp. 1–6, Apr. 2023.

[26] G. Park *et al.*, "Energy-Harvesting-Aware Adaptive Inference of Deep Neural Networks in Embedded Systems," ISLPED '23, pp. 1–6, Aug. 2023.

[27] O. Khan *et al.*, "DaCapo: An On-Device Learning Scheme for Memory-Constrained Embedded Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 22, pp. 142:1–142:23, Sept. 2023.

[28] R. Barjami *et al.*, "Intermittent Inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup," SenSys '24, pp. 647–660, ACM, Nov. 2024.

[29] W. Song *et al.*, "TaDA: Task Decoupling Architecture for the Batteryless Internet of Things," SenSys '24, pp. 409–421, ACM, Nov. 2024.

[30] C.-H. Yen *et al.*, "Keep in Balance: Runtime-reconfigurable Intermittent Deep Inference," *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 124:1–124:25, Sept. 2023.

[31] G. Gobieski *et al.*, "Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems," ASPLOS '19, pp. 199–213, ACM, Apr. 2019.

[32] S. Islam *et al.*, "Enabling Fast Deep Learning on Tiny Energy-Harvesting IoT Devices," DATE '22, pp. 921–926, Mar. 2022.

[33] C.-K. Kang *et al.*, "More Is Less: Model Augmentation for Intermittent Deep Inference," *ACM Trans. Embed. Comput. Syst.*, vol. 21, pp. 49:1–49:26, Oct. 2022.

[34] S. Lee and S. Nirjon, "Neuro.ZERO: A zero-energy neural network accelerator for embedded sensing and inference systems," SenSys '19, pp. 138–152, ACM, Nov. 2019.

[35] B. Islam and S. Nirjon, "Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems," vol. 4, pp. 82:1–82:29, Sept. 2020.

[36] L. L. Custode *et al.*, "Fast-Inf: Ultra-Fast Embedded Intelligence on the Batteryless Edge," SenSys '24, pp. 239–252, ACM, Nov. 2024.

[37] L. Caronti *et al.*, "Fine-grained Hardware Acceleration for Efficient Batteryless Intermittent Inference on the Edge," *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 82:1–82:19, Sept. 2023.

[38] D. Balsamo *et al.*, "Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1968–1980, 2016.

[39] D. Balsamo *et al.*, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *IEEE Embedded Systems Letters*, vol. 7, pp. 15–18, Mar. 2015.

[40] E. Yildiz *et al.*, "Adaptable Runtime Monitoring for Intermittent Systems," in *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 1175–1191, ACM, Apr. 2024.

[41] T. Dang *et al.*, "ioTree: A battery-free wearable system with biocompatible sensors for continuous tree health monitoring," MobiCom '22, pp. 352–366, ACM, Oct. 2022.

[42] M. Afanasov *et al.*, "Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus," SenSys '20, pp. 368–381, ACM, Nov. 2020.

[43] M. Katanbaf *et al.*, "MultiScatter: Multistatic Backscatter Networking for Battery-Free Sensors," SenSys '21, pp. 69–83, ACM, Nov. 2021.

[44] S. Babatunde *et al.*, "Greentooth: Robust and Energy Efficient Wireless Networking for Batteryless Devices," *ACM Transactions on Sensor Networks*, p. 3649221, Mar. 2024.

[45] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," WWC '01, pp. 3–14, Dec. 2001.

[46] A. Bhattacharyya *et al.*, "NvMR: Non-volatile memory renaming for intermittent computing," ISCA '22, pp. 1–13, ACM, June 2022.

[47] K. Ganesan *et al.*, "The What's Next Intermittent Computing Architecture," HPCA '19, pp. 211–223, Feb. 2019.

[48] K. Maeng and B. Lucia, "Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing," OSDI '18, pp. 129–144, 2018.

[49] N. A. Bhatti and L. Mottola, "HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing," IPSN '17, pp. 209–220, Apr. 2017.

[50] P. Rafferk *et al.*, "WoCA: Avoiding Intermittent Execution in Embedded Systems by Worst-Case Analyses with Device States," LCTES '24, pp. 83–94, ACM, June 2024.

[51] Texas Instruments, "LMV431, 1.5%, low-voltage (1.24-V) adjustable precision shunt regulator." https://www.ti.com/product/en-us/LMV431.

[52] Fujitsu Semiconductor, *MB85R4M2T*.

[53] J. San Miguel *et al.*, "The EH Model: Analytical Exploration of Energy-Harvesting Architectures," *IEEE Computer Architecture Letters*, vol. 17, pp. 76–79, Jan. 2018.

[54] J. Zhan *et al.*, "Exploring the Effect of Energy Storage Sizing on Intermittent Computing System Performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, pp. 492–501, Mar. 2022.

[55] P. Rafferk *et al.*, "CO2CoDe: Towards Carbon-Aware Hardware/Software Co-Design for Intermittently-Powered Embedded Systems," 2024.

[56] H. Reymond *et al.*, "EarlyBird: Energy belongs to those who wake up early," RTCSA '24, 2024.

Me-MPK: Accelerating Krylov Subspace Solvers via Memory-efficient Matrix-Power Kernel

Haozhong Qiu
College of Computer Science and Technology, National University of Defense Technology
Changsha, China
qiuhaozhong16@nudt.edu.cn

Chuanfu Xu, Jianbin Fang, Shengguo Li
College of Computer Science and Technology, National University of Defense Technology
Changsha, China
{xuchuanfu, j.fang, nudtlsg}@nudt.edu.cn

Liang Deng, Jian Zhang, Zhe Dai
China Aerodynamic Research and Development Center
Mianyang, China
dengliang11@nudt.edu.cn, zhangjian@cardc.cn
daizhe_cardc@163.com

Yue Ding, Yue Wang, Zhimeng Han
College of Computer Science and Technology
National University of Defense Technology
Changsha, China
{dingyue, wangyue, hanzhimeng24}@nudt.edu.cn

Yonggang Che, Jie Liu
College of Computer Science and Technology
National University of Defense Technology
Changsha, China
{ygche, liujie}@nudt.edu.cn

***Abstract*—This paper focuses on optimizing the Matrix-Power Kernel (MPK), which relies on a series of Sparse Matrix-Vector multiplications (SpMVs) using the same sparse matrix. MPK is a crucial component of Krylov subspace methods for solving large sparse linear systems in various fields, including circuit simulations. MPK offers a potential for matrix reuse in cache, which can accelerate memory-bound sparse solvers. Additionally, many sparse matrices encountered in applications are symmetric, allowing us to reduce the memory footprint for SpMVs by half. However, reusing the matrix introduces data dependencies between subsequent SpMVs, and symmetric SpMVs can result in data conflicts during shared-memory parallelization. Previous research has often focused on either matrix reuse or symmetry, failing to leverage both aspects effectively.**

This paper proposes a unified, memory-efficient approach called *Me-MPK* that takes advantage of both cache reuse and matrix symmetry for MPK on shared-memory multi-core systems. We first introduce a *unified dependency graph* for a sparse matrix, which represents all potential data dependencies and conflicts. Next, we perform architecture-aware recursive partitioning on this graph to create subgraphs and formulate a separating subgraph that decouples all dependencies and conflicts among the subgraphs. These independent subgraphs are then scheduled for parallel execution of SpMV or symmetric SpMV in a specified order to optimize cache reuse. We apply Me-MPK in two s-Step Krylov subspace solvers, and our evaluations show that Me-MPK significantly outperforms the current state-of-the-art solutions, delivering an average speedup of up to 2.00X and 1.86X on X86 and ARM CPUs, respectively. As a result, we achieve overall speedup in the sparse solvers of up to 1.65X and 1.58X.

***Index Terms*—SpMV, matrix power kernel, cache reuse, matrix symmetry, Krylov subspace solvers**

I. Introduction and Related Work

The Matrix-Power Kernel (MPK) involves a sequence of Sparse Matrix-Vector multiplications (SpMVs) to compute $\mathbf{Ax}$, $\mathbf{A^2x}$,..., $\mathbf{A^sx}$ for a matrix $\mathbf{A}$, a vector $\mathbf{x}$, and a constant s [1]–[3]. MPK is a crucial and time-consuming step in solving large-scale sparse linear systems, eigenvalue problems, and similar tasks [4]. Optimizing MPK is vital to enhance simulation efficiency across various fields, including circuit simulation and power grid analysis.

Given that SpMV is heavily constrained by memory, numerous efforts have been made to optimize memory efficiency for an individual SpMV invocation [5]–[8]. MPK offers a potential to exploit temporal locality for a sequence of SpMVs by reusing the matrix elements in cache, instead of repeatedly loading them from main memory. Meanwhile, the sparse matrices in many scientific applications are often symmetric [9]–[11]. Exploiting symmetric SpMV can significantly reduce memory usage by only storing either the upper or lower triangular part of the sparse matrix. Therefore, by taking advantage of both matrix reuse and symmetry, there is a significant opportunity to develop memory-efficient MPK, which can help mitigate the 'memory wall' problem that often challenges sparse solvers.

However, reusing the matrix in cache creates data dependencies between successive SpMV invocations. In the case of symmetric SpMV, there are data conflicts when multiple threads attempt to update the output vector simultaneously. These data dependencies and conflicts complicate the efficient and scalable implementation of multi-threaded processing on shared-memory multi-core systems.

Previous implementations of MPK have primarily focused on reusing matrices to minimize the frequency of distributed communications and synchronizations for communication-avoiding or s-Step algorithms [12]–[15]. The motivation behind exploiting cache reuse in MPK is largely rooted in classic blocking strategies used in stencil computations [16]–[19]. These methods typically involve identifying the neighboring blocks necessary for MPK concerning a specific matrix power s. This often leads to performance bottlenecks due to either

This work was funded by the National Natural Science Foundation of China under Grant No. 62272474. Co-first author: Chuanfu Xu. Corresponding authors: Chuanfu Xu, Jianbin Fang, Shengguo Li and Yonggang Che

redundant computations or irregular matrix accesses. A recent level-based cache blocking approach for MPK [2], based on the Recursive Algebraic Coloring Engine (RACE) [10], organizes matrix rows into levels using a distance-k coloring. This method maintains dependencies for cache reuse but limits parallelism, allowing only one level to be executed in parallel at a time. Therefore, it relies on point-to-point synchronization and level coarsening strategies to improve scalability, but its performance still degrades severely for large power. Zhang *et al.* partition a sparse matrix and create a computation pipeline for fine-grained cache reuse by reusing a matrix row rather than a row block in MPK [1]. Although the approach is easy to implement, it requires graph coloring for parallelization, which can result in high synchronization overhead and poor spatial locality [20]. To summarize, all MPK implementations overlook the symmetry of the matrix. Thus, our work aims to leverage both cache reuse and matrix symmetry to further enhance MPK performance.

There has been some work in implementing symmetric SpMV, primarily focused on efficiently managing data conflicts at the shared-memory level. For instance, a reduction-based approach [21], [22] mitigates conflicts by using thread-local vectors to store temporary results, followed by a reduction loop to accumulate these results into the output vector. However, this method increases memory usage and traffic, which grows linearly with the number of threads, thereby limiting its overall scalability. Recently, researchers have introduced graph coloring algorithms to identify independent row sets from the graph representation of the symmetric matrix for parallel execution [9], [10]. While coloring-based approaches demonstrate impressive performance for low-bandwidth symmetric matrices, they struggle to scale effectively with large, irregular, high-bandwidth matrices. Furthermore, existing implementations of symmetric SpMV typically focus on a single invocation and do not leverage cache reuse for MPK. In summary, prior work has not fully exploited on the advantages of both matrix reuse and matrix symmetry; they tend to address only one aspect or the other.

This paper optimizes MPK by introducing a new, unified and memory-efficient approach called *Me-MPK*, which takes advantage of both cache reuse and matrix symmetry for MPK on multi-cores. We present the *unified dependency graph* for a sparse matrix, where each vertex represents a row of the matrix, and each edge denotes potential data dependencies or conflicts between rows. We then apply architecture-aware recursive partitioning to this graph to create subgraphs. In each recursion, we formulate a separate subgraph, known as the *separator*, to eliminate all dependencies and conflicts among the subgraphs. All subgraphs are then reordered to enhance data locality. Finally, we schedule these independent subgraphs in a specific order for parallel execution, ensuring that the same subgraph can be reused in the cache at least twice for subsequent calls to SpMV or symmetric SpMV.

We apply Me-MPK in two *s*-Step Krylov subspace methods, i.e., the Conjugate Gradient (CG) method and the Biconjugate Gradient Stabilized (BiCGStab). Our work was evaluated on a 28-core Intel Xeon X86 CPU and a 32-core Kunpeng 920 ARMv8 CPU [23] with 214 sparse matrices from the SuiteSparse Matrix Collection [24]. We compare Me-MPK to the MPKs based on SpMV and symmetric SpMV from the vendor-tuned Intel oneMKL library [25] and the RACE approach [2]. On X86, Me-MPK outperforms the MPKs based on the oneMKL SpMV and symmetric SpMV by an average speedup of 1.95X and 1.56X for *s*=4, 2.00X and 1.59X for *s*=8. Compared to the RACE-based MPK, Me-MPK yields an average speedup of 1.32X and 1.20X for *s*=4, 1.42X and 1.26X for *s*=8 on X86 and ARM. As a consequence, Me-MPK accelerates the sparse solvers by an average speedup of up to 1.59X and 1.51X for *s*=4, 1.65X and 1.58X for *s*=8 on X86 and ARM, respectively.

II. Background

This section first introduces the background of SpMV and symmetric SpMV. Then we introduce the Matrix-Power Kernel (MPK) and its application in *s*-Step Krylov subspace solvers.

A. SpMV and Symmetric SpMV

The SpMV operation involves multiplying a sparse matrix $\mathbf{A}$ by a dense input vector $\mathbf{x}$ to produce a dense output vector $\mathbf{b}$. Sparse matrices are typically stored in a compressed format to conserve space, which involves only retaining their nonzero elements and their corresponding locations. The top of Fig.1(b) illustrates the popular Compressed Sparse Row (CSR) format for a symmetric sparse matrix $\mathbf{A}$, as shown in Fig.1(a). The CSR format uses two arrays: one for storing the nonzero values and another for their corresponding column indices. Additionally, a row pointer array indicates the number of nonzero elements in each row. Algorithm 1 (lines 1-8) presents the pseudo-code for the serial SpMV in CSR.

The most common storage format for symmetric sparse matrices is the Sparse Symmetric Skyline (SSS) format [4], which is a variant of CSR with an additional array to store the matrix's main diagonal. The bottom of Fig.1(b) shows the SSS format storing the upper triangular part of the matrix $\mathbf{A}$. Algorithm 1 (lines 9-18) shows the pseudo-code of serial symmetric SpMV in SSS. Shared-memory parallelization of the outer loop (line 10) will introduce race conditions due to concurrently writing the output vector $\mathbf{b}$ ($b[col]$ in line 15 and $b[i]$ in line 17). Fig.1(d) illustrates this problem for the symmetric matrix $\mathbf{A}$ using two threads. In this example, thread 0 not only updates entries b_0 to b_3 but also affects b_4 to b_6, leading to write conflicts between the two threads.

B. MPK and Its Application in Krylov Subspace Solvers

MPK calculates a sequence of SpMVs to obtain $\mathbf{Ax}$, $\mathbf{A^2x}$,... $\mathbf{A^sx}$ or their linear combination such as $\mathbf{Ax} + \mathbf{A^2x}$, for the same matrix $\mathbf{A}$. Algorithm 1 (lines 19-22) shows the pseudo-code for a traditional MPK implementation, which performs s consecutive calls to the SpMV function. If the caches are too small to hold the entire matrix, it will need to be loaded s times from the main memory.

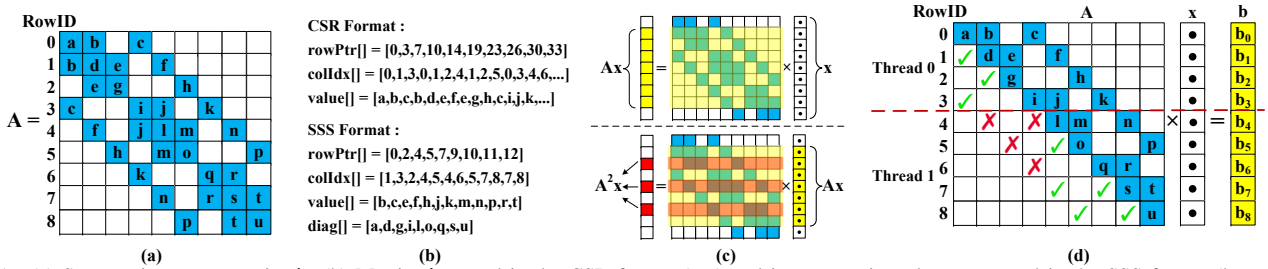


Fig. 1. (a) Symmetric sparse matrix $\mathbf{A}$; (b) Matrix $\mathbf{A}$ stored in the CSR format (top) and its upper triangular part stored in the SSS format (bottom); (c) Data reuse between $\mathbf{Ax}$ and $\mathbf{A^2x}$; (d) Data conflicts (red $\times$) between two threads using the upper triangular part of $\mathbf{A}$ in symmetric SpMV.

Algorithm 1: SpMV, symmetric SpMV and MPK

```
1  function SpMV(A, x, b)
2  for i ← 0 to N do
3      sum ← 0
4      for j ← A.rowPtr[i] to A.rowPtr[i + 1] do
5          sum = sum + A.value[j] · x[A.colIdx[j]]
6      end
7      b[i] = b[i] + sum
8  end
9  function SymmSpMV(A, x, b)
10 for i ← 0 to N do
11     sum ← A.diag[i] · x[i]
12     for j ← A.rowPtr[i] to A.rowPtr[i + 1] do
13         col = A.colIdx[j]
14         sum = sum + A.value[j] · x[col]
15         b[col] = b[col] + A.value[j] · x[i]
16     end
17     b[i] = b[i] + sum
18 end
19 function MPK(A, x, s)
20 for i ← 0 to s do
21     SpMV(A, x[:,i], x[:,i+1])
22 end
```

Algorithm 2: s-Step conjugate gradient method

```
1  Let A ∈ ℝ^{n×n} be a SPD matrix and x_0 be an initial guess.
2  Compute r_0 = b − Ax_0 and let index k = is (initially k = 0)
3  for i = 0, 1, ... until convergence do
4      Compute T = [r_k, Ar_k, ..., A^s r_k]          ▷ Matrix-power kernel
5      Let R_i = [r_k, Ar_k, ..., A^{s−1} r_k]        ▷ Extract R from T
6      Let P_i = [Ar_k, A^2 r_k, ..., A^{s−1} r_k]    ▷ Extract P from T
7      if i == 0 then                                 ▷ Find directions p
8          Set P_i = R_i
9      else
10         Compute C_i = −Q_i^T P_{i−1}               ▷ Block dot-products
11         Solve W_{i−1} B_i = C_i                    ▷ Find scalars β
12         Compute P_i = R_i + P_i B_i
13     end
14     Compute W_i = Q_i^T P_i                        ▷ Block dot-products
15     Compute g_i = P_i^T r_i
16     Solve W_i a_i = g_i                            ▷ Find scalars α
17     Compute x_{k+s} = x_k + P_i a_i                ▷ Compute new x_k
18     Compute r_{k+s} = b − Ax_{k+s}                 ▷ Recompute residual r_k
19     if ||r_{k+s}||_2 / ||r_0||_2 ≤ tol then stop end   ▷ Check convergence
20 end
```

MPK is a crucial component in many applications, such as Krylov subspace methods, multigrid methods, polynomial preconditioners, and eigenvalue problems. This work applies our MPK within two s-Step (or communication-avoiding) Krylov subspace methods: the Conjugate Gradient (CG) method for solving Symmetric Positive Definite (SPD) linear systems, and the Biconjugate Gradient Stabilized (BiCGStab) method for addressing non-SPD linear systems. Algorithm 2 provides the pseudo-code for the s-Step CG method [26][1]. These Krylov subspace solvers are widely used to solve large-scale sparse linear systems of transistor-level SPICE-like circuit simulations.

The s-step Krylov solvers generate s Krylov basis vectors at a time using MPK (line 4 in Algorithm 2), potentially reducing the communication cost of standard Krylov solvers by s. MPK also offers an opportunity to reuse the matrix elements in the cache. However, reusing cache data incurs dependencies between multiple SpMV invocations. Fig.1(c) illustrates this problem with $s = 2$. Suppose the submatrix (yellow rows) has been loaded into cache for $\mathbf{Ax}$, it is possible to reuse a subset (red rows) of this submatrix for $\mathbf{A^2x}$. However, this data reuse must fulfill the dependencies between successive SpMVs, i.e., the column indices of the subset must be in the range of the row indices of the original submatrix.

[1]For the pseudo-code of the s-Step BiCGStab method, please refer to [26]

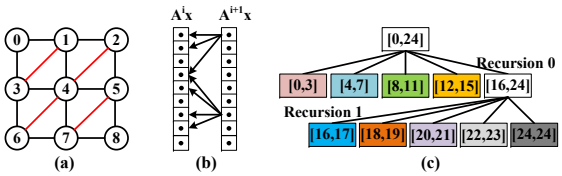


Fig. 2. (a) The unified dependency graph for the matrix in Fig.1(a); (b) Data dependencies between $\mathbf{A^ix}$ and $\mathbf{A^{i+1}x}$ for two vector elements; (c) The tree data structure corresponds to Fig.3(b). Numbers in the brackets indicate the starting and ending indices for matrix rows. [16,24] and [24,24] are the separators for the first and second recursions, respectively.

III. The Memory-efficient Matrix-Power Kernel

This section presents our unified approach to design memory-efficient MPK and its shared-memory parallelization.

A. Unified Dependency Graph

We propose *Unified Dependency Graph* (UDG), which is an undirected graph to represent all potential data dependencies and data conflicts in exploiting cache reuse and matrix symmetry for Me-MPK. Fig.2(a) shows the graph for the matrix $\mathbf{A}$ in Fig.1(a). Each vertex represents a matrix row. Each black edge indicates that a nonzero element of row i, which has a column index equal to j (or vice versa). These black edges correspond to all potential data dependencies among subsequent SpMV invocations. Fig.2(b) illustrates dependencies between two subsequent SpMVs for two vector elements: calculation of $\mathbf{A^{i+1}x}[0]$ and $\mathbf{A^{i+1}x}[6]$ depends on specific elements in $\mathbf{A^ix}$. When symmetric SpMV is employed in MPK, we introduce red edges in the graph, indicating that two rows have nonzero

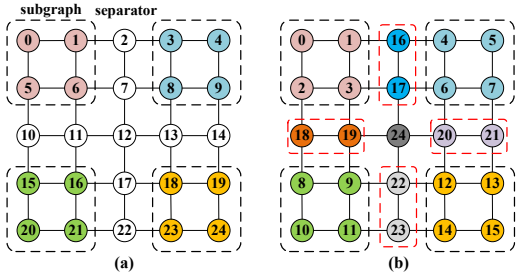


Fig. 3. (a) A graph partitioned into four subgraphs and one separator (without reordering); (b) After two recursive partitions (with reordering).

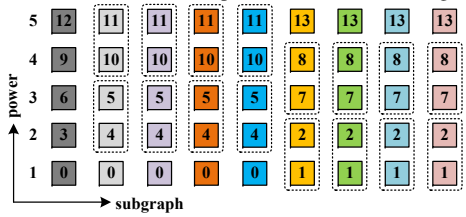


Fig. 4. Diagram illustrating the execution of Fig.3(b) in Me-MPK with the power *s=5*. Numbers in boxes indicate the execution order. Subgraphs with the same numbers are executed concurrently. Boxes in the same dotted rectangles indicate that subgraphs are reused in cache. For simplicity, this diagram assumes the sparse matrix is non-symmetric.

elements with the same column index. These black and red edges correspond to all potential data conflicts that may appear in a single invocation of shared-memory symmetric SpMV.

B. Architecture-aware Recursive Partitioning

For a given unified dependency graph of a sparse matrix, we propose an architecture-aware recursive approach to partition the graph. We start by partitioning the graph into *numPart* subgraphs. This can be implemented using existing graph partitioning tools like Metis [27]. For a CSR-based sparse matrix with 8-byte double values and 4-byte int indices, each subgraph contains no more than $partSize = \frac{cacheSize/8}{1.5*avg_nnz_per_row}$ rows, where $avg_nnz_per_row$ is the average number of nonzero elements per row, and *cacheSize* is the L2 cache size in bytes for the processor. This ensures that each subgraph can fit into the L2 cache. $numPart$ is set to $(\lfloor \frac{N/partSize}{numCore} \rfloor + 1) * numCore$, where N is the total number of matrix rows, allowing all subgraphs be evenly distributed among the *numCore* processor cores.

Then we formulate a special subgraph called *separator* by figuring out all vertices which have neighbors in other subgraphs. Next, we reorder vertices for each subgraph and the separator to ensure that each subgraph contains consecutive vertices, and the separator is indexed after the other subgraphs. This can improve the data locality when accessing the input/output vectors in SpMV. Fig.3 shows a graph after two recursive partitions (*numPart=4*). We use a tree pointer *treePtr* to save the result of recursive partitioning, used in the following parallel implementation. Fig.2(c) shows the tree data structure for Fig.3(b).

For invocations of standard SpMV in Me-MPK, we partition the separator one more time. For symmetric SpMV invocations, we recursively partition the separator multiple times. The optimal number of recursive partitions depends on the specific sparse symmetric matrices and the architectures used. This will be discussed further in the experiments.

Algorithm 3: Memory-efficient parallel MPK

```
1  function uSpMV(A, x, b, treePtr)
2  for i ← treePtr->st to treePtr->ed do
3      if A is symmetric then
4          line 11-17 in Algorithm 1
5      else
6          line 3-7 in Algorithm 1
7      end
8  end
9  function SymmSpMV_Recursive(A, x, b, treePtr)
10 if treePtr->separator == nullptr then
11     uSpMV(A, x, b, treePtr)
12     return
13 end
14 for i ← 0 to treePtr->numPatrs do in parallel
15     uSpMV(A, x, b, treePtr->son[i])
16 end
17 SymmSpMV_Recursive(A, x, b, treePtr->separator)
18 function puSpMV(A, x, b, treePtr)
19 if A is symmetric then
20     SymmSpMV_Recursive(A, x, b, treePtr)
21 else
22     for i ← treePtr->st to treePtr->ed do in parallel
23         line 3-7 in Algorithm 1
24     end
25 end
26 function Me_MPK(A, x, s, treeRoot)
27 if s==1 then
28     puSpMV(A, x[:,0], x[:,1], treeRoot)
29     return
30 end
31 i ← 0
32 separator ← treeRoot->separator
33 puSpMV(A, x[:,0], x[:,1], separator)
34 while i < s do
35     for j ← 0 to treeRoot->numParts do in parallel
36         uSpMV(A, x[:,i], x[:,i+1], treeRoot->son[j])
37         if i + 1 < s then
38             uSpMV(A, x[:,i+1], x[:,i+2], treeRoot->son[j])
39         else stop; end
40     end
41     i ← i+2
42     if i < s then
43         puSpMV(A, x[:,i-1], x[:,i], separator->separator)
44         for j ← 0 to separator->numParts do in parallel
45             uSpMV(A, x[:,i-1], x[:,i], separator->son[j])
46             uSpMV(A, x[:,i], x[:,i+1], separator->son[j])
47         end
48         puSpMV(A, x[:,i], x[:,i+1], separator->separator)
49     end
50     else
51         puSpMV(A, x[:,i-1], x[:,i], separator)
52     end
53 end
```

C. Parallel Implementation

After recursive partitioning of the graph for a sparse matrix, we obtain independent subgraphs (i.e., submatrices) at each recursion, because all edges (i.e., data dependencies and data conflicts) among subgraphs have been removed via separators. These submatrices, represented by the starting and ending row indices in the tree data structure, can be scheduled for parallel execution in a specified order to achieve cache reuse in MPK.

Algorithm 3 presents the pseudo-code of Me-MPK. To facilitate the implementation of Me-MPK for both symmetric and non-symmetric sparse matrices, the serial SpMV and symmetric SpMV for a submatrix (row index between `treePtr->st` and `treePtr->ed`) are encapsulated in a

TABLE I
DETAILED CONFIGURATIONS OF MULTI-CORE CPUS.

	Xeon Gold 6258R	Kunpeng 920
Core frequency	2.4 GHz	2.6 GHz
Core number	28	32
L1 cache (I/D)	32/32 KiB	64/64 KiB
L2 cache (per core)	1024 KiB	512 KiB
L3 cache	39 MiB	32 MiB
Main memory	128 GiB	128 GiB

unified kernel `uSpMV` (lines 1-8). Accordingly, the parallel version is encapsulated in another unified kernel `puSpMV` (lines 18-25). Note that for standard SpMV, shared-memory parallelization can be directly implemented at the row loop (line 22); for symmetric SpMV, we implement a recursive kernel `SymmSpMV_Recursive` (lines 9-17) to parallelize it at the subgraph loop (line 14). This can avoid data conflicts for shared-memory parallelization of symmetric SpMV.

Me-MPK is implemented using two kernels: `uSpMV` and `puSpMV`. Fig.4 shows the execution order for different subgraphs shown in Fig.3(b). Since each subgraph or submatrix can fit into the L2 cache and all data dependencies or conflicts have been addressed, we can immediately schedule SpMVs or symmetric SpMVs for subsequent powers to use the entire submatrix that has been loaded into L2 from the previous power. For example, the subgraphs marked with an execution order of 1 and those marked with an execution order of 2 are reused in L2, as indicated by the dotted rectangles. Fig.4 shows that most subgraphs, except the separator at the second recursion, can be reused successfully.

IV. PERFORMANCE EVALUATION

This section begins by introducing our experimental setup, which includes the hardware, software, and tested matrices. We then present and analyze the performance results.

A. Experiment Setup

Hardware and software. We perform experiments on two multi-core CPUs: one Intel Xeon Gold 6258R X86 CPU@2.4GHz and one HUAWEI Kunpeng 920 ARMv8 CPU@2.6GHz [23]. Table I shows the detailed configurations for each CPU. All codes are compiled in double precision with '-O3' using Intel oneAPI 2022.1 for the X86 architecture and GCC 8.4.1 for the ARM architecture, respectively.

We set thread affinity for all tests, binding each software thread to an individual physical core (without hyperthreading). We run different implementations 100 times for each matrix and measure the average GFLOPS.

Sparse matrices. We select 214 large matrices from the SuiteSparse Matrix Collection [24] for our tests, where each matrix has more than 100,000 rows and more than 200,000 non-zero elements. Out of these, 162 sparse matrices are numerically symmetric. Additionally, we chose 12 circuit matrices (Table II) for a detailed comparison of our approach with existing methods. For non-symmetric matrices, we perform two rounds of partitioning. For symmetric matrices, we profile each matrix across two CPUs and conduct no more than four rounds of partitioning for most of them. The preprocessing cost associated with recursive partitioning for a sparse matrix is generally less than the time taken for one solver iteration.

TABLE II
THE TESTED CIRCUIT MATRICES. SPD INDICATES WHETHER A MATRIX IS SYMMETRIC POSITIVE DEFINITE. N IS THE ROW NUMBER. NNZ IS THE NON-ZERO ELEMENT NUMBER.

ID	Circuit matrix	N	NNZ	SPD
1	ASIC_320k	321821	1931828	no
2	ASIC_320ks	321671	1316085	no
3	ASIC_680ks	682712	1693767	no
4	Freescale1	3428755	17052626	no
5	FullChip	2987012	26621983	no
6	G2_circuit	150102	726674	yes
7	G3_circuit	1585478	7660826	yes
8	memchip	2707524	13343948	no
9	nxp1	414604	2656177	no
10	pre2	659033	5834044	no
11	rajat30	643994	6175244	no
12	transient	178866	961790	no

TABLE III
AVERAGE SPEEDUPS OF VARIOUS MPKS USING MKL ON X86 AND CSR ON ARM AS BASELINES.

	Intel 6258			Kunpeng 920		
	s=1	*s=4*	*s=8*	*s=1*	*s=4*	*s=8*
MKL-symm	1.27	1.23	1.23	-	-	-
RACE	1.10	1.43	1.35	1.07	1.52	1.47
Me-MPK(*reuse*)	1.17	1.53	1.59	1.21	1.51	1.54
Me-MPK(*reuse+symm*)	1.69	1.95	2.00	1.60	1.84	1.87

B. Performance Results

1) Overall Performance: On the X86 CPU, we evaluate the following MPK implementations: ❶ MKL. This version calls the Intel oneMKL SpMV in MPK, without exploiting cache reuse and matrix symmetry; ❷ MKL-symm. This version calls the Intel oneMKL symmetric SpMV to exploit matrix symmetry in MPK; ❸ RACE. This version is implemented by [2], which exploits cache reuse in MPK using the Recursive Algebraic Coloring Engine (RACE) approach [10]; ❹ Me-MPK(*reuse*). This version calls the standard CSR-based SpMV, but implements our strategy of cache reuse in MPK; ❺ Me-MPK(*reuse+symm*). This version implements our strategy of cache reuse, and it also calls our symmetric SpMV to exploit matrix symmetry in MPK. On the ARM CPU, we compare our work with MPK based on the standard SpMV using CSR format (referred to as CSR) and MPK based on RACE (referred to as RACE).

Fig.5 and Fig.6 show the overall performance for different MPK versions on the two CPU processors. Table III lists the average speedups of various MPKs using MKL on X86 and CSR on ARM as baselines. Compared to other MPKs, our two MPK versions, i.e., Me-MPK(*reuse*) and Me-MPK(*reuse+symm*), achieve higher speedups on both CPU platforms. Among all MPKs, Me-MPK(*reuse+symm*) performs best: On X86, it improves the performance by 1.95X and 2.00X, 1.56X and 1.60X, 1.32X and 1.42X, compared to MPKs based on oneMKL SpMV, oneMKL symmetric SpMV and RACE, for *s*=4 and *s*=8; On ARM, it improves the performance by 1.84X and 1.87X, 1.20X and 1.26X, compared to MPKs based on CSR and RACE, for *s*=4 and *s*=8. Compared to Me-MPK(*reuse*), Me-MPK(*reuse+symm*) enhances the performance by 1.32X and 1.21X, 1.14X and 1.19X on X86 and ARM for *s*=4 and *s*=8. These results

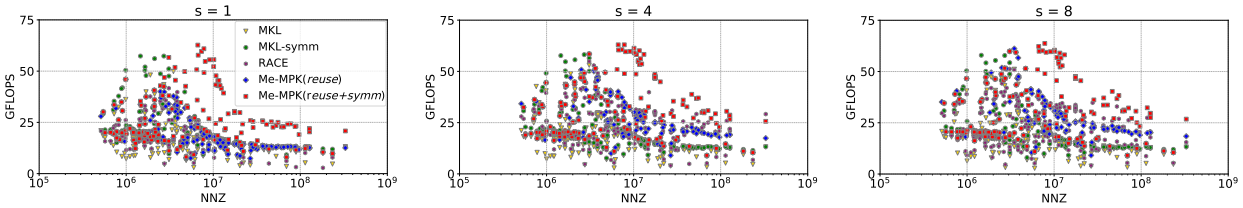


Fig. 5. Performance in GFLOPS for different MPK implementations using 214 sparse matrices on the Intel 6258R CPU. Note that MKL-symm and Me-MPK(*reuse+symm*) are evaluated using 162 symmetric matrices.

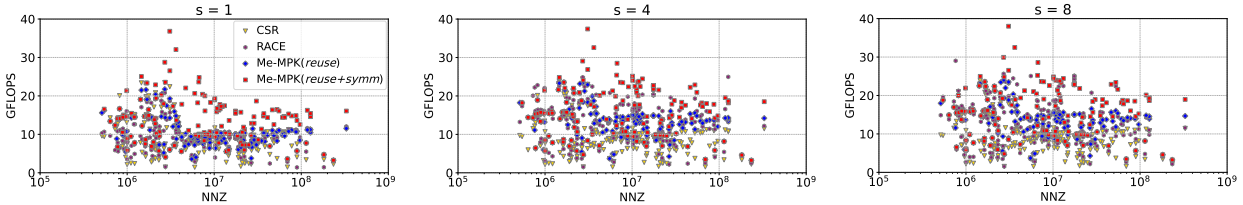


Fig. 6. Performance in GFLOPS for different MPK implementations using 214 sparse matrices on the Kunpeng 920 CPU. Note that Me-MPK(*reuse+symm*) is evaluated using 162 symmetric matrices.

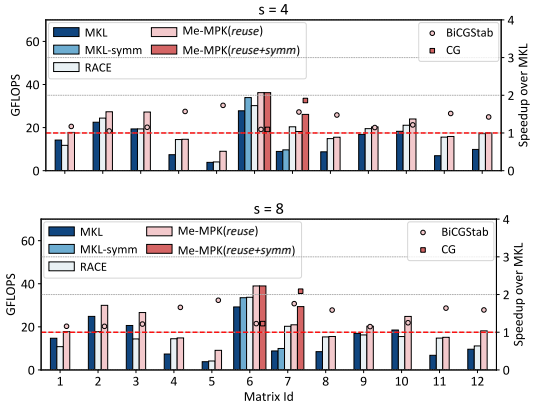


Fig. 7. Performance in GFLOPS for different MPK implementations using the 12 circuit matrices on the Intel 6258R CPU. The right Y-axis indicates the speedup for solvers compared to MKL-based baseline.

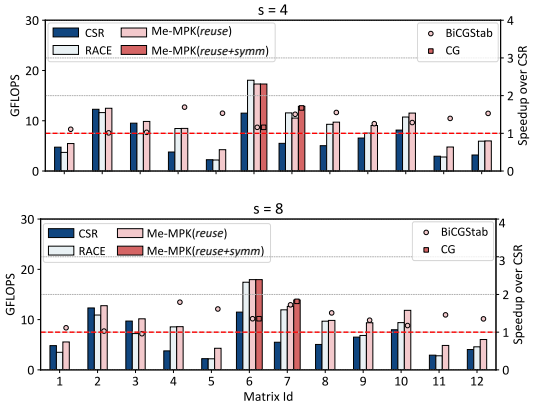


Fig. 8. Performance in GFLOPS for different MPK implementations using the 12 circuit matrices on the Kunpeng 920 CPU. The right Y-axis indicates the speedup for solvers compared to CSR-based baseline.

demonstrate the advantage of exploiting both matrix reuse and matrix symmetry in MPK. Note that the performance of RACE-based MPK drops for large power s. This is mainly because RACE reuses matrix elements in the L3 cache where s is the reuse distance, and increasing s will probably exceed the cache capacity, making it unable to reuse the data [2]. On the contrary, our approach always reuses data in the L2 cache for two successive SpMVs (i,e., $\mathbf{A^i x}$ and $\mathbf{A^{i+1}x}$), and the number of total cache reuse will increase with s because we reuse data (s-*1*) times for a given power s.

We also evaluate Me-MPK in two s-Step Krylov subspace solvers (i.e., CG and BiCGStab). Compared to MKL-based solvers on X86 and CSR-based solvers on ARM, Me-MPK accelerates the performance for a single solver iteration by an average speedup of 1.76X and 1.60X (s=4), 1.81X and 1.62X(s=8) for CG. The average speedup for BiCGStab is 1.34X and 1.33X (s=4), 1.40X and 1.38X (s=8) on X86 and ARM, respectively. Note that our approach fulfills all data dependencies and conflicts when exploiting matrix reuse and symmetry in MPK, therefore it has no negative effect on the solver convergence.

2) Performance on Circuit Matrices: Fig.7 and Fig.8 show the performance for the 12 circuit matrices. For 10 non-SPD matrices, Me-MPK(*reuse*) yields an average speedup of 1.57X and 1.61X, 1.18X and 1.38X, compared to MKL and RACE for s=4 and s=8 on X86; On ARM, the average speedup is 1.53X and 1.56X, 1.19X and 1.30X, compared to CSR and RACE for s=4 and s=8. For 2 SPD matrices, MPK(*reuse+symm*) delivers an average speedup of 1.95X and 2.11X, 1.70X and 1.85X, 1.24X and 1.30X, 1.17X and 1.18X, compared to MKL, MKL-symm, RACE and MPK(*reuse*)for s=4 and s=8 on X86; On ARM, the average speedup is 1.87X and 1.99X, 1.03X and 1.09X, 1.11X and 1.12X, compared to CSR, RACE and Me-MPK(*reuse*) for s=4 and s=8. Taking MKL-based solvers on X86 as baselines, we accelerate the s-Step CG and BiCGStab solvers by 1.63X and 1.40X (s=4), 1.65X and 1.41X (s=8) for circuit simulations. Taking CSR-based solvers on ARM as baselines, the average speedup is 1.49X and 1.38X, 1.59X and 1.40X for s=4 and s=8.

V. Conclusion

We have proposed Me-MPK, a memory-efficient Matrix-Power Kernel (MPK) designed for shared-memory multi-core systems. Me-MPK effectively leverages both cache reuse and matrix symmetry, significantly enhancing the performance of memory-bound sparse matrix solvers, such as the s-Step CG and BiCGStab methods that use MPK.

REFERENCES

[1] Y. Zhang, S. Li, F. Yuan, D. Dong, X. Yang, T. Li, and Z. Wang, "Memory-aware optimization for sequences of sparse matrix-vector multiplications," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 379–389.

[2] C. Alappat, G. Hager, O. Schenk, and G. Wellein, "Level-based blocking for sparse matrices: Sparse matrix-power-vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 581–597, 2022.

[3] J. Gurhem, M. Vandromme, M. Tsuji, S. G. Petiton, and M. Sato, "Sequences of sparse matrix-vector multiplication on fugaku's a64fx processors," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 751–758.

[4] Y. Saad, *Numerical Methods for Large Eigenvalue Problems: Revised Edition*. SIAM, 2011.

[5] S. Filippone, V. Cardellini, D. Barbieri, and A. Fanfarillo, "Sparse matrix-vector multiplication on gpgpus," *ACM Transactions on Mathematical Software (TOMS)*, vol. 43, no. 4, pp. 1–49, 2017.

[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

[7] R. W. Vuduc, *Automatic performance tuning of sparse matrix kernels*. University of California, Berkeley, 2003.

[8] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, 2009, pp. 233–244.

[9] A. Elafrou, G. I. Goumas, and N. Koziris, "Conflict-free symmetric sparse matrix-vector multiplication on multicore architectures," *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[10] C. Alappat, A. Basermann, A. R. Bishop, H. Fehske, G. Hager, O. Schenk, J. Thies, and G. Wellein, "A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication," *ACM Trans. Parallel Comput.*, vol. 7, no. 3, jun 2020.

[11] H. Qiu, C. Xu, J. Fang, J. Zhang, L. Deng, Y. Ding, Q. Wang, S. Chen, Y. Che, and J. Liu, "A conflict-aware divide-and-conquer algorithm for symmetric sparse matrix-vector multiplication," in *2024 SC24: International Conference for High Performance Computing, Networking, Storage and Analysis SC*. IEEE Computer Society, 2024, pp. 761–775.

[12] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, "Avoiding communication in computing krylov subspaces," *EECS Dept., UC Berkeley, Tech. Rep. UCB/EECS-2007-123*, 2007.

[13] M. Hoemmen, *Communication-avoiding Krylov subspace methods*. University of California, Berkeley, 2010.

[14] E. C. Carson, *Communication-avoiding Krylov subspace methods in theory and practice*. University of California, Berkeley, 2015.

[15] J. Dongarra, S. Tomov, P. Luszczek, J. Kurzak, M. Gates, I. Yamazaki, H. Anzt, A. Haidar, and A. Abdelfattah, "With extreme computing, the rules have changed," *Computing in Science & Engineering*, vol. 19, no. 3, pp. 52–62, 2017.

[16] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, "Minimizing communication in sparse matrix solvers," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–12.

[17] T. Muranushi and J. Makino, "Optimal temporal blocking for stencil computation," *Procedia Computer Science*, vol. 51, pp. 1303–1312, 2015.

[18] J. Morlan, S. Kamil, and A. Fox, "Auto-tuning the matrix powers kernel with sejits," in *High Performance Computing for Computational Science-VECPAR 2012: 10th International Conference, Kope, Japan, July 17-20, 2012, Revised Selected Papers 10*. Springer, 2013, pp. 391–403.

[19] D. Huber, M. Schreiber, and M. Schulz, "Graph-based multi-core higher-order time integration of linear autonomous partial differential equations," *Journal of Computational Science*, vol. 53, p. 101349, 2021.

[20] H. Qiu, C. Xu, J. Fang, L. Deng, J. Zhang, Q. Wang, Y. Ding, Z. Dai, Y. Che, S. Chen, and J. Liu, "Towards scalable unstructured mesh computations on shared memory many-cores," in *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 109–119.

[21] V. H. F. Batista, G. O. Ainsworth, and F. L. B. Ribeiro, "Parallel structurally-symmetric sparse matrix-vector products on multi-core processors," *ArXiv*, vol. abs/1003.0952, 2010.

[22] T. Gkountouvas, V. P. Karakasis, K. Kourtis, G. I. Goumas, and N. Koziris, "Improving the performance of the symmetric sparse matrix-vector multiplication in multicore," *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 273–283, 2013.

[23] J. Xia, C. Cheng, X. Zhou, Y. Hu, and P. Chun, "Kunpeng 920: The first 7nm chiplet-based 64-core arm soc for cloud services," *IEEE Micro*, 2021.

[24] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, pp. 1–25, 2011.

[25] Intel Corporation, *Intel oneAPI Math Kernel Library (oneMKL)*, Intel Corporation, 2024, accessed: March 25, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

[26] M. Naumov, "S-step and communication-avoiding iterative methods," Technical Report NVR-2016-003, NVIDIA, Tech. Rep., 2016.

[27] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, pp. 359–392, 1998.

Near-Memory LLM Inference Processor based on 3D DRAM-to-logic Hybrid Bonding

Sanghyeok Han*, Byungkuk Yoon*, Gyeonghwan Park*, Choungki Song†, Dongkyun Kim† and Jae-Joon Kim*
Seoul National University*, SK Hynix, DRAM Development†
{hansh778, bkyoon, gyeonghwan.park, kimjaejoon}@snu.ac.kr, {choungki.song, dongkyun.kim}@sk.com

***Abstract*—Large language model (LLM) inference poses dual challenges, demanding substantial memory bandwidth and computing resources. Recent advancements in near-memory accelerators leveraging 3D DRAM-to-logic hybrid-bonding (HB) interconnects have gained attention due to their highly parallel data transfer capabilities. We address limitations in previous HB-DRAM accelerators, such as those stemming from distributed controller designs, by introducing an architecture with a centralized controller and dual-IO scheme. This approach not only reduces the chip area overhead but also enables reconfigurable GEMV/GEMM operations, boosting the performance. Simulations for the OPT 66B model show that our proposed accelerator achieves 2.9X, 3.5X, and 2.5X higher performance compared to NPU, DRAM-PIM, and heterogeneous designs (DRAM-PIM + NPU), respectively.**

I. Introduction

Transformer-based large language models (LLMs) [1]–[3] have emerged as pivotal components in various language processing applications, ranging from chatbots to machine translations [4]–[6]. Moreover, these models are extending their influence into other domains such as vision applications and the integration of multiple domains [7]–[10]. LLMs demand substantial external memory due to their vast number of parameters, placing a strain on memory data bandwidth. Consequently, state-of-the-art deep learning platforms, including graphic processing units (GPUs) and neural processing units (NPUs), lean towards utilizing high bandwidth memory (HBM) as the primary memory source.

DRAM processing-in-memory (PIM)-based architectures [11]–[13] have been proposed to significantly boost data bandwidth, leveraging bank-level parallelism. However, DRAM-PIMs face challenges in achieving high performance for end-to-end LLM processing. The advent of multi-batching techniques [14]–[16] underscores a growing need to enhance LLM inference throughput by increasing the proportion of compute-intensive matrix-matrix multiplications (GEMM), which can lead to a degradation in the effectiveness of DRAM-PIM system optimized for memory-bound matrix-vector multiplications (GEMV). To overcome the challenge, heterogeneous systems which employ both NPU and DRAM-PIM together have been recently proposed [17], [18]. While the systems show performance improvement over NPU-only or DRAM-PIM-only system by selectively assigning GEMV and GEMM operations to DRAM-PIM and NPU, achieving equal load balancing between PIM and NPU is challenging, making it difficult to increase resource utilization. Furthermore, Tall-and-Skinny GEMM (TS-GEMM) appearing in multi-batch LLM inference shows both memory-intensive and compute-intensive traits, making both NPU and DRAM-PIM suffer.

To address these multifaceted challenges, we focus on the potential of Hybrid Bonding (HB) technology which provides high-density vertical interconnect between stacked silicon chips, enabling high performance and efficiency [19]. Recently, several accelerators based on 3D DRAM-to-Logic HB interconnect have been reported to demonstrate outstanding performance by exploiting the high compute density and high memory bandwidth which HB technology offers [20]–[22]. However, we observe that several design challenges still remain for efficient accelerator architecture based on HB, especially for LLM applications. First, previous works on near-memory processors based on logic-to-DRAM HB technology [20]–[22] placed a DRAM controller for each DRAM bank to handle data at the bank level to transfer control signals via the bank level 3D HB interconnect. However, assigning a controller for each bank incurs significant area overhead [21]. Second, operating frequency of processing elements is limited to the HB-DRAM bank-level data rate which lowers the performance of the compute-bound GEMM in LLM inference.

To address these challenges, we propose a centralized controller based on a dual I/O scheme, which exploits both conventional DRAM I/O structure and bank-level I/O. The conventional structure I/O for external access in HB-DRAM does not utilize the benefit from bank-level parallelism of HB. However, we notice that its bus structure connects to multiple banks so that the control signals can be broadcast to the banks from the centralized memory controller. Therefore we make the centralized controller send the control signals to banks and the banks just *drop* the data to logic die. Replacing the distributed controller with the centralized controller saves significant area, enabling higher compute density. The proposed dual I/O enables reconfigurable GEMM/GEMV operations with variable operating frequency. Our design demonstrates higher frequency GEMM operations by transferring data at higher per-pin data rate using conventional DRAM I/O as well as lower frequency yet massively parallel GEMV operations using bank-level HB-interconnect.

Our main contributions can be summarized as follows:

- We present the dual I/O scheme and a centralized DRAM controller. The schemes provides significant area savings over previous designs that employed distributed DRAM controller. The dual I/O scheme enables reconfigurable GEMM/GEMV operations with variable operating frequency of processing elements: higher frequency GEMM operations as well as lower frequency yet massively parallel GEMV operations.
- Based on the proposed concepts, we present HB-NPU, a 3D DRAM-to-logic HB technology-based processing-near-memory (PNM) LLM inference accelerator architecture that can support both GEMM and GEMV operations efficiently.
- We demonstrate that HB-NPU exhibits 2.9X, 3.5X, and 2.5X higher performance than NPU, DRAM-PIM, and the heterogeneous system respectively, on OPT-66B using the custom simulator based on DRAMSim3 simulator [23].

II. Background

A. Memory bottleneck in LLM inference

LLM [1]–[3] inference is divided into two phases: prefill and decoding. These phases exhibit significantly different operation intensities (OI). Prefill, characterized by parallel processing of multiple tokens, is dominated by compute-bound GEMM operations, whereas decoding, processing only one token at a time, is dominated by memory-bound GEMV operations. Batching [14]–[16] allows concatenation of tokens from multiple client requests that share the same weight matrix, enabling the conversion of GEMV operations

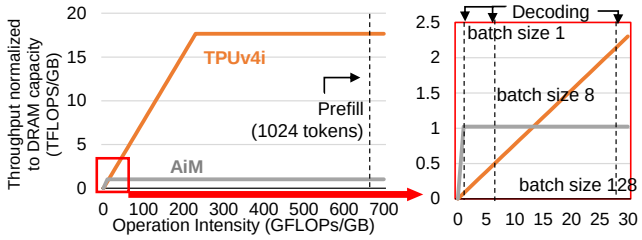


Fig. 1: Roofline model of representative NPU and DRAM-PIM; Google TPUv4i [24], and SK Hynix AiM [11]. Operation intensities for OPT-13B [3] inference with BF16 format and context length of 1024 are used. In the Y-axis, throughput with the same DRAM capacity is shown to compare TPUv4i and AiM which have different DRAM capacities.

in decoding phase to GEMM operations. However, the activation-activation multiplication in attention layer still need to be done with memory-bound GEMV operations even in the multi-batching scenario. Fig. 1 shows the roofline model of state-of-the-art accelerators for OPT-13B model [3]. While TPUv4i [24], a representative NPU, has abundant compute resources, AiM [11], a DRAM-PIM, has extremely high memory bandwidth thanks to its high bank-level parallelism. Given the same DRAM capacity, NPU outperforms the DRAM-PIM in prefill phase with high operation intensity because DRAM-PIM has a much lower compute density than NPU. On the other hand, decoding phase shows extremely low operation density, resulting in substantial memory bottlenecks in NPU. As a result, an efficient GEMV unit as well as an efficient GEMM unit is desired for an efficient LLM inference. With multi-batching, NPU can outperform DRAM-PIM even in decoding phase. Nevertheless, the performance is still much lower than its maximum throughput. This is because attention layers still depend on memory-bound GEMV operations, and it is hard to increase the batch size due to the strain on the memory capacity by memory parameters and KV cache, making input matrix shape tall and skinny (i.e. called Tall-and-Skinny GEMM or TS-GEMM). The size of KV cache increases with the batch size linearly so that the batch size is typically limited to less than 64 in real LLM inference servings [25]. Consequently, it is also desired to develop hardware optimized for TS-GEMM acceleration as well as reconfigurable GEMM/GEMV.

B. Hybrid Bonding (HB) Technology

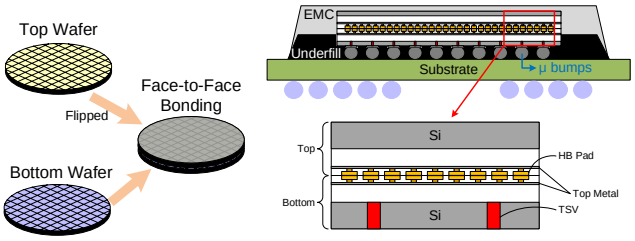
Hybrid bonding (HB) has emerged as a promising alternative to traditional solder bumps, driven by the limitations in scaling solder bump pitch. In 3D System-in-Package (SiP) designs, where efficient chip-to-chip connections are essential, growing data transfer demands can lead to increased power consumption, causing communication inefficiencies. HB technology, with its exceptionally small sub-10μm pitch and low electrical loading, addresses these challenges by enabling more connections, higher bandwidth, and reduced power consumption for data transfer.

Fig. 2 illustrates the wafer-to-wafer HB stacking process in a face-to-face configuration, showcasing the cross-section of the stacked wafers. The top wafer is flipped to align face-down with the bottom wafer, allowing direct contact between HB pads on the top metal layer. This configuration significantly reduces connection thickness, leading to lower parasitic capacitance and resistance and achieving a connection quality similar to on-chip interconnects [19]. TSVs on the bottom die provide off-chip interfaces and power delivery.

III. Proposed Centralized Controller and Dual-I/O

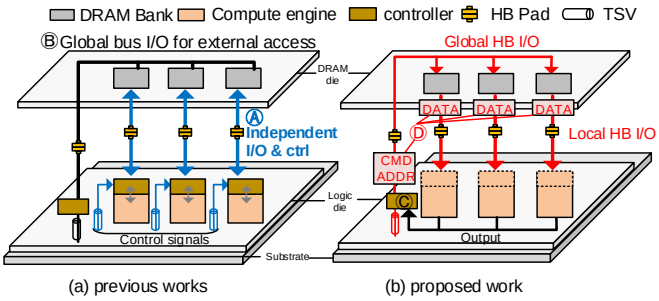
A. Distributed I/O Channels in Previous HB DRAM

The I/O configuration of previous HB DRAM shows a distinct structure compared to conventional DRAM, as depicted in Fig. 3a.



(a) Wafer-to-wafer bond (b) Cross section view

Fig. 2: Illustration of 3D-stacked chip using hybrid bonding



(a) previous works (b) proposed work

Fig. 3: (a) Distributed memory controllers in previous HB DRAM [20]–[22] and (b) the proposed centralized memory controller in our design Note that extra TSVs for power delivery are omitted here.

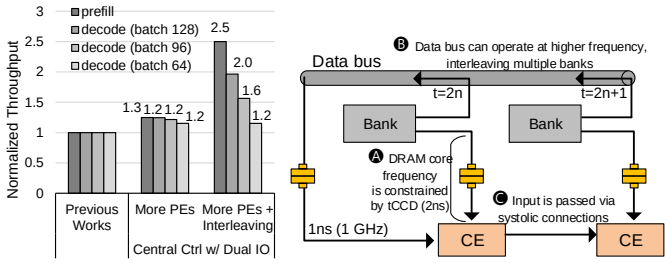
HB DRAM supports independent I/O and controllers for each bank [20]–[22] enabling higher bandwidth through multiple connections between the DRAM die and logic die (Ⓐ). This configuration provides significant advantages in data transfer rates over conventional DRAM systems. However, this distributed memory controller design incurs significant area overhead. Each bank requires dedicated components, including individual control paths, address decoder, timing controls, and Built-In-Self-Test (BIST) units, of which area consumes approximately 17% of a bank area, as reported in [21].

Additionally, the structure requires HB connections not only between memory banks and compute engines but also for off-chip communication via a conventional DRAM I/O global bus shared among multiple banks (Ⓑ). Since the substrate is attached to the logic die, the off-chip datapath from the DRAM banks must pass through HB connections to interface with the TSVs on the backside of the logic die. We leverage this unused I/O with low data bandwidth in previous PNMs to enhance performance and efficiency.

B. Proposed Centralized controller and Dual-I/O

To overcome the limitations in the distributed memory controller-based designs, we propose a centralized controller and dual HB I/O between DRAM and NPU: Local bank-to-engine HB I/O and Global HB I/O (Fig. 3b). We eliminate local memory controllers for each compute engine, and instead use a centralized memory controller for global bus I/O to broadcast the control signals to banks (Ⓒ). In our design, local HB I/O is used to read the data from multiple DRAM banks to corresponding logic in parallel, and global HB I/O is used to read and write the data with lower bandwidth. Data from multiple banks are read in parallel via local HB I/O once each DRAM bank receives command/address through global HB I/O, so each bank does not need a dedicated controller (Ⓓ).

A previous work [21] claimed that distributed controllers were preferred in their vision AI accelerator design since centralized controllers require large area overhead to route and gather the distributed I/Os from different banks to the centralized controller. However, one of our key findings is that the centralized controller



(a) Speedup by Dual-I/O (b) Bank interleaving for high frequency

Fig. 4: (a) Analysis of speedup of the proposed design for OPT-13B with context length 1024 and various batch sizes. We can allocate more processing engines (PEs) by removing local memory controllers [21] and boost PE frequency by bank interleaving. (b) Bank interleaving using Global HB I/O.

is the better option for LLM accelerations. Firstly, the output traffic in matrix multiplication is smaller than the input traffic, so write operations require much lower bandwidth compared to read operations. Therefore, by using shared I/O instead of aggregating all distributed I/O, the number of I/O can be significantly reduced in write operation. Secondly, centralized control is more effective for DRAM access. Although the distributed controllers allow execution of different addresses for each bank in parallel, most operations in LLM, particularly Matrix Multiplication, exhibit regular memory access patterns, thus enabling to broadcast the same address to the banks simultaneously. For instance, in AiM [11], a PIM command is sent to all banks simultaneously with the same bank row and column addresses for executing matrix-vector multiplication.

By employing a centralized DRAM controller that handles both normal access and massively-parallel data access, the area overhead due to distributed controllers can be eliminated and the area can be used for extra compute resources. It enhances the compute density and throughput for compute-intensive layers in LLM. Replacing the distributed controller area with more computing resources results in approximately 1.2x-1.3x improvement in throughput for the OPT-13B model, as shown in Fig. 4a (labeled as *More PEs*).

Additionally, Global HB *data* I/O is useful for input data read in compute-bound operations since it can achieve a higher per-pin data rate than Local HB I/O for the same number of I/Os. By activating multiple banks in parallel and reading them in an interleaved manner, we can circumvent DRAM timing constraints and enhance per-pin data rate, thereby supporting higher logic frequencies. In our HB-NPU design example shown in Fig. 4b, processing engines (PEs) operate at 500MHz due to the 2ns of column-to-column delay (tCCD) timing constraint between each column-read command, when the input data directly come from DRAM banks via Local HB I/O (Ⓐ). On the other hand, Global HB I/O can provide the higher 1GHz frequency by activating two banks in parallel and reading one of them alternately (Ⓑ). To supply inputs from global I/O to all PEs, we organized all PEs connected via systolic connections to efficiently pass the inputs to neighboring PEs (Ⓒ). Utilizing this bank interleaving technique, we use Global HB I/O to feed input data into the PE array during compute-bound GEMM operations, which do not require the high memory bandwidth provided by Local HB I/Os. Such variable frequency operations using the proposed dual I/O further improve the throughput as shown in Fig. 4a, particularly in prefill and larger batches with high operation intensities.

IV. HB-NPU: Architecture and Dataflow

Based on the aforementioned centralized controller and dual-I/O, we develop a prototype for an LLM inference accelerator, *HB-NPU*.

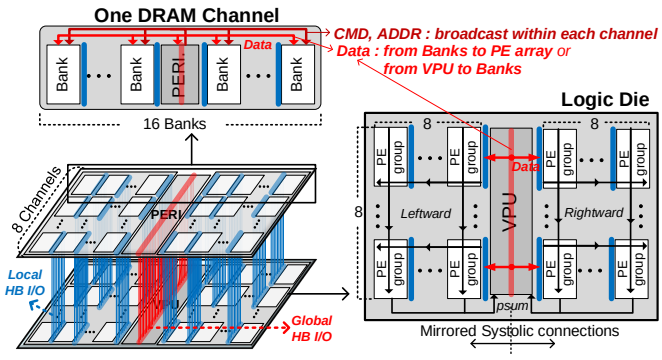


Fig. 5: Organization of the proposed HB-NPU.

For a fair comparison, we matched the compute-memory ratio of TPUv4i [24] in our design by constructing a single chip consisting of a 2GB DRAM die and a Logic die with a 128×128 systolic array and a Vector Processing Unit (VPU) with 128×8 lanes for BF16.

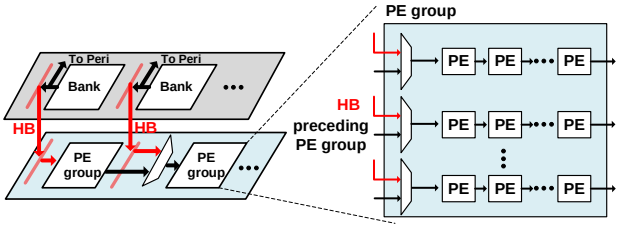
A. HB-NPU Organization

Fig. 5 shows the overall organization of the proposed HB-NPU architecture, which consists of a DRAM die and a logic die that are connected with HB interconnect. The DRAM bank array and the systolic array are tightly coupled so that a DRAM bank and the corresponding group of Multiply-Accumulate (MAC) processing engines (PEs) are placed at the same coordinate in each die and connected via Local HB I/O vertically. Typically, DRAM peripheral circuits for off-chip interface are located at the center of the DRAM die. Consequently, we also place VPU and peripheral units at the center of the Logic die and connect the logic units with DRAM via Global HB I/O. VPU handles the post-processing of output before writing it back to DRAM via Global HB I/O. The PE array is split into two 128×64 configurations to support the placement.

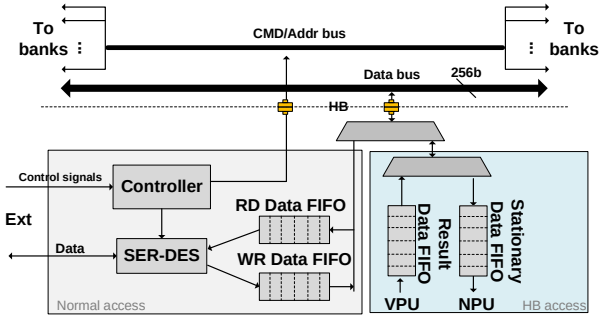
Memory Organization The DRAM die in HB-NPU consists of 8 channels with 16 banks per channel as shown in Fig. 5. The control signals and data coming into DRAM via Global HB I/O (red line) are broadcast across all banks within the channel but not shared across channels. The column data I/O from each bank is 256-bits that can serve 16 PEs (BF 16) at once. To feed the 128 PEs in a column simultaneously, vertical 8 banks must be always activated. To reduce the bank activation overhead, we align the channels with the PE array column. The DRAM timing constraint, tFAW (Four Activation Window) limits the number of banks *per channel* that can be activated within a specified time window which incurs significant latency overhead.

PE Array Organization As depicted in Fig. 5, following the 8 channels with 16 banks per channel configuration of the DRAM, the PE array in the Logic die is also structured into PE groups with an 8×16 configuration. The left and right sides of the VPU are mirrored, facilitating the connection of Global I/O to the PE interconnection and enabling data transferred into the PE array.

Unlike the DRAM-PIM [11] which can place only 16×1 PEs for each DRAM bank due to the area limitation, HB-NPU places 16×8 PEs as a *PE group* for each DRAM bank counterpart, thereby increasing compute resource significantly. Each DRAM bank supplies input and weight data to the PE group via the Local HB I/O. The final partial sum values are sent to VPU for accumulation and post-processing such as activation functions, normalization, softmax. The completed output data is sent back from VPU to the DRAM die via Global HB I/O and stored in banks.



(a) Local HB connection and datapath of input in a PE group.



(b) Global HB connection of one memory channel. Upper part of dotted line is on DRAM die peripheral and the lower part is on the logic die VPU.

Fig. 6: Datapaths using Dual HB I/O connections.

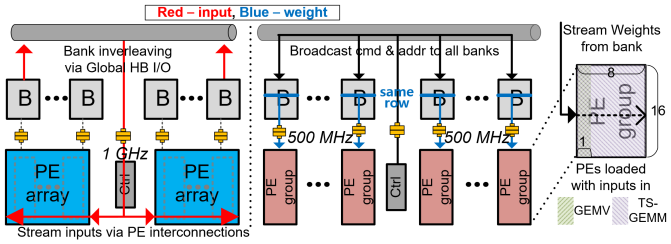


Fig. 7: Reconfigurable dataflow of HB-NPU. We use weight stationary flow with higher frequency for GEMM and input stationary with all-bank activation via Local HB for GEMV/TS-GEMM with lower frequency.

Datapaths using Dual HB I/O Fig. 6 shows the datapaths of Local HB I/O and Global HB I/O. The Local HB I/Os stream data from each DRAM bank to the PE array directly (Fig. 6a). Data coming from the bank either goes to the DRAM peripheral or moves through the Local HB I/O, where the consecutive columns of a bank row are streamed into the PE array sequentially. The first PE columns of each group of PEs can select the input data either from the preceding PE group or upper DRAM bank via the Local HB I/O. Fig. 6b shows the Global I/O datapath of one memory channel. Commands and addresses are sent to the banks by the centralized controller, and data, multiplexed between the external path and the NPU in the logic die, is communicated to the data bus in the DRAM die shared by the banks within each channel. Except for the external I/O interface including controller, Serializer-Deserializer (SER-DES), and data FIFOs, we have two extra FIFOs for result data from VPU and stationary data into NPU. The output data comes out from the systolic array and enters into Result Data FIFO and input data is supplied through the Stationary Data FIFO to load the PE registers. Therefore, we design Dual-IO support with minimal modification of Result Data FIFO and Stationary Data FIFO, exploiting existing DRAM I/Os.

B. Reconfigurable GEMM/GEMV dataflow of HB-NPU

Since each PE group in HB-NPU can receive input either vertically from DRAM or horizontally from the preceding PE group, we can selectively configure the dataflow for GEMM or GEMV kernels based on the given workloads. We adopt the weight-stationary dataflow for GEMM and input-stationary dataflow for GEMV considering the impact of DRAM timing constraints. Weight-stationary dataflow is efficient for GEMM operations since weights have a high chance of reuse. On the other hand, weight-stationary dataflow is inefficient in GEMV case since there is no reuse chance of weights when supplying inputs, necessitating frequent DRAM bank row switching to load the next weight tile. DRAM bank row switching causes substantial energy consumption and significant delay due to timing constraints. Therefore, it is more efficient to preload the input into PE registers and reuse it multiple times by streaming the weight matrix. Consequently, we opt for input-stationary dataflow for GEMV operations in our design.

GEMM Operations For GEMM, as shown in Fig. 7a, weight data is preloaded into PE registers, and input data is streamed into the PEs in a systolic fashion via Global HB I/O at 1 GHz frequency thanks to the bank interleaving technique. As described in Fig. 4b, Global HB I/O can boost memory bandwidth by accessing several banks in an interleaved manner, enabling 1 GHz frequency operation which is 2X higher frequency than the case when Local HB I/O is used.

GEMV Operations For GEMV, conversely, we preload inputs via Global HB I/O. Then, we activate all DRAM banks for memory-intensive weight streaming into PEs via local HB I/O's by broadcasting the command and address to all banks (Fig. 7b). While the operation frequency of PEs is restricted by the DRAM timing constraint by using Local HB I/O, massively parallel weight streaming significantly accelerates GEMV operations compared to conventional NPUs. As described in section III-B, matrix multiplication in LLM does not require a complex memory access pattern. Therefore, we can easily map LLM parameters on DRAM to access the same row of all DRAM banks simultaneously similar to conventional DRAM-PIM architectures. The large PE array in our HB-NPU is divided into 16 PE groups, and in each PE group, only the first PE column is involved in GEMV computation since the streamed weights can be computed by only one input vector.

Tall-and-Skinny GEMM Meanwhile, batch sizes in LLM inference are often limited in multi-batching scenarios due to the pressure on memory capacity by KV cache burden, making inputs become a tall and skinny matrix. On a conventional systolic array (e.g. 128×128), such tall and skinny GEMM (TS-GEMM) workloads lead to reduced PE utilization. The proposed HB-NPU has a unique feature to address both PE utilization and memory bandwidth issues by utilizing input stationary dataflow for TS-GEMM. We run TS-GEMM with high memory bandwidth by streaming weights into the PE groups via Local HB I/Os, while keeping inputs stationary in PE registers. As shown in Fig. 7b (right), larger number of PE columns in each PE group participate in the TS-GEMM computation than the GEMV dataflow in which only the first PE column in each PE group is utilized.

Data Mapping Each input and weight tile is divided into 8 memory channels and distributed to all PE rows via Global HB I/O. Weights are prearranged in DRAM prior to execution. For the activation-activation multiplication in attention layers, we evenly allocate different heads to each PE group, optimizing PE utilization. For arranging input data, since Global HB I/O can access any DRAM banks, VPU can send the completed output data of producer layers to the target

bank locations via Global HB I/O to make them ready for processing of the next consumer layers.

V. Evaluation

A. Methodology

Baselines We assess our architecture against three types of representative LLM accelerators: NPU, DRAM-PIM, and heterogeneous system (NPU+PIM). All baselines and HB-NPU have the same 32GB capacity of DRAM on a board and the boards are interconnected by PCIe 4.0 16x for scaling-out to simulate large-size models. Detailed specifications are provided in Table I.

- **NPU**: NPU baseline features a 4 128×128 systolic array with 8 GB HBM2 and vector processors referring to [24]. We simulated the NPU with ScaleSim cycle-level simulator [26].
- **DRAM-PIM**: DRAM-PIM baseline has a similar configuration to [11], that has 16 processing elements per bank and one global buffer per channel. We built a DRAM-PIM simulator based on the DRAMSim3 simulator [23].
- **Heterogeneous system (NPU+PIM)**: Recently, heterogeneous systems combining NPUs and DRAM-PIMs [17], [18] have been proposed to accelerate both GEMM and GEMV operations in batched LLM decoding. Note that the heterogeneous system has 2x larger DRAM capacity than that of both the baseline and HB-NPU, with a half allocated to NPU and the other half to PIM. We assigned FC layers with GEMM operations to the NPUs and Attention layers with GEMV operations to the PIMs with NPU-PIM pipelining techniques that enhance system utilization.

Cycle-level simulation of HB-NPU We built a cycle-level simulator for HB-NPU based on the DRAMSim3 simulator[1] [23]. We added custom commands for HB-NPU operations and implemented a custom command generator for BLAS functions. We developed a custom Python program to generate a DRAM trace file from model information and memory specifications. Our program allows users to input various LLM workload information (e.g., number of heads, hidden dimensions, etc.), dataflow options, and DRAM specifications (DRAM capacity, device width, etc.). Then the program divides the operand data and allocates them to the physical address of the bank. We also consider chip-to-chip communication overhead for large-scale system. The generated trace file is read into the simulator and our custom command generator then derives DRAM commands from the provided transactions.

LLM Workloads We simulated the OPT model variations [3] across four different parameter sizes for multi-batch inference of LLM. We conducted experiments for batch sizes of 32, 64, and 128 using two token lengths, (1024, 128) and (128, 1024) for input and output tokens. Model parallelism is employed to distribute parameters and KV caches across multiple devices, with a mixture of pipeline parallelism (PP) [27] and tensor parallelism (TP) [28]. While high TP increases communication overhead, high PP divides a batch into several micro-batches, leading to PE under-utilization. Table II shows the LLM configuration with the model parallelism; Tensor parallelism (TP) and Pipeline parallelism (PP) we exploited.

B. Results

Decoding results Fig. 8 shows the simulation results for throughput of baselines and HB-NPU for multi-batch inference cases. While NPU throughput increases as the batch size increases, DRAM-PIM exhibits relatively low throughput because it is optimized to GEMV operations. The heterogeneous system, incorporating both NPU and PIM modules, demonstrates higher performance than NPU-only or DRAM-PIM-only baselines. However, TS-GEMM workloads with low batch sizes cause both NPU and PIM modules to suffer compared to HB-NPU. The HB-NPU, with its support for a reconfigurable systolic array and high bandwidth, demonstrates the highest throughput compared to the baselines. Overall, our HB-NPU shows 3.0x, 4.1x, and 2.0x performance improvement on average over NPU, DRAM-PIM, and the heterogeneous system (DRAM-PIM + NPU), respectively. The higher performance of HB-NPU is particularly noticeable for larger model with lower batch size, which is expected to be a main workload in inference servings due to the increasing demands for larger AI model and longer context [33], [34].

Fig. 9 shows the breakdown of runtime among various inference engines. Fig. 9a represents the case in which computation of attention layer is dominant with long input sequence, while Fig. 9b represents the case when the FC layer is dominant with large model dimension. In Fig. 9a, the NPU is suffering from inefficiency in GEMV computations, because of the large size of KV cache with long input sequence. As batch size increases, both NPU and PIM exhibit significantly larger runtime increases compared to the other two systems, due to their lack of ability to accelerate GEMV or GEMM, respectively. In Fig. 9b, the runtimes for both the NPU and the heterogeneous system (which includes an NPU) decrease as the batch size increases, leading to higher PE utilization. In contrast, DRAM-PIM's computation time for the FC layer increases linearly with batch size, making it significantly slower than the other systems. In both scenarios, HB-NPU exhibits the best performance, effectively computing both FC and attention layers. While heterogeneous system also effectively computes both layers, they feature dedicated processing units for GEMM and GEMV so that the mismatch in the ratio of GEMM

[1]HB-NPU simulator is available at https://github.com/shh8228/HB-NPU-simulator.git

TABLE I: Hardware specification of baselines and HB-NPU.

NPU	Composition	4 128x128 PE arrays, 4 128x8 vector units 144 MB SRAM, 1 GHz
	Memory	2 4H HBM2 of 8 GB (1 GB/slice); off-chip bandwidth of 614 GB/s
	System configuration	4 chips/board: 32GB with 16 128x128 PEs Inter-chip bandwidth of 50 GB/s
PIM	Memory configuration	4H Stacked DRAM of 8 GB (2 GB/die) 8 channels/die, 16 banks/channel Column I/O 256 b, row size 2KB Core frequency 500 MHz
	Timing parameters	$t_{RCD} = 14ns, t_{RP} = 14ns$ $t_{CCD} = 2ns, t_{FAW} = 30ns$
	PEs	500 MHz, 16-lane MAC/bank
	System configuration	4 chips/board: 32 GB with 32K PEs 2 KB input buffer per channel
Ours	Memory die	2 GB Capacity, 8 channels, 16 banks/channel Column I/O 256b, row size 2KB Core frequency 500 MHz
	Timing parameters	$t_{RCD} = 14ns, t_{RP} = 14ns$ $t_{CCD} = 2ns, t_{FAW} = 30ns$
	Logic die	8x16 PGs, 16x8 PEs per PG 128x8 vector unit, 1MB output accumulator 1 GHz(GEMM)/500 MHz(GEMV, TS-GEMM)
	System configuration	16 chips/board: 32GB with 16 128x128 PEs Inter-chip bandwidth of 50 GB/s

TABLE II: OPT and model parallelism configuration. *Heterogeneous system has the twice larger memory capacity. $^{+}$HB-NPU has 4x higher TPs due to lower chip capacity.

Model	#L	#H	d_{model}	(TP^{+},PP)	DRAM Capacity*
2.7B	32	32	2560	(8,1)	64 GB
6.7B	32	32	4096	(8,2)	128 GB
13B	40	40	5120	(16,2)	256 GB
66B	64	72	9216	(32,4)	1024 GB

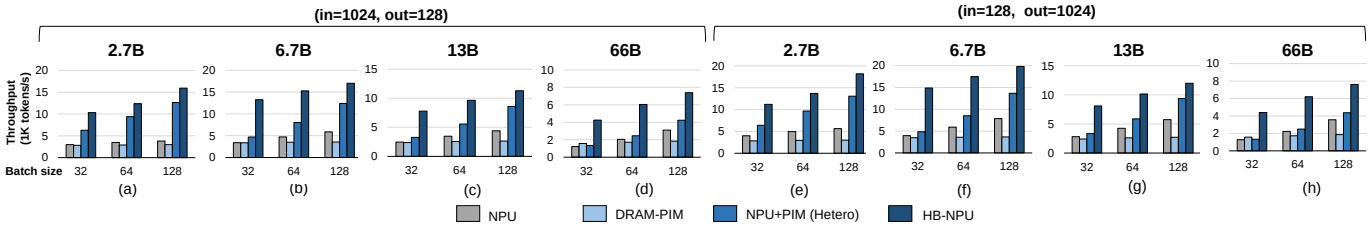


Fig. 8: Throughput comparison for decoding in OPT models with various batch sizes and token sizes.

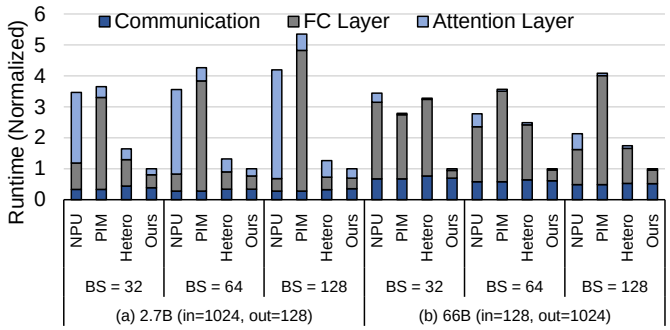
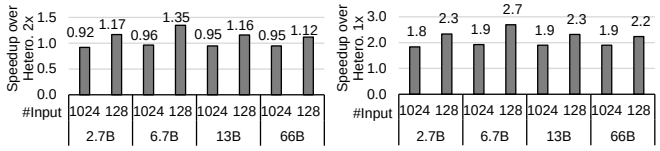


Fig. 9: Runtime breakdown for decoding, normalized to ours (HB-NPU).



(a) Heterogeneous system with doubled DRAM capacity.

(b) Heterogeneous system with the same DRAM capacity as HB-NPU.

Fig. 10: Prefill speedup of HB-NPU compared to heterogeneous system with (a) doubled DRAM capacity and (b) same DRAM capacity as HB-NPU .

to GEMV operations leads to lower PE utilization. In contrast, HB-NPU utilizes reconfigurable processing units, leading to superior performance across various scenarios.

Prefill results Fig. 10a shows the speedup of HB-NPU in prefill, compared to heterogeneous system. HB-NPU shows higher performance than heterogeneous system with 128 input tokens due to the lower operation intensity. With the larger token size 1024, heterogeneous system is slightly faster than HB-NPU because heterogeneous system utilizes larger system resources than HB-NPU. However, considering the difference in the memory capacity and number of PEs, HB-NPU achieves higher efficiency than heterogeneous system.

Implication of DRAM/PE resources in Heterogeneous System In the experimental setting for Fig. 8 and 10a, we assumed that the heterogeneous system has twice larger DRAM size and more PEs than other baselines and our HB-NPU as the heterogeneous system consists of an NPU with dedicated DRAM and a DRAM-PIM similar to [18]. Therefore, for a fair comparison, it is worth comparing HB-NPU with the heterogeneous system that has the same DRAM capacity. Fig. 10b and Fig. 11 show that the performance gap between our HB-NPU and the heterogeneous system increases as the DRAM capacity and the number of PEs in the heterogeneous system are reduced.

C. Area and Energy Analysis

Area overhead of VPU To assess whether the VPU unit can be implemented in the area budget of a conventional DRAM peripheral area [29], we first synthesized the VPU of 128x16 ALUs with

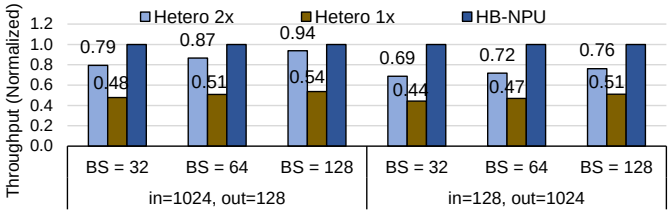


Fig. 11: Throughput comparison of HB-NPU (1024 GB DRAM) with the heterogeneous system with 2048 GB DRAM (Hetero 2x) and 1024 GB DRAM (Hetero 1x) for OPT-66B. Throughput values are normalized to that of HB-NPU.

accumulation buffer following the TPU series setup [30], with a 28nm technology. Then, we scaled the area to 14nm using the equation from [32]. We also included the area of DRAM Physical layer (PHY) based on the HBM3 specification. We observed that the VPU and DRAM PHY consume 83.5% of the target area budget, implying that no extra area is required.

Energy analysis Our HB-NPU consumes much lower energy than other baselines, due to its energy-efficient dataflow using Dual-I/O. HB-NPU consumes lower energy than NPU due to the short HB interconnect. HB-NPU also consumes lower energy than DRAM-PIM thanks to its more energy-efficient GEMM dataflow than that of DRAM-PIM. Overall, simulation results using DRAMsim3 [23] show that HB-NPU consumes about 19x, 9x, 3x lower energy than DRAM-PIM, NPU, and heterogeneous system, respectively, on the OPT-2.7B model.

VI. Conclusion

We introduce HB-NPU, a versatile PNM architecture for LLM inference designed to accelerate both GEMM and GEMV operations leveraging 3D DRAM-to-Logic HB technology. HB-NPU alleviates memory bottlenecks and enhances compute resource density through its centralized controller and dual HB I/O schemes. This reconfigurability for GEMM/GEMV operations is significant, as most existing AI accelerators are optimized for either GEMM or GEMV. Moreover, HB-NPU's support for tall-and-skinny GEMM enables higher performance in LLM inference involving limited batch size. The combination of these features positions HB-NPU as a promising solution for efficient, high-performance LLM inference processing.

VII. Acknowledgement

This work was supported in part by SK Hynix and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2023-00208606, NeuroHub+: Scheduler and Simulator for General In-Memory Neural Network Accelerators, IITP-2023-RS-2023-00256081: artificial intelligence semiconductor support program to nurture the best talents), BK21 FOUR program at Seoul National University, and ISRC at Seoul National University. The EDA tool was supported by the IC Design Education Center(IDEC). (Corresponding Author: Jae-Joon Kim).

REFERENCES

[1] T. Brown, *et al*. "Language models are few-shot learners," *NeurIPS*, 2020.

[2] H. Touvron, *et al.,* "Llama: Open and efficient foundation language models," *arXiv preprint*, 2023.

[3] S. Zhang, *et al.,* "Opt: Open pre-trained transformer language models," *arXiv preprint*, 2022.

[4] J. Wei, *et al.,* "Leveraging large language models to power chatbots for collecting user self-reported data," *HCI*, 2024.

[5] S. Pujar, *et al.,* "Invited: Automated Code generation for Information Technology Tasks in YAML through Large Language Models," *DAC*, 2023.

[6] B. Zhang, *et al.,* "Prompting large language model for machine translation: a case study," *ICML*, 2023.

[7] W. Wang, *et al.,* "Visionllm: Large language model is also an open-ended decoder for vision-centric tasks," *NeurIPS*, 2023.

[8] D. Gufran, *et al.,* "VITAL: Vision Transformer Neural Networks for Accurate Smartphone Heterogeneity Resilient Indoor Localization," *DAC*, 2023.

[9] Q. Ye, *et al.,* "Mplug-octopus: The versatile assistant empowered by a modularized end-to-end multimodal llm," *ACMMM*, 2023.

[10] C. Yang, *et al.,* "Late Breaking Results: Fast Fair Medical Applications? Hybrid Vision Models Achieve the Fairness on the Edge," *DAC*, 2023.

[11] S. Lee, *et al.,* "A 1ynm 1.25v 8gb, 16gb/s/pin gddr6-based acceleratorin-memory supporting 1tflops mac operation and various activation functions for deep-learning applications," *ISSCC*, 2022.

[12] Y.-C. Kwon, *et al.,* "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2tflops programmable computing unit using bank-level parallelism, for machine learning applications," *ISSCC*, 2021.

[13] J. Park, *et al.,* "NTT-PIM: Row-Centric Architecture and Mapping for Efficient Number-Theoretic Transform on PIM," *DAC*, 2023.

[14] G.-I. Yu, *et al.,* "Orca: A distributed serving system for Transformer-Based generative models," *OSDI*, 2022.

[15] W. Kwon, *et al.,* "Efficient memory management for large language model serving with pagedattention," *SOSP*, 2023.

[16] R. Pope, *et al.,* "Efficiently scaling transformer inference," *MLSys*, 2023.

[17] G. Heo, *et al.,* "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," *ASPLOS*, 2024.

[18] J. Park, *et al.,* "Attacc! unleashing the power of pim for batched transformerbased generative model inference," *ASPLOS*, 2024.

[19] B. Ayoub, *et al.,* "Impact of process variations on the capacitance and electrical resistance down to 1.44 μm hybrid bonding interconnects," *EPTC*, 2020.

[20] D. Niu, *et al.,* "184qps/w 64mb/mm23d logic-to-dram hybrid bonding with process-near-memory engine for recommendation system," *ISSCC*, 2022.

[21] Z. Yue, *et al.,* "Exploiting similarity opportunity of emerging ai models on 3d hybrid bonding architecture," *ISCA*, 2024.

[22] B. Fujun, *et al.,* "A stacked embedded dram array for lpddr4/4x using hybrid bonding 3d integration with 34gb/s/1gb 0.88pj/b logic-to-memory interface," *IEDM*, 2020.

[23] "Dramsim3," https://github.com/umd-memsys/DRAMsim3.

[24] N. P. Jouppi, *et al.,* "Ten lessons from three generations shaped google's tpuv4i : Industrial product," *ISCA*, 2021.

[25] P. Patel, *et al.,* "Splitwise: Efficient Generative LLM Inference Using Phase Splitting," *ISCA*, 2024.

[26] A. Samajdar, *et al.,* "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint*, 2019.

[27] Y. Huang, *et al.,* "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *NeurIPS*, 2019.

[28] D. Narayanan, *et al.,* "Efficient large-scale language model training on gpu clusters using megatron-lm," *SC*, 2021.

[29] M.-J. Park, *et al.,* "A 192-gb 12-high 896-gb/s hbm3 dram with a tsv auto-calibration scheme and machine-learning-based layout optimization," *JSSC*, vol. 58, no. 1, pp. 256–269, 2023.

[30] T. Norrie, *et al.,* "The design process for google's training chips: Tpuv2 and tpuv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, 2021.

[31] N. P. Jouppi, *et al.,* "In-datacenter performance analysis of a tensor processing unit," *ISCA*, 2017.

[32] A. Stillmaker, *et al.,* "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, 2017.

[33] A. Chowdhery, *et al.,* "Palm: Sclaing language modeling with pathways," *Journal of Machien Learning Research*, 2023.

[34] Gemini Team, *et al.,* "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint*, 2024.

DuQTTA: Dual Quantized Tensor-Train Adaptation with Decoupling Magnitude-Direction for Efficient Fine-Tuning of LLMs

Haoyan Dong[1], Hai-Bao Chen[1,*], Jingjing Chang[1], Yixin Yang[1],
Ziyang Gao[1], Zhigang Ji[1], Runsheng Wang[2], Ru Huang[2]
[1]The Department of Micro-Nano Electronics, Shanghai Jiao Tong University, Shanghai, China
[2]School of Integrated Circuits, Peking University, Peking University, Beijing, China
[*]Corresponding author: haibaochen@sjtu.edu.cn

***Abstract*—Recent parameter-efficient fine-tuning (PEFT) techniques have enabled large language models (LLMs) to be efficiently fine-tuned for specific tasks, while maintaining model performance with minimal additional trainable parameters. However, existing PEFT techniques continue to face challenges in balancing both accuracy and efficiency, especially when addressing scalability and the demands of lightweight deployment for LLMs. In this paper, we propose an efficient fine-tuning method of LLMs based on dual quantized Tensor-Train adaptation with decoupling magnitude-direction (DuQTTA). The proposed DuQTTA method employs Tensor-Train decomposition and dual-stage quantization to minimize model size and resource consumption. Additionally, it employs an adaptive optimization strategy and a decoupled update mechanism to improve model performance, thereby minimizing suboptimal outcomes and ensuring alignment with the full-parameter fine-tuning goals. Experimental results indicate that the proposed DuQTTA method outperforms existing PEFT methods, achieving up to a 65× compression rate compared to the LLaMA2-7B models, meanwhile delivering improvements of 4.44%, 3.14%, and 0.97% over LoRA on LLaMA2-7B, LLaMA3-8B, and LLaMA2-13B, respectively. The proposed DuQTTA method is effective in compressing LLMs for deployment on resource-constrained edge devices.**

***Index Terms*—LLMs, Tensor-Train, dual quantization, adaptive optimization strategy, magnitude-direction decoupled**

I. Introduction

The leading LLMs today are pre-trained on extensive general-domain datasets, showcasing strong generalization across applications like question-answering, summarization, and translation. However, practitioners often seek to tailor these broad models to specific downstream tasks, particularly smaller, specialized ones. This requires adjusting the pre-trained model's weights to optimize performance on these new tasks. With the rapid expansion in the size of LLMs and their training datasets, full-parameter fine-tuning for downstream adaptation has become computationally prohibitive, as it necessitates extensive weight modifications through costly gradient-based methods.

PEFT methods have been developed for enabling fine-tuning by adjusting only a minimal subset of parameters [1]. One such efficient technique, Low-Rank Adaptation (LoRA) [2], achieves parameter efficiency without altering model architecture. LoRA confines training to added low-rank matrices within the pre-trained weights, significantly reducing the number of trainable parameters and thereby lowering training costs. Tensor decomposition further aids in fine-tuning by reducing the number of parameters required, as seen in LoRETTA [3]. Notably, LoRA's restriction to a low-rank subspace imposes coupling between parameter magnitude and direction changes within this space, limiting independent variation in both dimensions. As observed in [4], LoRA fine-tuning may face suboptimality issues as model width grows, leading them to recommend varied learning rates. Despite these advancements, PEFT methods still struggle to balance accuracy with minimizing trainable parameters.

In this paper, we propose DuQTTA, an ultra-low-parameter and high-precision PEFT method that delivers performance on par with or better than full-parameter fine-tuning while substantially reducing trainable parameters and memory requirements. This approach enhances the efficient deployment of LLMs, especially on edge devices. DuQTTA tackles these challenges by integrating dual quantization (DQ) with Tensor-Train (TT) decomposition, an adaptive optimization strategy (AOS), and a decoupled update mechanism. The main contributions of DuQTTA are as follows:

- We propose dual quantization and TT decomposition within DuQTTA to achieve a substantial reduction in trainable parameters. By first applying TT decomposition, we reparameterize model weights into TT cores. This is followed by dual quantization process: the TT cores undergoes initial quantization, and then a second quantization is applied to the reconstructed tensor. This approach significantly lowers resource cost with minimal accuracy loss.
- We propose AOS that addresses potential suboptimal fine-tuning in large-width conditions by dynamically adjusting learning rates after tensor core reconstruction. By tuning the learning rates across different components, this strategy effectively resolves performance bottlenecks associated with large-scale parameter spaces, boosting fine-tuning efficiency and model performance.
- We propose a magnitude-direction decoupled update mechanism that decomposes pre-trained weights into two components—magnitude and direction—to facilitate fine-tuning. By leveraging TT for directional updates, our approach significantly reduces the number of trainable parameters while enabling more precise updates. This refined parameter adjustment enhances the adaptability and effectiveness of the model across a wide range of downstream tasks.

II. Related Work

A. Parameter-Efficient Fine-Tuning

PEFT reduces the fine-tuning costs of LLMs by training a relatively small subset of parameters to adapt to downstream tasks. Existing PEFT methods can be categorized into adapter-based methods, prompt-based methods, and LoRA and its

This work was supported in part by the Nature Science Foundation of China under Grants T2293704 and T2293700.

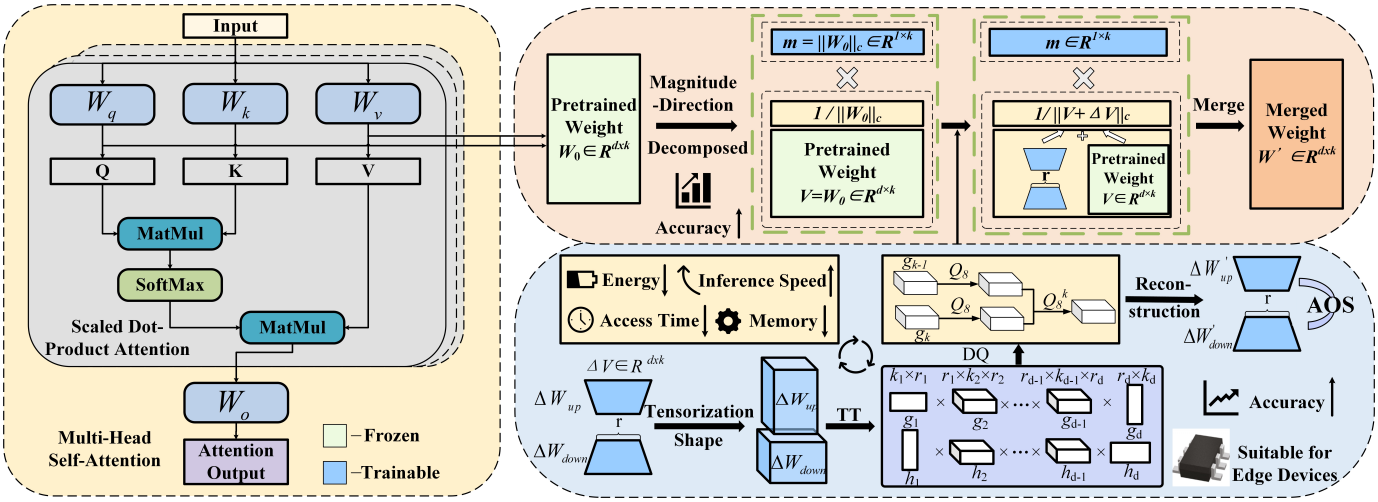


Fig. 1. The overall architecture of the prposed DuQTTA with decoupling magnitude-direction for efficient fine-tuning of LLMs.

variants. The first category, known as adapter-based methods, involves introducing additional trainable modules into the original frozen backbone, as seen in works [5–7]. Houlsby et al. [1] proposed adding linear modules sequentially to existing layers, while He et al. [5] suggested integrating these modules in parallel with the original layers to enhance performance. The second category, prompt-based methods, focuses on introducing trainable tokens into the input or hidden layers of the base model and fine-tuning these vectors, significantly reducing the number of trainable parameters, though sometimes at the cost of accuracy, as shown in [8–10]. However, both types may lead to inference latency compared to the original model, and this issue becomes more pronounced as model size increases.

B. LoRA and Its Variants

During fine-tuning, LoRA introduces low-rank matrices as side branches to approximate weight updates in the original pre-trained weights [2]. This approach avoids additional inference latency while allowing the low-rank matrices for downstream tasks to be seamlessly merged with the pre-trained weights before inference. For instance, Dettmers et al. [11] introduced QLoRA, a quantized version of LoRA, which further reduces computational costs by quantizing pre-trained weights down to as few as 4 bits. Additionally, Yang et al. [3] incorporates Tensor-Train decomposition during fine-tuning, significantly reducing the number of trainable parameters by decomposing the two update matrices of LoRA. Meanwhile, Hayou et al. [4] addresses the suboptimal updates in LoRA under large-width conditions, proposing different learning rates for the two update matrices to improve fine-tuning performance. To further refine the training of quantized LoRA, Li et al. [12] introduced a new method called LoftQ, which enables better initialization for quantized training. Meanwhile, other recent works have focused on reducing the number of parameters in LoRA through various approaches, such as using different ranks for update matrices across layers [13, 14], applying fixed low-rank projections [15], and parameterizing low-rank matrices with a random basis [16].

III. METHODOLOGY

A. Overall Architecture of the Proposed DuQTTA

We propose an advanced architecture named DuQTTA, which achieves high precision while maintaining low resource consumption. Fig. 1 illustrates its optimization process. First, DuQTTA utilizes a magnitude-direction decoupled update mechanism that separates the pre-trained weights obtained from the Q and V components within the attention mechanism of LLMs into distinct magnitude and direction components. The direction component is then decomposed using TT decomposition, significantly reducing the number of trainable parameters. Second, DQ is applied to the TT cores. Each TT core is first quantized to 8-bit precision, followed by an 8-bit quantization of the reconstructed tensor. This approach further reduces memory consumption and power consumption while accelerating inference, making it more suitable for deployment on edge devices. Finally, the architecture incorporates AOS to update the reconstructed matrices, further improving accuracy.

B. Tensorized TT Layer and Tensor Reparameterization

In this section, we present an approximate representation of the linear layers in LLMs that require fine-tuning, achieved by reducing the number of trainable parameters using the TT format [17]. Prior to applying the TT decomposition, the original weight matrix W is reshaped into a d-dimensional tensor $\mathcal{W} \in \mathbb{R}^{l_1 \times \cdots \times l_d}$, where entries are indexed by d indices $p_1, p_2, \ldots, p_k, \ldots, p_d$. Each p_k is within $[1, l_k]$ for $k = 1, 2, \ldots, d$. The high-order tensor $\mathcal{W} \in \mathbb{R}^{l_1 \times \cdots \times l_d}$ can then be decomposed and represented by using a chain of d TT cores $\mathcal{G}_1, \ldots, \mathcal{G}_k, \ldots, \mathcal{G}_d$ [18], where each core $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times l_k \times r_k}$ for $k \in [1, d]$. Each element in W can be derived from these TT cores, given by

$$\mathcal{W}_{p_1, p_2, \ldots, p_d} = \prod_{k=1}^{d} \mathcal{G}_{p_k}^{(k)}, \tag{1}$$

where $\mathcal{G}_{p_k}^{(k)} = \mathcal{G}^{(k)}(:, p_k, :)$ is the p_k-th lateral slice of the k-th TT core $\mathcal{G}^{(k)}$. $r \in \{r_0, r_1, \ldots, r_d\}$, called the TT rank with $r_0 = r_d = 1$, directly affects the compression rate of the TT decomposition. During fine-tuning, only these TT cores are stored, significantly reducing the memory footprint compared to the full weight matrix. It should be noted that the TT decomposition reduces the number of parameters from the original $M \times N$ to $\sum_{k=1}^{d} r_{k-1} \times l_k \times r_k$.

To fully exploit the intrinsic low-rank structure of the weight matrix, we extend the reparameterization of the update matrices by incorporating tensor factorization on top of the

standard LoRA framework. Specifically, the update process can be written as follows:

$$y = W_0 x + \Delta W x = W_0 x + BAx, \tag{2}$$

where x and y represent the input and output vectors, respectively. $W_0 \in \mathbb{R}^{h\times l}$ denotes the pretrained weight matrix, while $B \in \mathbb{R}^{h\times r}$ and $A \in \mathbb{R}^{r\times l}$ are the low-rank matrices that constitute the update matrix ΔW. Here, $r \ll \min\{h, l\}$ is the LoRA rank.

In the proposed DuQTTA method, we reshape the two update matrices ΔW_{up} and ΔW_{down} into tensors of shapes $l_1 \times \cdots \times l_d$ and $j_1 \times \cdots \times j_d$, respectively. Both ΔW_{up} and ΔW_{down} are then decomposed into TT cores. The process of applying a tensorized update in the fully connected layer, which performs a linear transformation on the input x, can be given by

$$\begin{aligned} y &= W_0 x + \text{TT}(\Delta W_{\text{up}}) \cdot \text{TT}(\Delta W_{\text{down}}) x \\ &= W_0 x + \prod_{k=1}^{d} \mathcal{G}_{p_k}^{(k)} \prod_{k=1}^{d} \mathcal{Q}_{p_k}^{(k)} x, \end{aligned} \tag{3}$$

where W_0 represents the pretrained weights, and ΔW_{up} and ΔW_{down} are represented as TT layers. The TT layers consist of TT cores $\mathcal{G}_{p_1}^{(1)}, \ldots, \mathcal{G}_{p_k}^{(k)} \ldots, \mathcal{G}_{p_d}^{(d)}$ and $\mathcal{Q}_{p_1}^{(1)}, \ldots, \mathcal{Q}_{p_k}^{(k)}, \ldots, \mathcal{Q}_{p_d}^{(d)}$.

As a result, the computational complexity can be reduced from $O(n^d)$ to $O(dnr^2)$. Since the TT-ranks are much smaller than n and d, TT decomposition significantly reduces the number of trainable parameters and saves memory usage.

C. Dual Quantization

The core idea is to apply dual quantization to the entire feedforward channel of the linear layers that require fine-tuning during the training process, with emphasis on the fact that weight quantization is integrated into training itself, rather than applied post-training. Before presenting the weight quantization process, we first explain Quant$(\cdot)$, the most commonly used quantization function in the DuQTTA framework. Quant$(\cdot)$ approximates a continuous value x to its nearest discrete counterpart, given by

$$\text{Quant}(x, \text{bits}) = \frac{\text{round}(x \cdot 2^{\text{bits}-1})}{2^{\text{bits}-1}}, \tag{4}$$

where round$(\cdot)$ is the function that rounds the value to the closest integer, and "bits" represents the target bit-width of x. The quantization of weights follows the same procedure throughout backpropagation. For clarity, we employ the same straight-through estimator (STE) approach during backpropagation, as used in most quantization studies [19–21].

For the weights of the linear layers that require fine-tuning, we convert their parameters from FP values to low-bit integers to reduce memory usage and computational cost, given by

$$\text{Quant}\mathcal{K}(x, \text{bits}) = \text{clip}\big(\text{Quant}(x, \text{bits}), -1 + \Delta,\ 1 - \Delta\big), \tag{5}$$

where $\Delta = 1/(2^{\text{bits}-1})$ represents the smallest interval between weight values, and clip$(\cdot)$ projects the data values onto a specified minimum and maximum range. The dual quantization strategy plays a pivotal role and is implemented in two stages, as illustrated in Fig. 2. In the first stage, the TT cores $\mathcal{G}_{p_1}^{(1)}, \ldots, \mathcal{G}_{p_k}^{(k)} \ldots, \mathcal{G}_{p_d}^{(d)}$, obtained through TT decomposition, are quantized based on their weights. This step converts floating-point numbers into low-bit-width integers (typically 8-bit), significantly reducing the memory usage and computational complexity, making it well-suited for resource-constrained hardware deployments. During the inference process, the quantized TT cores are first reconstructed into full linear layers represented in floating-point values. Since the reconstructed tensors are reverted to floating-point representations, a second stage of quantization is applied to compress these reconstructed linear layers back into low-bit-width representations. This additional quantization stage further enhances computational efficiency during inference. The process of dual quantization can be expressed as follows:

$$\begin{aligned} \mathcal{G}_{p_k}^{(k,q)} &= \text{Quant}(\mathcal{G}_{p_k}^{(k)}, 8), \quad k \in \{1, \ldots, d\}, \\ \mathcal{T}_1^q &= \text{Quant}\mathcal{K}(\mathcal{G}_{p_1}^{(1,q)} \times \mathcal{G}_{p_2}^{(2,q)}, 8), \\ \mathcal{T}_i^q &= \text{Quant}\mathcal{K}(\mathcal{T}_{i-1}^q \times \mathcal{G}_{p_{i+1}}^{(i+1,q)}, 8), \quad i \in \{2, \ldots, d-1\}, \end{aligned} \tag{6}$$

where Quant$(\cdot)$ represents the first stage of quantization, applied to the TT cores $\mathcal{G}_{p_k}^{(k,q)}$. Quant$\mathcal{K}(\cdot)$ represents the second stage of quantization, which is applied to the reconstructed TT cores, resulting in $\mathcal{T}_k^q$.

Dual quantization achieves both low memory usage and superior computational efficiency. The first stage focuses on compressing model storage, while the second stage further reduces the computational overhead associated with floating-point operations during inference. This combined dual quantization approach not only allows the model to maintain high accuracy while significantly lowering memory requirements but also greatly enhances inference speed. This dual quantization design, when combined with the TT decomposition approach, creates a fine-tuning strategy highly suitable for devices with limited memory or computational resources, positioning it as an important solution for efficient fine-tuning of LLMs.

D. Adaptive Optimization Strategy

As previously mentioned, we apply TT decomposition to reparameterize the update matrices of linear layers during fine-tuning, while employing dual quantization to further reduce memory usage and computational complexity. However, during forward propagation, the decomposed TT cores are gradually recombined through tensor products, ultimately reconstructing a matrix with the same shape as the original. Under conditions of large matrix width, using a uniform learning rate for these reconstructed matrices can result in suboptimal fine-tuning performance, limiting the model's ability to effectively capture and learn critical features. After TT decomposition, the original weight is represented as two key matrices, which are obtained by the sequential multiplication of multiple TT cores during forward propagation. Due to the differences in the dimensionality and function of these matrices in feature learning, AOS should be employed, rather than applying a uniform learning rate, with adjustments based on the network width n. The first matrix, $\Delta W'_{\text{up}} \in R^{n\times r}$, projects low-dimensional features into a higher-dimensional space. The second matrix, $\Delta W'_{\text{down}} \in R^{r\times n}$, remaps these features back to a lower-dimensional space. To ensure their contributions to the loss function are comparable, we enforce $\eta_{\Delta W'_{\text{up}}} \left\| \frac{\partial \mathcal{L}}{\partial \Delta W'_{\text{up}}} \right\|_1 \approx \eta_{\Delta W'_{\text{down}}} \left\| \frac{\partial \mathcal{L}}{\partial \Delta W'_{\text{down}}} \right\|_1$, where $\eta_{\Delta W'_{\text{up}}}$ and $\eta_{\Delta W'_{\text{down}}}$ are the learning rates for $\Delta W'_{\text{up}}$ and $\Delta W'_{\text{down}}$, respectively, and $\|\cdot\|_1$ denotes the L1 norm, defined as the sum of the absolute values of all components. Assuming the model output is given by $Y = X\Delta W'_{\text{up}}\Delta W'_{\text{down}}$, the gradients of the loss function with respect to $\Delta W'_{\text{up}}$ and $\Delta W'_{\text{down}}$ are given by

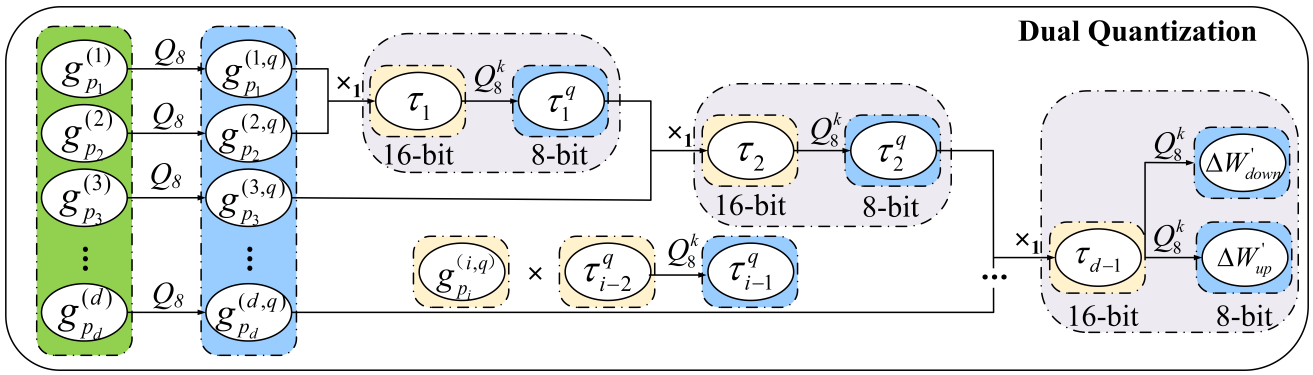


Fig. 2. The update process of TT decomposition based fine-tuning with dual quantization.

$$\begin{aligned}\frac{\partial L}{\partial \Delta W'_{\text{up}}} &= X^{\mathrm{T}}\frac{\partial L}{\partial Y}(\Delta W'_{\text{down}})^{\mathrm{T}},\\ \frac{\partial \mathcal{L}}{\partial \Delta W'_{\text{down}}} &= (\Delta W'_{\text{up}})^{\mathrm{T}}X^{\mathrm{T}}\frac{\partial L}{\partial Y},\end{aligned} \tag{7}$$

where L denotes the loss function, while $\Delta W'_{\text{up}}$ and $\Delta W'_{\text{down}}$ represent the two key matrices obtained after tensor reconstruction. The variable $X \in \mathbb{R}^{b\times n}$ denotes the input feature matrix, and $Y \in \mathbb{R}^{b\times n}$ represents the output feature matrix, with $\frac{\partial L}{\partial Y}$ indicating the gradient of the loss function with respect to the model output. To maintain numerical stability during forward propagation, the initial values of $\Delta W'_{\text{up}}$ and $\Delta W'_{\text{down}}$ are typically initialized based on their standard deviations, set as $\sqrt{\frac{1}{r}}$ and $\sqrt{\frac{1}{n}}$, respectively. This results in the relationship $\eta_{\Delta W'_{\text{up}}} \times \sqrt{\frac{1}{r}} \approx \eta_{\Delta W'_{\text{down}}} \times \sqrt{\frac{1}{n}}$.

From the preceding analysis, we deduce $\frac{\eta_{\Delta W'_{\text{down}}}}{\eta_{\Delta W'_{\text{up}}}} \approx O(\sqrt{n})$ when $r = O(1)$. Therefore, optimal performance can be achieved by setting the learning rate ratio between $\Delta W'_{\text{up}}$ and $\Delta W'_{\text{down}}$ to a fixed constant λ, with $\lambda = O(\sqrt{n})$ and significantly greater than 1. The optimal value of λ depends on the type of LLMs and the specific task at hand. The updates for the reconstructed tensors in AOS are given by

$$\begin{aligned}\Delta W'_{\text{down}} &\leftarrow \Delta W'_{\text{down}} - \eta \times G_{\Delta W'_{\text{down}}},\\ \Delta W'_{\text{up}} &\leftarrow \Delta W'_{\text{up}} - \lambda\eta \times G_{\Delta W'_{\text{up}}},\end{aligned} \tag{8}$$

where η denotes the learning rate, $G_{\Delta W'_{\text{down}}}$ and $G_{\Delta W'_{\text{up}}}$ represent the gradients. We have implemented AOS to ensure that the update pace of the two matrices aligns with their respective dimensional characteristics, as illustrated in Fig. 3. This approach significantly enhances LLMs fine-tuning performance while maintaining training stability and rapid convergence.

E. Magnitude-Direction Decoupled Update Mechanism

Due to the limitations of the fine-tuning approach mentioned above, both the tensor decomposition and reconstruction processes restrict parameter updates to a low-rank subspace. This constraint causes the changes in magnitude and direction to be highly coupled within this subspace, limiting their degrees of freedom. As a result, they tend to update proportionally, making it challenging to adjust direction changes with large or small magnitudes independently. In contrast, full-parameter fine-tuning allows for flexible and independent adjustment of both direction and magnitude. To align with the learning dynamics of full-parameter fine-tuning, we propose the magnitude-direction decoupled update mechanism. This mechanism decomposes the weight matrix W into a magnitude component and a direction component, where the magnitude component m is treated as a trainable parameter, while the direction component V remains fixed, allowing independent optimization of each component, given by

$$W = \text{broadcast}(m) \odot \frac{V}{\|V\|_c}, \tag{9}$$

where $W \in \mathbb{R}^{d\times k}$ is the weight matrix, $m \in \mathbb{R}^{1\times k}$ represents the magnitude component, and $V \in \mathbb{R}^{d\times k}$ denotes the direction component. The notation $\|\cdot\|_c$ indicates the column-wise norm of the matrix, applied to each column vector individually. Here, m is a row vector, and through *broadcasting*, each element m_j of m is applied to the corresponding column $V_{:,j}$ of V, where $j \in \{1, 2, \ldots, k\}$. In this column-wise scaling process, the direction of each column vector $V_{:,j}$ remains unchanged, while its magnitude is controlled by the corresponding element m_j in m. Specifically, *broadcast*(m) means that the vector m is broadcasted to match the shape of the matrix V, and $\odot$ denotes the element-wise multiplication, indicating that each element of m interacts with the corresponding element of V.

Specifically, the magnitude component is initialized by using the norm of the pre-trained weights, while the direction component is refined through TT decomposition, with the fine-tuning update process described as follows:

$$\begin{aligned}W' &= \text{broadcast}(m) \odot \frac{V + \Delta V}{\|V + \Delta V\|_c},\\ &= \text{broadcast}(m) \odot \frac{W_0 + \Delta W'_{\text{up}}\Delta W'_{\text{down}}}{\|W_0 + \Delta W'_{\text{up}}\Delta W'_{\text{down}}\|_c},\end{aligned} \tag{10}$$

where W' represents the updated weights after decoupling via the magnitude-direction components. Given that the direction component accounts for a substantial portion of the model's parameters, the use of TT decomposition significantly enhances computational efficiency.

IV. EXPERIMENTAL RESULTS

We conduct a comprehensive evaluation of the proposed DuQTTA method across various sizes of LLMs on downstream tasks. For LLaMA2-7B and LLaMA3-8B, we select seven datasets CB, BoolQ, WSC, COPA, ReCoRD, SQuAD, and DROP for assessing the performance of DuQTTA in three dimensions: (a) classification ability, (b) reasoning ability, and (c) generation ability. In addition, several PEFT methods are

also evaluated for comparison. Due to limited computational resources, we only report experimental results for LLaMA2-13B on inference tasks and generation tasks. The results indicate that the proposed DuQTTA outperforms existing PEFT methods, such as LoRA, Adapters, and Prefix Tuning, using fewer trainable parameters while achieving results comparable to full parameter fine-tuning. Further experiments confirm that DuQTTA reduces memory usage and power consumption while accelerating inference. We also perform ablation studies on tensor rank to identify optimal rank settings. The batch size ranges from 1 to 2, the learning rate from 5e-5 to 2e-4, and the number of epochs from 1 to 3, respectively. All experiments are conducted using the AdamW optimizer [22] on NVIDIA A100-80GB GPUs, with hyperparameters varying across tasks and models.

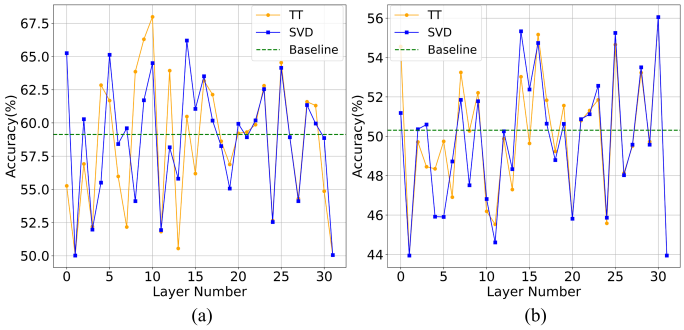


Fig. 4. (a) The comparison of layer-wise decomposition performance between TT and SVD on the FEVER dataset using LLaMA2-7B; (b) the same comparison on the TruthfulQA dataset.

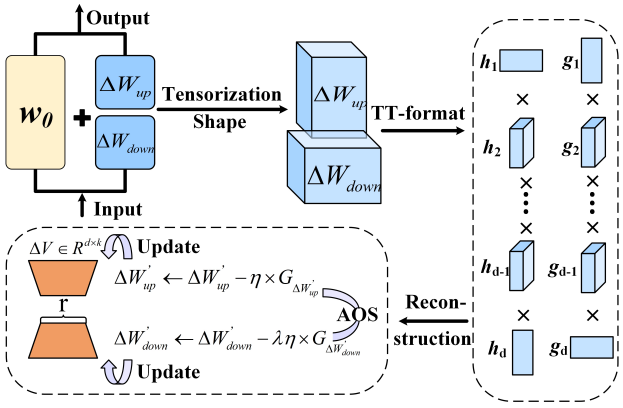


Fig. 3. The update process of TT decomposition based fine-tuning with adaptive optimization strategy.

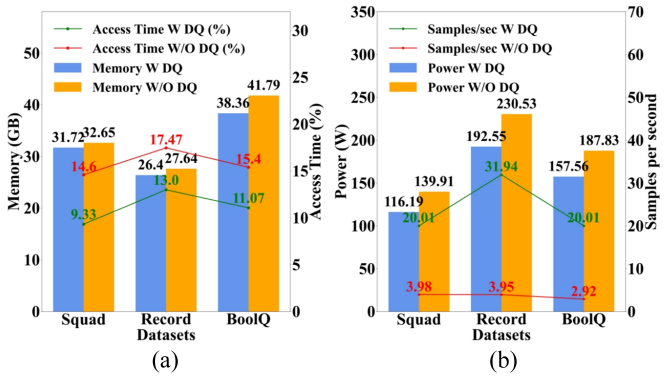


Fig. 5. (a) The comparison of memory usage and access time before and after implementing dual quantization; (b) the comparison of power consumption and inference rate.

A. *Low-Rank Analysis of LLMs*

Inspired by the Layer Selective Rank Reduction (LASER) approach [24], our work explores the intrinsic low-rank nature of LLMs. LASER demonstrated that applying Singular Value Decomposition (SVD) to selected layer weight matrices reduces over-parameterization, while maintaining or even improving the accuracy, thereby supporting the hypothesis of inherent low-rank structures within LLMs. However, LASER's focus on decomposing and approximating individual weight matrices fails to fully capture the potential tensor structures embedded within these matrices. This limitation suggests that LASER may not be optimally leveraging the inherent low-rank characteristics of the model architecture. To address this, our study explores the application of layer-wise TT decomposition. Our objective is to demonstrate that TT decomposition can more comprehensively exploit the intrinsic low-rank characteristics of the model, potentially leading to more efficient compression and improved performance. Fig. 4 presents a comparison between TT and SVD methods applied to MLP layers (gate_proj, up_proj, and down_proj) of the LLaMA2-7B model for FEVER and TruthfulQA tasks. The original MLP layers contain **202,375,168** parameters. SVD reduces this to 37,112,980, while TT decomposition further reduces it to a remarkable **223-fold** reduction with only **906,240** parameters. For the FEVER dataset, where the original model achieves an accuracy of 59.13%, TT decomposition applied to **50%** of the layers not only significantly reduces parameter count but also improves the accuracy. Similarly, for the TruthfulQA dataset, **44%** of the layers exhibit enhanced accuracy following TT decomposition, despite the substantial parameter reduction.

These experimental results indicate that LLMs are over-parameterized with inherent low-rank structures. Selectively applying SVD or TT decomposition reduces parameter count while enhancing model performance, with TT decomposition proving especially effective. Even with a **223-fold** parameter reduction, TT decomposition maintains accuracy comparable to the original model, striking an optimal balance between compression and performance. This makes TT decomposition particularly promising for compressing LLMs for deployment on resource-constrained edge devices, where efficient model performance is crucial.

B. *Comparisons with State-of-the-Art Methods*

Compared to other widely used PEFT methods, the proposed DuQTTA demonstrates superior performance. As shown in Table I, LoRETTArep achieves significant parameter reduction but exhibits lower accuracy. To evaluate the individual contributions of each strategy within DuQTTA, we conduct ablation experiments. The ablation results for the LLaMA2-7B model, as presented in the last three rows of the table, highlight the impact of AOS, DQ, and the magnitude-direction decoupled update mechanism applied to tensorized linear layers. It can be seen from Table I that AOS improves the average performance across seven public tasks by **2.99%** while maintaining a trainable parameter count that is only **one-eighth** of LoRA's. DQ significantly reduces memory usage and computational complexity, with only a minimal decrease in accuracy. Additionally, by using the magnitude-direction decoupled update mechanism, which allows for the

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT FINE-TUNING METHODS ON LLAMA2-7B AND LLAMA3-8B MODELS ACROSS VARIOUS BENCHMARKS.

Model	Method	Params	Classification			Multiple Choice		Generation		Avg
			CB	BoolQ	WSC	COPA	ReCoRD	SQuAD	DROP	
LLaMA2-7B	FT	6738.42M	66.07	84.6	63.46	86	81.1	90.71	51.38	74.76
	Adapter [7]	50.33M	66.07	71.8	62.50	84	78.8	88.45	49.14	71.54
	LoRA [2]	4.19M	67.86	84.8	62.50	81	79.4	90.56	45.96	73.15
	Prefix [23]	1.31M	51.78	78.6	61.53	83	81.0	90.56	45.95	70.35
	LoRETTArep [3]	0.51M	55.35	78.1	57.61	86	80.3	88.47	42.71	69.79
	TT&AOS	0.52M	69.64	85.6	63.46	86	79.9	90.96	51.79	75.34
	TT&AOS&DQ	0.52M	69.64	**87.1**	63.46	87	78.8	90.29	49.64	75.13
	DuQTTA	0.78M	**73.21**	86.5	**63.46**	**87**	**81.1**	**91.59**	**51.95**	**76.40**
LLaMA3-8B	LoRA [2]	3.41M	94.64	87.4	67.31	88	83.2	90.00	66.06	82.37
	LoRETTArep [3]	0.50M	73.21	83.0	66.35	90	**83.4**	91.08	66.92	79.14
	DuQTTA	0.67M	**96.43**	**89.1**	**69.23**	**93**	82.9	**93.51**	**70.58**	**84.96**

independent optimization of magnitude and direction components, DuQTTA can align with the learning dynamics of full-parameter fine-tuning, thereby achieving a **4.44%** increase in average accuracy compared to LoRA, with only a 0.26M increase in trainable parameters. Under comparable parameter conditions, DuQTTA also delivers a **9.47%** accuracy improvement over LoRETTArep, while achieving up to a **65**× compression rate compared to other PEFT methods.

DuQTTA continues to demonstrate superior performance in downstream tasks for LLaMA3-8B model. As shown in Table I, with a comparable number of parameters, DuQTTA significantly outperforms LoRETTArep by **7.35%**. Moreover, while DuQTTA uses only **one-fifth** of the trainable parameters of LoRA, it achieves a **3.14%** improvement in average accuracy across the seven tasks.

TABLE II
COMPARISONS OF FINE-TUNING METHODS ON LLAMA2-13B

Method	Params	COPA	ReCoRD	SQuAD	DROP	Avg
LoRA [2]	6.55M	90	83.4	92.7	59.1	81.3
LoRETTArep [3]	0.77M	86	84.4	90.9	53.2	78.6
DuQTTA	1.11M	**90**	**84.7**	**93.0**	**60.7**	**82.1**

Due to limited computational resources, we only report the results for inference and generation tasks using the LLaMA2-13B model. As shown in Table II, DuQTTA significantly outperforms LoRETTArep by 4.43% in accuracy, with a comparable number of parameters. Additionally, while DuQTTA uses only one-sixth of the parameters compared to LoRA, it achieves a 0.97% improvement in accuracy for average performance across both inference and generation tasks.

C. Ablation Study

Effectiveness of Dual Quantization. We conduct ablation studies on the SQuAD, ReCoRD, and BoolQ datasets using the LLaMA2-7B model to verify the effectiveness of DQ. As shown in Table I, the proposed DQ only has 0.28% accuracy drop. However, as illustrated in Fig. 5, this approach provides substantial benefits under identical hyperparameter settings, and it reduces memory usage by 0.93GB, 1.24GB, and **3.43GB** for SQuAD, ReCoRD, and BoolQ, respectively. Additionally, GPU access time is reduced by **5.27%**, 4.47%, and 4.33%, while power consumption decreases by 23.72W, 37.98W, and **30.27W**. Notably, inference speed improves by 5.03×, **8.08**×, and 7.94×. These experimental results demonstrate that, despite a slight accuracy loss, DQ offers significant improvements in GPU memory usage, GPU access time, power consumption, and inference speed. These advantages are particularly notable in resource-constrained edge devices, making DQ a suitable choice for meeting the lightweight requirements of edge computing.

TABLE III
PERFORMANCE ON BOOLQ WITH DIFFERENT RANKS FOR LLAMA2-7B.

Model & Method	Rank	Param	BoolQ
LLaMA2-7B (TT&AOS&DQ)	4	0.33M	83.6
LLaMA2-7B (TT&AOS&DQ)	8	0.52M	**87.1**
LLaMA2-7B (TT&AOS&DQ)	16	1.22M	86.2

Analysis about the Tensor-Train Rank. In this study, the impact of different tensor ranks on model performance is investigated. To isolate the effect of tensor rank, LLaMA2-7B is used as the baseline model and evaluated on the BoolQ dataset, with AOS applied on top of DQ and TT. Tensor ranks of 4, 8, and 16 are tested in separate experiments, making tensor rank the only differentiating factor. As shown in Table III, setting the tensor rank to 8 yields a significant performance improvement compared to a rank of 4, with only a slight increase in trainable parameters. Compared to a tensor rank of 16, a rank of 8 strikes a balance, offering superior performance with fewer trainable parameters. Therefore, a tensor rank of 8 emerges as the most effective choice in our experiments.

V. CONCLUSION

In this work, we propose a lightweight and efficient fine-tuning method named DuQTTA for LLMs. This method employs dual quantization and TT decomposition to effectively reduce memory usage and power consumption, while also accelerating inference with minimal accuracy loss. To address the suboptimal performance of LLMs under large-width conditions, we propose an adaptive optimization strategy that dynamically adjusts the learning rates of the reconstructed tensor. Additionally, we employ a magnitude-direction decoupled update mechanism that separates the update process of TT cores into magnitude and direction components, aligning with the learning dynamics of full-parameter fine-tuning. This further improves the adaptability and effectiveness of the model across various downstream tasks. For LLMs such as LLaMA2-7B, LLaMA3-8B, and LLaMA2-13B, the proposed DuQTTA method demonstrates superior performance over existing PEFT methods, with accuracy improvements of 4.44%, 3.14%, and 0.97% over LoRA, respectively. Furthermore, DuQTTA achieves up to a 65× compression rate compared to LLaMA2-7B, drastically reducing trainable parameters and resource consumption during fine-tuning, while still matching or surpassing the accuracy of full parameter fine-tuning. These advantages make DuQTTA particularly well-suited for efficient implementation on resource-constrained edge devices.

References

[1] N. Houlsby, A. Giurgiu, and e. a. Jastrzebski, "Parameter-efficient transfer learning for nlp," in *International Conference on Machine Learning*, 2019, pp. 2790–2799.

[2] E. J. Hu, yelong shen, and e. a. Phillip Wallis, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[3] Y. Yang, J. Zhou, and e. a. Wong, Ngai, "LoRETTA: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models," in *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, vol. 1, 2024, pp. 3161–3176.

[4] S. Hayou, N. Ghosh, and B. Yu, "LoRA+: Efficient low rank adaptation of large models," in *International Conference on Machine Learning*, 2024.

[5] J. He, C. Zhou, and e. a. Xuezhe Ma, "Towards a unified view of parameter-efficient transfer learning," in *International Conference on Learning Representations*, 2022.

[6] R. Karimi Mahabadi, S. Ruder, and e. a. Dehghani, Mostafa, "Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks," in *Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2021, pp. 565–576.

[7] R. K. mahabadi, J. Henderson, and S. Ruder, "Compacter: Efficient low-rank hypercomplex adapter layers," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[8] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 3045–3059.

[9] A. Razdaibiedina, Y. Mao, and e. a. Khabsa, Madian, "Residual prompt tuning: improving prompt tuning with residual reparameterization," in *Findings of the Association for Computational Linguistics*, 2023, pp. 6740–6757.

[10] Y. Wang, J. Wu, and e. a. Dabral, Tanmaya, "Non-intrusive adaptation: Input-centric parameter-efficient fine-tuning for versatile multimodal modeling," *ArXiv preprint arXiv:2310.12100*, 2023.

[11] T. Dettmers, A. Pagnoni, and e. a. Holtzman, Ari, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, p. 10088–10115, 2023.

[12] Y. Li, Y. Yu, and e. a. Chen Liang, "Loftq: LoRA-fine-tuning-aware quantization for large language models," in *International Conference on Learning Representations*, 2024.

[13] M. Valipour, M. Rezagholizadeh, and e. a. Kobyzev, Ivan, "DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation," in *Annual Conference of the European Chapter of the Association for Computational Linguistics*, 2023, pp. 3274–3287.

[14] Q. Zhang, M. Chen, and e. a. Alexander Bukharin, "Adaptive budget allocation for parameter-efficient fine-tuning," in *International Conference on Learning Representations*, 2023.

[15] D. J. Kopiczko, T. Blankevoort, and Y. M. Asano, "VeRA: Vector-based random matrix adaptation," in *International Conference on Learning Representations*, 2024.

[16] S. A. Koohpayegani, K. Navaneet, and e. a. Nooralinejad, Parsa, "Nola: Compressing lora using linear combination of random basis," in *International Conference on Learning Representations*, 2024.

[17] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, pp. 2295–2317, 2011.

[18] N. Lee and A. Cichocki, "Regularized computation of approximate pseudoinverse of large matrices using low-rank tensor train decompositions," *SIAM Journal on Matrix Analysis and Applications*, vol. 37, pp. 598–623, 2016.

[19] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in Neural Information Processing Systems*, vol. 28, pp. 3123–3131, 2015.

[20] S. Wu, G. Li, and e. a. Feng Chen, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations*, 2018.

[21] Y. Yang, L. Deng, and e. a. Wu, Shuang, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Networks*, vol. 125, pp. 70–82, 2020.

[22] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[23] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2021, pp. 4582–4597.

[24] P. Sharma, J. T. Ash, and D. Misra, "The truth is in there: Improving reasoning in language models with layer-selective rank reduction," in *International Conference on Learning Representations*, 2024.

ELF: Efficient Logic Synthesis by Pruning Redundancy in Refactoring

Dimitris Tsaras[†,‡], Xing Li[‡], Lei Chen[‡,*], Zhiyao Xie[†], Mingxuan Yuan[‡]
[†]The Hong Kong University of Science and Technology, Hong Kong
[‡]Noah's Ark Lab, Huawei, Hong Kong

***Abstract*—In electronic design automation, logic optimization operators play a crucial role in minimizing the gate count of logic circuits. However, their computation demands are high. Operators such as refactor conventionally form iterative cuts for each node, striving for a more compact representation - a task which often fails 98% on average. Prior research has sought to mitigate computational cost through parallelization. In contrast, our approach leverages a classifier to prune unsuccessful cuts preemptively, thus eliminating unnecessary resynthesis operations. Experiments on the refactor operator using the EPFL benchmark suite and 10 large industrial designs demonstrate that this technique can speedup logic optimization by $3.9\times$ on average compared with the state-of-the-art ABC implementation.**

***Index Terms*—AND-Invert Graph, Logic Synthesis, Logic Refactoring, Pruning, Machine Learning**

I. Introduction

In logic synthesis, Boolean resynthesis is a technology-independent process that optimizes a logic network to reduce its nodes and levels [1]–[7]. It analyzes the Boolean expressions and iteratively applies local transformations (e.g., operators) to the logic gates and their interconnections. By leveraging these transformations, Boolean resynthesis optimizes digital circuits for lower power consumption, delay, and area. While Boolean resynthesis operators are essential for circuit optimization, they must be used with caution due to their computational expenses.

Since most resynthesis operators are heuristic and suboptimal, iterative runs are necessary to identify further potential improvements in the logic circuit [1]. However, each successive iteration yields diminishing returns in terms of area improvement, while increasing the computational cost of logic synthesis. Moreover, the demand for a shorter turn-around time keeps increasing in the fast-paced electronics industry. By reducing the runtime of key optimization operators, designers can efficiently explore a wider range of design options, leading to better design solutions. Such an improvement of synthesis operators is design-agnostic, benefiting the synthesis process of almost all digital designs.

In recent years, machine learning (ML) has been increasingly leveraged to enhance logic synthesis. Neto et al. [8] employ ML to reinvigorate logic optimization algorithms and explore more advanced designs. Other recent research works utilize ML techniques for sequence tuning to improve the PPA [9]–[13]. However, none of the ML-driven solutions improve the efficiency of the operators, which are the runtime bottleneck for realistic industrial designs with millions of nodes. For example, executing the refactor operator on a synthetic circuit with 23 million nodes in the EPFL benchmark suite [14] can take up to 1.5 hours. Despite some recent efforts in GPU acceleration [15], [16], there might be scenarios with hardware constraints that do not support such parallelism. Also, such GPU parallelism is potentially compatible with other improvements in operators. To the best of our knowledge, no prior work has addressed the redundancy in refactor or proposed a method to prune the unnecessary iterations. In this work, we propose ELF, a novel and general method, which prunes the redundancy in logic optimization operators and significantly accelerates the refactor operator. Our contributions are summarized as follows:

- We present the redundancy problem in logic refactoring, and we provide a detailed examination of redundancy in both academic and industrial designs, which, on average, exhibit a failure to improve a node's cut by 98%.
- Then, we propose an efficient and accurate model to detect redundant operations during the refactoring process. Our model achieves an average accuracy of 87% and recall of 93% on academic circuits, and an average accuracy of 85% and recall of 95% on industrial designs.
- Finally, we propose ELF, an efficient logic refactor [2] operator by integrating the redundancy detection model in the synthesis process. Compared with the state-of-the-art original ABC [17] implementation, our method achieves an average speedup of $5.29\times$ with a minimal design quality loss of less than 0.27% on academic designs. Similarly, on industrial designs, our method achieves a speedup of $2.80\times$ with a negligible design quality loss of less than 0.08%.

The remainder of this paper is structured as follows. Background and related work are reviewed in Section II. Section III provides an in-depth discussion of our proposed technique. Experimental results are presented in Section IV, followed by concluding remarks and future directions in Section V.

II. Background & Related Work

A Boolean network is a directed acyclic graph (DAG) that maps an n-dimensional Boolean vector to an m-dimensional one, such that $\{0,1\}^n \rightarrow \{0,1\}^m$. The input and output vectors are known as primary inputs (PIs) and primary outputs (POs). And-Inverter Graphs (AIGs), containing only two-input AND gates and inverters, are commonly used to represent Boolean networks due to their simplicity and scalability [18].

Logic synthesis transformations can be classified into two categories: local operators, which are fast but achieve limited optimization, and global operators, which are slower but capable of more significant PPA improvements. Local operators only consider a node or small cut, resulting in a runtime linear to the cut size. In contrast, global resynthesis operators consider the entire graph when identifying optimization opportunities. Although global operators can substantially improve PPA, their prohibitive runtime makes them impractical for large industrial circuits.

We focus on the refactor operator, as its an essential component in industrial optimization flows and there has been little effort in literature to improve. While, refactor provides a lower area reduction relative to other operators, its usage is critical since its transformations unlock further optimizations in the AIG [2]. Nevertheless, its

*Corresponding author: Lei Chen (*lc.leichen@huawei.com*)

running time is significant. For example, in a popular logic synthesis flow such as Resyn2, the refactor step typically accounts for $20-40\%$ of the total execution time, despite being invoked only twice, whereas the balance and rewrite steps are invoked three and four times, respectively. Amaru et al. proposed a Boolean filtering for these operators to reduce the number of gates process without affecting the quality of result (QoR) [19]. However in their results they do not report the runtime for the operators, so we cannot evaluate the effectiveness of the method.

1) **Refactor** [2] iterates over all the nodes and forms a large cut for each one of them. Each cut is transformed into an SOP and factored into a more compact form, while maintaining the function of the root node. If there are more removed nodes than nodes added, the new resynthesized subgraph gets committed. As refactor has seen minimal follow-on work since its proposal, research opportunities likely remain.
2) **Rewrite** [1] greedily iterates over all nodes, forming all 4-input cuts. It substitutes each cut with a precomputed minimal subgraph selected from 222 NPN classes. The subgraph with the maximum non-negative gain, i.e. removed nodes minus added nodes, is committed. Recent extensions consider 5-input subgraphs [20], apply rewrite in a sliding window [7], and improvements in the efficiency by utilizing an engineered dynamic-scoring function [21].
3) **Resubstitution** [2] attempts to express the function of a node with other nodes present in the network called divisors. A transformation is accepted if the new implementation is better than the current node implementation using the immediate fanins. Resubstitution can be generalized to k-resubstitution. Let k be the exact number of nodes added and l the nodes within the MFFC of the node. If k-resubstitution is possible and $l > k$, the change is committed as the total number of nodes in the graph decreased. Since the graph is an AIG in our setting, the newly added nodes are two-input AND gates with potential inverters in the inputs or outputs.

III. Methodology

Section III-A presents a general perspective on logic optimization operators and in more detail the refactor operator, motivating the need for our pruned approach, ELF. Section III-B delves into the technical details of the proposed pruning method, ELF, as well as optimizations to enhance performance. Finally, Section III-C enumerates the features uses for the classifier along with their rationale.

A. Logic Optimization Operators & Motivation

There is a range of logic optimization operators such as rewrite, refactor, and resubstitution which differ in their local transformations but, their algorithmic frameworks share key similarities. Each operator iterates through nodes in the same sequence, applying a distinct function to each node. Algorithm 1 illustrates the general pipeline of logic optimization.

In Algorithm 1, the Boolean_Optimization function resynthesizes the cut with fewer nodes, but commits the change only if the gain, defined as removed nodes minus added nodes, is positive. After cut optimization, functionality must be preserved by adding the necessary nodes. However, in most iterations, the cut fails resynthesis, wasting computation time. Specifically in refactor on the arithmetic circuits in EPFL arithmetic benchmark, at most 7.34% of cuts are refactored. Similar trends hold for other operators, which can be even more time-intensive. Therefore, we propose ELF, an efficient learned classifier that given structural cut information, predicts whether a cut can get refactored. A key component behind ELF are the hand-crafted lightweight features that capture salient structural information to create compact cut representations.

Algorithm 1 General form of Logic Optimization

```
for each node in AIG do
    Apply Boolean_Optimization(node, AIG)
    Gain = nodes removed − nodes added
    if Gain > 0 then
        Commit change in AIG
```

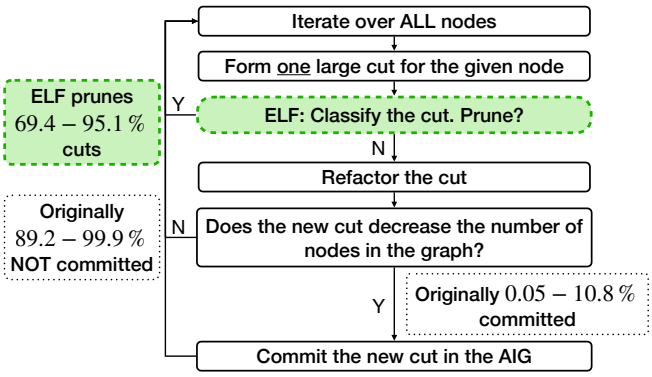


Fig. 1. Flow of ELF and the original refactor demonstrating the redundancy.

B. ELF: Pruning Cuts

As shown in the Algorithm 1, refactor iterates over all nodes in the And-Inverter Graph (AIG), forms a cut for each node and resynthesizes it. However, approximately $89.2-99.9\%$ of the formed cuts may fail to get optimized, wasting computational effort. We address this by classifying each node's benefit from resynthesis prior to function application. Specifically, we predict whether a given node is unlikely to be optimized, pruning it from the search space before proceeding. Only beneficial nodes undergo attempted resynthesis. Figure 1 demonstrates the flow of the original ABC implementation of refactor with its shortcomings and the green cells highlight how ELF changes the flow and prevents unnecessary computation. ELF can be very effective as it can omit to refactor $69.4-95.1\%$ of the nodes.

At the core of our method, ELF, is an efficient and compact classifier. While refactor is time-consuming on large industrial circuits, it operates efficiently on a per-cut basis. This characteristic necessitates rapid inference to effectively reduce overall execution time. The stringent time constraints inherently disqualify most advanced model architectures, such as graph neural networks, for this application. To illustrate, the inference time per cut using a small Graph Convolutional Network (GCN) is approximately 30 times that of resynthesizing the cut itself.

After extensive experimentation with various model architectures, including XGBoost trees and diverse ensemble models, we found that a feedforward neural network offers the optimal balance between execution speed and generalization to unseen circuits, while maintaining high recall and accuracy. The computational complexity of a single layer in a feedforward neural network is $\Theta(m*h+k*h)$, where m represents the number of inputs, k the number of outputs and h the number of hidden units.

Achieving high recall is crucial for our classifier, as our goal is to reduce runtime without compromising area optimization. Recall

Algorithm 2 ELF: Efficient Logic Refactoring

```
Collect features for each node's cut
Classify each node
for each node in AIG do
    if the node is classified to fail then
        Continue
    Refactor(node, AIG)
    Gain = nodes removed − nodes added
    if Gain > 0 then
        Commit change in AIG
```

quantifies the percentage of nodes correctly classified as 1. A recall of 100% ensures no degradation in the quality of logic optimization. This emphasis on recall aligns with our objective of enhancing computational efficiency while preserving the effectiveness of the refactor operator in logic synthesis workflows. Equally crucial is high accuracy, which directly correlates with runtime reduction. As we accurately prune more nodes, we expect a proportional decrease in runtime. The synergy between high recall and accuracy thus forms the cornerstone of our approach, enabling efficient circuit optimization without sacrificing quality.

To further decrease the inference time we use some engineering tricks such as kernel fusion and graph compilation for the model. Additionally, before node iteration, we batch all the cut data into one tensor so that we can maximize vectorization and minimize the process of packing/unpacking the data into tensors. Therefore, the complexity of classifying N nodes with a model that has complexity $\Theta(M)$ is $\Theta(NM)$. Despite concerns about diminished accuracy, experiments validate the viability of the batch classification approach. An intuitive explanation for this phenomenon is that the cuts at the start, are more likely to have available optimizations, which might be lost while refactoring the AIG. Although, we might unnecessarily resynthesize cuts that no longer can be optimized, this will only impose a degradation in the runtime, but not the area. The detailed implementation of ELF is outlined in Algorithm 2.

As seen in Algorithm 1 the logic optimization methods share an overall similar structure. Thus, it is possible to extend ELF to work for other operators other than refactor. The ultimate goal would be to have a unified logic optimization operator where the classifier conducts multiclass prediction to select the Boolean resynthesis function for each node, including skipping non-beneficial nodes. Thus, streamlining the entire process of logic synthesis.

C. Feature Selection

The features must add minimal runtime overhead while encapsulating salient information about the number of nodes to be removed and the logic sharing. Logic sharing refers to nodes that connect to nodes outside the cut. Meaning that their deletion is non-trivial as they affect the functionality of other parts of the circuit. A straightforward approach to represent the cut is to use the cut's root truth table. Although the truth table captures the functionality of the cut, it fails to encapsulate logic sharing. Additionally, large truth tables scale exponentially with the number of leaves, potentially incurring substantial computational costs. Logic optimization operators resynthesize first the cut as an isolated graph then, consider the impact of reinserting the optimized cut. If the nodes that need to be added to maintain the logic sharing exceed the number of nodes removed, the cut does not get committed. Thus, structural cut information can potentially capture both node deletion opportunities and logic sharing information.

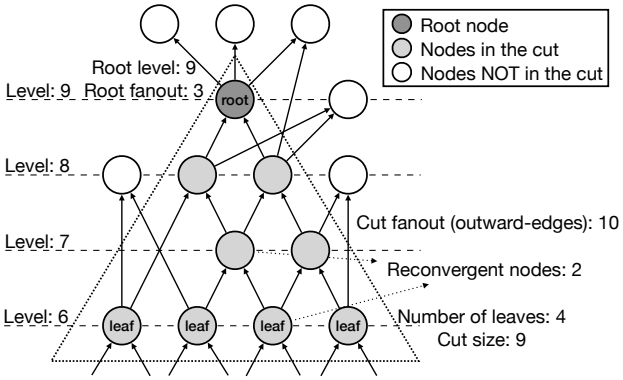


Fig. 2. Example of the features used to represent a cut in an AIG

We represent cuts by 6-dimensional feature vectors, comprising: *root fanout*, *root level*, *total cut fanout*, *cut size*, *number of reconvergent nodes*, and *number of leaf nodes*. Figure 2, contains an example of a cut with its features. In refactor, given a node it forms a reconvergence driven cut, where the given node is the cut's root node. The *root level* refers to the level of the given node within the AIG. The fanout refers to the outward going edges of a node. So, in Figure 2 the cut's *root level* is 9 and has 3 outgoing edges meaning the *root fanout* is 3. The triangle in the figure represents the cut and *cut size* refers to the number of nodes within it, which is 9 in this case. Similarly to the *root fanout*, the *total cut fanout* refers to the cut's total number of outgoing edges, which is 10. For a node to be a leaf there must be at least one path from a primary input (PI) that goes through the leaf and reaches the root of the cut without going through any other leaf. In Fig. 2, there are 4 *leaf nodes*. Finally, if there are two paths starting from a node v_s and ending to the node v_e they are denoted as reconvergent paths and v_s is a reconvergent node. Note that we only count the nodes that are part of reconvergent paths locally within the cut, for simplicity. In Fig. 2 there are only 2 locally *reconvergent nodes* marked with the arrows.

The rationale behind features related to the fanout are that they capture logic sharing as they indicate that they connect with nodes outside the cut. The bigger the fanout, the harder it is to replace the nodes within the cut. The *cut size* indicates the number of nodes to potentially prune, but its role can be to regularize other features. For example, the neural network might learn to divide *total cut fanout* by the *cut size*. The leaves cannot be simplified so a higher *number of leaves* might be less likely to be successfully optimized. The *root's level* implies the importance of a node. The higher level of a node, the more important it might be. Finally, in Theorem 1 in a recent paper by Riener et al. [7], they proved that if a certain type of optimization exists, then reconvergent nodes should exist. Thus, we include the number of *reconvergent nodes* as a feature.

These lightweight features are ideal for our problem as they are cheap to compute. We achieve efficient collection by accumulating the features during cut construction. Our flexible approach can run on CPUs or parallelized to run on GPUs.

IV. Experimental Results

In section IV-A, we introduce the implementation details and information about the datasets. Section IV-B compares the efficiency and effectiveness of the ELF over the baseline method, the original state-of-the-art refactor implementation of ABC, followed by Section IV-C, which presents the experiments on the classifier's quality metrics. Finally, Section IV-D contains explainability experiments

along with an analysis of the features used and their impact on the model's output.

A. Experimental Settings

We evaluate on the EPFL arithmetic benchmarks [14] and industrial designs. While the results are consistent for all the circuits, we show the results for the 6 largest EPFL circuits, and train the model on all the AIGs in the suite except the one we test on. Similarly for the industrial circuits, we train on all the other designs except the one we test on for 10 designs and report the average results. Note, we keep the two datasets separate for reproducibility. Table I and Table II contains statistics for the AIGs in the EPFL benchmarks and industrial designs, on which we report detailed results. The *Refactored* column the percentage represents the ratio of nodes that get resynthesized. It is evident that the datasets are highly imbalanced, since percentage of nodes that do get refactored are mostly around 1%. Although these circuits vary in size, demonstrating the scalability of our method, they are smaller than real industrial circuits, which often contain tens of millions of nodes and exhibit similar trends. Only three AIGs contain a relatively higher number of nodes that can get refactored, *sqrt* in the EPFL benchmarks and two industrial designs *design* 5 and 10, which contain 7.34%, 10.8%, and 4.53% nodes that can be resynthesized, respectively. This imbalance makes the learning significantly more difficult. Additionally, our classifier needs to be recall driven. Meaning that we do not want to misclassify cuts, since it will impact the quality of the operator. High accuracy is relatively easy to achieve, in the current setting, simply by classifying every cut as a 0. The difficulty in our task is to find a good balance between recall and accuracy. Thus, minimizing running time, with minimal impact on the number of AND gates.

To show that the models can generalize to unseen data, we train the classifier to all the arithmetic datasets other than the one we are testing on. Similarly for the industrial design circuits, none of the circuits that get tested are part of training. Additionally, each dataset is standardized individually with mean variance normalization. This way we can make the model infer to sizes that it might have not seen so far. The features we use to train the classifier with, can be calculated with simple accumulators while forming the cut with almost no additional runtime cost. This is necessary as refactor is already an efficient algorithm, leaving little running time available for inference. The features we use are *root fanout*, *root level*, *cut total fanout*, *cut size*, *number of reconvergent nodes*, and *number of leaf nodes*.

TABLE I
ARITHMETIC EPFL CIRCUITS STATISTICS

Design	And	Level	PIs	POs	Refactored
div	57247	4372	128	128	285 (0.50 %)
hyp	214335	24801	256	128	1992 (0.93 %)
log2	32060	444	32	32	530 (1.65 %)
multiplier	27062	274	128	128	247 (0.91 %)
sqrt	24618	5058	128	64	1806 (7.34 %)
square	18484	250	64	128	177 (0.96 %)

TABLE II
INDUSTRIAL CIRCUITS STATISTICS

Design	And	Level	PIs	POs	Refactored
design 1	384971	65	13135	13127	1142 (0.30 %)
design 2	267358	49	27800	20603	1184 (0.44 %)
design 3	628777	36	35552	34480	1569 (0.25 %)
design 4	159763	44	35784	34712	1273 (0.80 %)
design 5	428904	51	52344	51283	46376 (10.8 %)
design 6	507027	35	26292	25220	603 (0.12 %)
design 7	305218	72	20228	19148	839 (0.28 %)
design 8	77130	40	18357	18325	42 (0.05 %)
design 9	190600	71	26168	26139	807 (0.42 %)
design 10	423661	40	42257	33849	19180 (4.53 %)

The experiments were run on an Intel(R) Xeon(R) Platinum 8180M CPU @ 2.50GHz with 1 thread and 1TB of RAM. The models were trained using pytorch and the model was run directly in C++ within ABC [17] and executed with the ONNX Runtime framework. To efficiently standardize and execute the model we merged a Mean Variance Normalization node directly with the model and run all the cut data into one batch. While ML libraries are quite optimized, they have quite a bit of overhead when it comes to wrapping and unwrapping the data into tensors. Taking advantage of vectorization and limiting the overhead of the tensor objects, inference is very fast.

The model has 4 fully-connected layers with shape $6 \rightarrow 12 \rightarrow 12 \rightarrow 6 \rightarrow 1$ and a total of 325 parameters. Although more sophisticated architectures could potentially achieve better results, they are often prohibited by the tight time-budget. For instance, convolutional layers could be employed to enable the model to learn feature extraction autonomously, but this would require a GPU to fully leverage their capabilities. We initialize the parameters of each layer with Xavier initialization and the biases set to 0. For training we used a batch size of 64, 30 epochs up with early stopping and patience of 10 epochs. The optimizer was Adam with learning rate of 0.1 and a learning rate scheduler of cosine annealing with warm restarts. While we tried a range of losses such as focal loss, class balanced loss and class balanced focal loss, binary cross entropy worked the best. Additionally, to augment the data, we used MixUp [22] to improve generalization and overall model metrics. We tested more data augmentation techniques such as Synthetic Minority Over-sampling Technique [23] and One Sided Selection [24], but a weighted random sampler proved better. Furthermore, we tested dropout layers to potentially improve generalization, but it proved ineffective.

B. Comparison with Baseline

To demonstrate the effectiveness of ELF we compare it against the original implementation of refactor within ABC [17]. Additionally, we call the *rf -l* version of the operator. The quality metrics we report include the number of AND gates, levels, and their relative differences, the runtimes and the speedup of the proposed method. Let the relative difference be δ for a metric m_{ELF} for the proposed method, ELF, and m_{ABC} for the original implementation of refactor in ABC. Then, we measure the differences as $\delta = \frac{m_{\text{ELF}} - m_{\text{ABC}}}{m_{\text{ABC}}} \times 100\%$. From an industrial perspective, the area degradation should not exceed 0.5%, while achieving a $1.25\times$ speedup.

Table III presents the results of applying the original implementation of refactor operator within ABC vs this work, ELF, on the arithmetic circuits in the EPFL benchmark. The experiments show a significant runtime speedup varying from $2.50\times$ up to $7.69\times$. Simultaneously, there is minimal increase in the area. In the worst case, the number of nodes can increase by 0.27% for *div*, while the area difference is 0% for other circuits. The level is the same among all circuits. While, the runtime of the refactor operator is generally small for the circuits in the EPFL benchmark, industrial designs might contain several million nodes making the original implementation expensive i.e. several hours. We later present experiments on large synthetic AIGs.

The significant runtime speedup enables us to apply ELF twice without exceeding the execution of the original implementation of refactor in ABC. Thus, Table IV contains the results of chaining

TABLE III
REFACTOR'S PERFORMANCE IN ORIGINAL ABC VS ELF. EXPERIMENTS ON ARITHMETIC CIRCUITS IN THE EPFL BENCHMARK.

		Original ABC			ELF (this work)			Difference		
Design	**Nodes**	**Runtime** (s)	**And**	**Level**	**Runtime** (s)	**And**	**Level**	**Speedup**	**And**	**Level**
div	57247	1.00	56745	4372	0.21	56897	4372	4.76 ×	+0.27 %	0 %
hyp	214335	6.82	212341	24801	0.93	212341	24801	7.33 ×	0.00 %	0 %
log2	32060	0.71	31517	445	0.13	31564	445	5.46 ×	+0.15 %	0 %
multiplier	27062	0.77	26814	274	0.10	26815	274	7.69 ×	0.00 %	0 %
sqrt	24618	0.25	22811	5932	0.10	22872	5931	2.50 ×	+0.27 %	0 %
square	18484	0.52	18302	250	0.13	18314	250	4.00 ×	+0.07 %	0 %

TABLE IV
REFACTOR'S PERFORMANCE IN ORIGINAL ABC VS APPLYING ELF TWICE. EXPERIMENTS ON ARITHMETIC CIRCUITS IN THE EPFL BENCHMARK.

		Original ABC			ELF × 2 (this work applied twice)			Difference		
Design	**Nodes**	**Runtime** (s)	**And**	**Level**	**Runtime** (s)	**And**	**Level**	**Speedup**	**And**	**Level**
div	57247	1.00	56745	4372	0.43	56664	4372	2.32 ×	−0.14 %	0 %
hyp	214335	6.82	212341	24801	2.02	212296	24801	3.38 ×	−0.02 %	0 %
log2	32060	0.71	31517	445	0.53	31564	445	1.34 ×	+0.15 %	0 %
multiplier	27062	0.77	26814	274	0.35	26814	274	2.20 ×	0.00 %	0 %
sqrt	24618	0.25	22811	5932	0.17	22872	5931	1.47 ×	+0.27 %	0 %
square	18484	0.52	18302	250	0.27	18307	250	1.93 ×	+0.03 %	0 %

TABLE V
REFACTOR'S PERFORMANCE IN ORIGINAL ABC VS ELF. EXPERIMENTS ON INDUSTRIAL CIRCUITS.

		Original ABC			ELF (this work)			Difference		
Design	**Nodes**	**Runtime** (s)	**And**	**Level**	**Runtime** (s)	**And**	**Level**	**Speedup**	**And**	**Level**
design 1	384971	4.10	382941	54	1.32	383021	54	3.10 ×	+0.02 %	0.00 %
design 2	267358	2.95	265387	48	0.85	265582	49	3.47 ×	+0.07 %	+2.08 %
design 3	628777	6.35	627059	36	1.91	627059	36	3.32 ×	0.00 %	0.00 %
design 4	159763	1.63	157097	44	0.38	157228	44	4.29 ×	+0.08 %	0.00 %
design 5	428904	5.03	294992	46	2.17	295168	48	2.32 ×	+0.06 %	+4.35 %
design 6	507027	5.75	506416	36	2.32	506420	36	2.48 ×	0.00 %	0.00 %
design 7	305218	3.31	303609	64	1.48	303680	64	2.24 ×	+0.02 %	0.00 %
design 8	77130	0.82	77082	40	0.33	77082	40	2.48 ×	0.00 %	0.00 %
design 9	190600	1.75	189471	71	0.77	189520	68	2.27 ×	+0.03 %	−4.23 %
design 10	423661	4.93	381423	39	2.45	381433	39	2.01 ×	0.00 %	0.00 %

TABLE VI
REFACTOR'S PERFORMANCE IN ORIGINAL ABC VS ELF. EXPERIMENTS ON LARGE SYNTHETIC CIRCUITS.

Design	**Nodes**	**ABC Runtime** (s)	**ELF Speedup**	**And Diff.**
sixteen	16216836	2243.63	2.97 ×	+0.07 %
twenty	20732893	3138.46	2.87 ×	+0.06 %
twentythree	23339737	3914.77	2.85 ×	+0.06 %

ELF twice against the baseline. In the worst case, ELF maintained a $1.34\times$ speedup in runtime, but it failed to improve the quality further. Overall, a subsequent application of the proposed method can improve the results except in two circuits, log2 and sqrt. It is important to note that three of the circuits, multiplier, sqrt, and square, do not have any potential improvements in general when applying the original refactor operator twice. In contrast, for *div* and *hyp* by applying ELF twice we can still have a significant speedup in the running time and reduce the area of the AIG. Note that, these two graphs are the two largest arithmetic circuits in the EPFL benchmark.

Table V shows that ELF is equally effective in 10 industrial designs. These circuits are larger than the ones within the EPFL benchmark and vary from 77k-629k nodes. The trend from the results on the circuits in the EPFL benchmark continues to industrial design that are larger. The runtime speedup varies from $2.01\times$ to $4.29\times$, while the added number of nodes is minuscule. In the worst case the number of nodes increased by 0.08%, but was also as low as 0%. No information about these 10 circuits was part of the training data, which demonstrates that the classifier succeeded in generalizing. Thus, demonstrating the effectiveness of ELF and potential applicability to industrial solutions.

Finally, Table VI presents the condensed results of applying the original implementation of refactor operator within ABC vs this work, ELF, on synthetic circuits in the EPFL benchmark. Originally, the runtime of refactor can be longer than 1 hour for circuits such as the one with 23 million nodes, but ELF can theoretically speedup the operator by $2.90\times$ on average with little impact on the And nodes difference. To effectively observe these speedups in practice for large circuits, it is necessary to invest engineering efforts into improving the implementation of reinserting the refactored cut back into the AIG.

C. Model Quality Experiments

To effectively prune cuts our classifier needs to be recall-driven first and also achieve a high enough accuracy. Recall and accuracy are calculated as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}.$$

Recall directly affects the area, as it measures the proportion of nodes that were correctly classified as 1. Instead, accuracy is related to the overall runtime. The more accurate the model the more nodes the algorithm can skip. Thus, reducing the running time.

Table VII contains the model quality metrics of ELF on the arithmetic circuits in the EPFL benchmark. The classifier achieves a high recall and accuracy, while it was never trained on the test circuit

at all. Recall is consistently above 93% and up to 100% for some circuits, but it is only 76% for *div*. Finally, accuracy is overall high, varying from 77% up to 96%, even though the recall is also high. Therefore, this justifies the runtime reduction, while maintaining a low impact on the area, varying from 0% to +0.27%.

TABLE VII
MODEL QUALITY METRICS OF ELF ON THE ARITHMETIC CIRCUITS IN THE EPFL BENCHMARK.

Design	Recall	Accuracy	TP	TN	FP	FN
div	76 %	84 %	217	48045	8905	68
hyp	100 %	77 %	1992	162061	50282	0
log2	93 %	90 %	494	28295	3235	36
multiplier	100 %	96 %	246	25740	1075	1
sqrt	97 %	92 %	1745	20951	1860	61
square	94 %	84 %	167	15422	2885	10

Table VIII contains quality metrics of the ELF classifier. The results are consistent with the ones on the arithmetic circuits. Recall is high varying from 81% to 100% and so is accuracy 81 − 93%. We used the same classifier to test the quality on over 170 industrial designs. Recall is on average 91.8% and accuracy 81.0%.

TABLE VIII
MODEL METRICS OF ELF ON THE INDUSTRIAL CIRCUITS

Design	Recall	Accuracy	TP	TN	FP	FN
design 1	94 %	92 %	1069	351154	32477	73
design 2	81 %	85 %	958	226380	39711	226
design 3	100 %	93 %	1564	582528	44604	5
design 4	89 %	93 %	1132	147701	10780	141
design 5	100 %	81 %	46272	310728	70552	104
design 6	100 %	87 %	600	439699	66644	3
design 7	91 %	79 %	762	240122	63985	77
design 8	100 %	79 %	42	60857	16211	0
design 9	94 %	85 %	755	160845	28924	52
design 10	100 %	74 %	19144	293832	110537	36

D. Features Study

t-SNE plots [25] are a widely used technique for visualizing high-dimensional data in a two-dimensional space. Figure 3 presents such a visualization of our feature space, where orange points represent instances that underwent refactoring, and blue points indicate those that did not. The plot reveals the presence of distinct clusters, albeit dispersed across the space. This cluster distribution can be attributed to three key factors: the highly imbalanced nature of our dataset, the relatively low dimensionality of our feature space, and the discrete characteristics of our features. Despite these challenges, the discernible separation between clusters suggests that effective classification is feasible.

Figure 4 illustrates the SHAP [26] (SHapley Additive exPlanations) values for each feature in our model. SHAP values quantify the contribution of each input feature to the model's prediction, indicating both the magnitude and direction of influence. Positive SHAP values push the prediction towards 1 (indicating likely refactoring), while negative values push towards 0 (suggesting no refactoring). While SHAP values may not fully capture complex feature interactions, they provide valuable insights. For instance, the plot reveals that a low number of reconvergent nodes decreases the likelihood of refactoring, which aligns with the intuition that reconvergent nodes present optimization opportunities. Conversely, cuts with a high number of leaves, high root node level, or large cut size are less likely to be refactored. These features, while potentially not individually

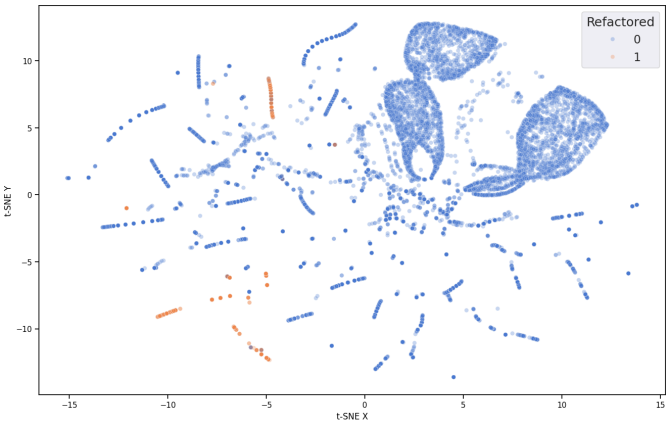


Fig. 3. t-SNE visualization of the feature space

explanatory, can collectively convey important information about the cut. For example, a high leaf count in a moderately sized cut might indicate lower node redundancy, as leaf nodes are irreplaceable. The cut fanout shows ambiguous SHAP values, but the difference between the cut's fanout and the root's fanout represents the number of nodes without external edges, which are crucial for maintaining AIG functionality and thus cannot be replaced.

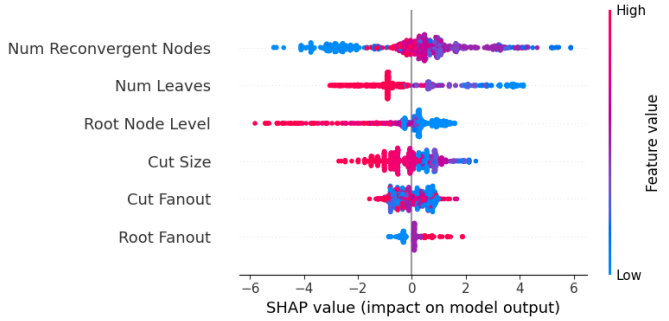


Fig. 4. SHAP (SHapley Additive exPlanations) values for each feature in the model

V. CONCLUSION

In summary we propose ELF a pruning strategy for the refactor operator that can speedup resynthesis by 3.9× on average with a minimal effect on the area. To achieve this we trained a compact model to classify, if the cuts will get successfully refactored. The features used to train the classifier are lightweight and ideal for time sensitive ML applications. An extensive experimental analysis on the EPFL benchmark suite and industrial circuits demonstrates the effectiveness in reducing the running time of the refactor operator. Given the similar structure of other logic synthesis operators such as rewrite and resubstituion, ELF can be adjusted to prune the unnecessary computation of those functions. A further extension of this work is to propose a unified logic optimization operation. Instead of having a sequence of operators, the classifier can choose the function to apply directly to the node or skip it. Such an operator could potentially significantly reduce the running time, achieve better PPA and eliminate the need to perform sequence tuning.

VI. ACKNOWLEDGEMENT

This work is partially supported by Hong Kong Research Grants Council (RGC) GRF Grant 16200724.

REFERENCES

[1] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: a fresh look at combinational logic synthesis," in *DAC*, 2006.

[2] A. Mishchenko, S. Cho, S. Chatterjee, and R. K. Brayton, "Combinational and sequential mapping with priority cuts," in *ICCAD, San Jose, CA, USA* (G. G. E. Gielen, ed.), IEEE, 2007.

[3] R. Brayton, J.-H. R. Jiang, and S. Jang, "Sat-based logic optimization and resynthesis," in *IWLS*, 2007.

[4] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *ICCAD*, 2015.

[5] M. Soeken, G. De Micheli, and A. Mishchenko, "Busy man's synthesis: Combinational delay optimization with sat," in *DATE*, IEEE, 2017.

[6] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. De Micheli, "Structural rewriting in xor-majority graphs," in *ASP-DAC*, 2019.

[7] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *ASP-DAC*, IEEE, 2022.

[8] W. L. Neto, M. T. Moreira, Y. Li, L. Amarù, C. Yu, and P.-E. Gaillardon, "Slap: A supervised learning approach for priority cuts technology mapping," in *DAC*, IEEE, 2021.

[9] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *DAC*, 2018.

[10] C. Yu, "Flowtune: Practical multi-armed bandits in boolean optimization," in *ICCAD*, 2020.

[11] A. Grosnit, C. Malherbe, R. Tutunov, X. Wan, J. Wang, and H. Ammar, "Boils:bayesian optimisation for logic synthesis," in *DATE*, 2022.

[12] X. Li, L. Chen, F. Yang, M. Yuan, H. Yan, and Y. Wan, "Himap: A heuristic and iterative logic synthesis approach," in *DAC*, 2022.

[13] C. Feng, W. Lyu, Z. Chen, J. Ye, M. Yuan, and J. Hao, "Batch sequential black-box optimization with embedding alignment cells for logic synthesis," in *ICCAD*, 2022.

[14] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *IWLS*, 2015.

[15] V. Possani, Y.-S. Lu, A. Mishchenko, K. Pingali, R. Ribas, and A. Reis, "Unlocking fine-grain parallelism for aig rewriting," in *ICCAD*, 2018.

[16] S. Lin, J. Liu, T. Liu, M. D. Wong, and E. F. Young, "Novelrewrite: node-level parallel aig rewriting," in *DAC*, 2022.

[17] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*, 2010.

[18] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Trans. Electron. Comput.*, 1963.

[19] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, J. Olson, R. Brayton, and G. De Micheli, "Improvements to boolean resynthesis," in *DATE*, 2018.

[20] N. Li and E. Dubrova, "Aig rewriting using 5-input cuts," in *ICCD*, 2011.

[21] X. Li, L. Chen, J. Zhang, S. Wen, W. Sheng, Y. Huang, and M. Yuan, "Effisyn: Efficient logic synthesis with dynamic scoring and pruning," in *ICCAD*, 2023.

[22] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2018.

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, 2002.

[24] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *ICML*, 1997.

[25] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, 2008.

[26] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *NIPS*, 2017.

On Bit-level Reverse Engineering of Vehicular CAN Bus

Yunlang Cai[1,2,#], Hanxue Shi[1,2,#], Xiaohang Wang[1,2,*], Haoting Shen[1,2,*], Li Lu[1,2], Kui Ren[1,2]
[1]*State Key Laboratory of Blockchain and Data Security, Zhejiang University, China*
[2]*Hangzhou High-Tech Zone (Binjiang), Institute of Blockchain and Data Security*
17620486393@163.com, sarahshi99@163.com, xiaohangwang@zju.edu.cn, htshen@zju.edu.cn, li.lu@zju.edu.cn, kuiren@zju.edu.cn

***Abstract*—The Controller Area Network (CAN) bus is a cornerstone of modern vehicles, orchestrating functions from engine control to auxiliary systems. However, its lack of inherent security measures makes it vulnerable to cyberattacks. Accurately mapping CAN signals with car-control actions is critical for detecting security breaches, as it allows pinpointing potential vulnerabilities exploited to compromise vehicular functions. Despite this, existing CAN reverse engineering methods struggle to achieve bit-level resolution due to the huge search space of unique IDs and payload combinations. To address this challenge, we propose a systematic framework for reverse engineering CAN bus messages, achieving precise mapping of control bits in CAN frames to car-control actions. The framework was validated on Tesla Model 3, Leapmotor C10 and C11, demonstrating its versatility across different vehicle platforms. In particular, it successfully identified 43 car-control actions on the Tesla Model 3, showcasing its extensive coverage. Furthermore, its low resource consumption enables seamless integration into compact platforms like the Raspberry Pi, supporting practical deployment in real-world automotive systems.**

Keywords—CAN Bus Reverse Engineering, Signal Boundary Identification, Fuzz Testing

I. INTRODUCTION

The Controller Area Network (CAN) bus is a cornerstone of modern vehicle communication, orchestrating functions from critical engine control to auxiliary systems like lighting and climate management. Its standardized design ensures reliability and efficiency, but also introduces a significant vulnerability: the lack of inherent security measures. As vehicles increasingly rely on software-driven control and connectivity features, this vulnerability leaves them exposed to a variety of cyber threats. Attackers could exploit CAN's openness to inject malicious messages, potentially compromising vehicle safety and user privacy.

Each automotive manufacturer (OEM) uses proprietary DBC files to define the meaning of CAN signals, obscuring the internal logic of its systems and masking unique security flaws. This fragmentation not only complicates cross-manufacturer diagnostics but also hinders the identification and mitigation of potential vulnerabilities. Reverse engineering of CAN bus messages is essential for overcoming these obstacles. By decoding the proprietary definitions in DBC files, reverse engineering enables a detailed interpretation of CAN signals, uncovering hidden security flaws and supporting the development of effective cybersecurity measures.

However, achieving fine-grained decoding of CAN signals faces significant challenges. As illustrated in Figure 1, data frames are continuously transmitted on the CAN bus, each containing an ID field and a payload field. The ID field determines the frame's priority, while the 64-bit payload field indicates the specific information carried by the frame. Within the payload, manufacturers predefine various fields to represent different command content or status information, as seen in the example fields in the figure. Given that a typical vehicle has hundreds of unique IDs and each payload can have up to 2^{64} possible combinations, the search space for identifying precise control bits of car-control actions is huge. This huge search space creates a major barrier to mapping individual bits to specific car-control actions, which is crucial for identifying potential vulnerabilities.

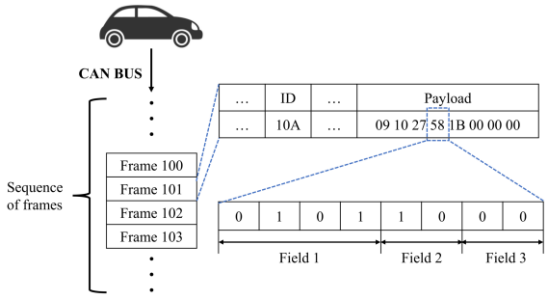


Fig. 1. CAN Bus Data Frame Layout and Search Space Complexity

Although there were many CAN bus signal reverse engineering methods proposed, they cannot address this huge search space problem in CAN signal reverse engineering. Signal boundary identification methods [7–12] delineate broader data field boundaries within CAN messages but do not establish relationships to specific car-control actions or achieve bit-level resolution. Machine learning approaches [13], [17], while helpful in identifying CAN message IDs related to actions, stop at the ID level and lack the capability to decode payload details. OBD-II protocol-based methods [14–16] rely on diagnostic data, which restricts their ability to handle non-diagnostic signals. Together, these limitations underscore the need for a high resolution (bit level) interpretation of the CAN bus, which is vital to pinpoint the car-control actions.

To address these limitations, we present a comprehensive solution designed to efficiently decode CAN signals with high coverage and precision. Our systematic framework for reverse engineering CAN bus messages effectively manages the challenge of huge search spaces while achieving bit-level resolution. By analyzing signal boundaries across different vehicle states, such as stationary or during specific car-control actions (e.g., activating windshield wipers), the framework filters out irrelevant IDs. Targeted fuzz testing further explores payload variations to trigger car-control actions, refining the resolution to the frame level. Finally, control bit analysis isolates the exact bits responsible for specific actions, completing the mapping with bit-level resolution. This streamlined, step-by-step approach not only overcomes technical obstacles but also ensures lightweight and adaptable deployment on compact platforms like the Raspberry Pi. Validated across multiple vehicle models, it proves highly

* Xiaohang Wang and Haoting Shen are the corresponding authors.
# The two authors contribute equally to this work and should be regarded as co-first authors.

This work was supported in part by the key R&D programme of Zhejiang Province No. 2024C01012, in part by the National Key Research and Development Program of China No. 2023YFB4404404, in part by the National Natural Science Foundation of China under Grants 92373205 and 62374146, in part by Ant Group Research Fund, in part by the Key Technologies R&D Program of Jiangsu (Prospective and Key Technologies for Industry) under Grant BE2023005-2, and in part by CIE-Smartchip research fund No. 2023-004.

practical and robust, making it a strong candidate for real-world CAN signal reverse engineering tasks.

The contributions of this paper are threefold:

- We introduce a systematic framework combining state-based signal boundary identification, targeted fuzz testing, and control bit analysis. This approach effectively narrows down the search space for relevant CAN signals, achieving precise bit-level resolution in mapping control bits to car-control actions. This not only overcomes the inherent complexity of CAN signal reverse engineering but also significantly improves accuracy compared to existing methods.
- The proposed framework is highly efficient, with low CPU and memory requirements, enabling seamless integration into compact devices like Raspberry Pi systems. We prototyped a wireless automatic reverse engineering tool (using Raspberry Pi). This design supports on-site and portable automotive testing.
- Our method has been validated on three distinct vehicle models, including Tesla Model 3 and Leapmotor C10/C11. It successfully identified 43 car-control actions in a single model Tesla Model 3, achieving extensive coverage with high adaptability. These findings highlight the framework's practicality and versatility across diverse automotive platforms.

II. Related Work

A. Signal Boundary Identification

Signal boundary identification methods [7–12] focus on delineating data field boundaries within CAN messages but vary in their resolution and scope. The READ framework [7] introduced the first automated approach using bit-flip rate analysis, establishing the foundation for subsequent advancements. Building on READ, LibreCAN [8] optimized boundary identification and linked car-control actions with CAN IDs, though it did not achieve bit-level resolution. Choi et al. [9] further refined boundary accuracy with statistical analysis of bit-flip time series but lacked action-specific mapping. Later methods, such as CA-CRE [10] and ACRE [11], employed classification and heuristics to improve format identification but remained limited in granularity. Karamolegkos et al. [12] extended this line of research to protocol-level reverse engineering but similarly fell short of identifying precise control bits.

B. Machine Learning-Based Approaches

Machine learning approaches assist in identifying CAN message IDs related to car-control actions but generally stop at the ID level, lacking the capability to decode payload details. Buscemi et al. [16] applied pattern recognition to identify ID-action relationships but do not decode payload bits. Young et al. [17] used unsupervised learning to classify CAN message IDs, improving functional ID recognition but without detailed payload analysis.

C. OBD-II Diagnostic Protocol-Based Methods

The OBD-II diagnostic protocol, widely used for vehicle diagnostics, provides access to real-time data from the electronic control unit (ECU). In OBD-II-based reverse engineering, Kang et al. [13] introduced an automated tool that correlates OBD-II responses with CAN traffic to identify basic control signals. Building on this, Blaauwendraad and Kieberl [14] applied statistical correlation methods to improve the accuracy of matching OBD-II data to CAN signals across different vehicles. Song and Kim [15] advanced this approach by mapping specific signal boundaries within CAN data to car-control actions, offering a more detailed view of CAN signal content. However, all these methods are fundamentally constrained by the OBD-II protocol's request-response structure, which limits reverse engineering to predefined Parameter Identifiers (PIDs). Since the ECU only responds to specific, and manufacturer-defined PIDs, the scope of reverse-engineered information is restricted to the data fields allowed by these definitions, excluding many non-diagnostic control signals crucial for comprehensive CAN analysis.

III. The Proposed Reverse Engineering Method

A. Overview

The proposed method uses both state-based signal boundary identification and targeted fuzz testing, systematically narrowing down the search space to locate control bits for specific car-control actions. The workflow begins with collecting CAN messages under stationary and operational states, where the stationary state provides baseline data and the operational state captures vehicle control actions such as door unlocking and window adjustments, as shown in Figure 2. After grouping messages by CAN ID, payload fields are extracted and organized into a matrix for further analysis (in section III. *B*).

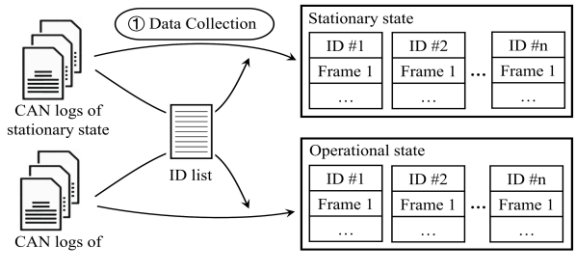


Fig. 2. State-Based CAN Message Collection and Preprocessing Workflow

Signal boundaries are identified using bit-flipping rate (BFR) analysis, which classifies fields into constant, counter, random, and co-flipping types (in section III. *C*). As illustrated in Figure 3, this classification highlights key areas for subsequent exploration and provides a foundation for related-ID locating. By comparing signal classifications across states, the related-ID locating step identifies IDs exhibiting state-dependent changes, capturing IDs most likely to be relevant for car-control actions (in section III.*C*.2).

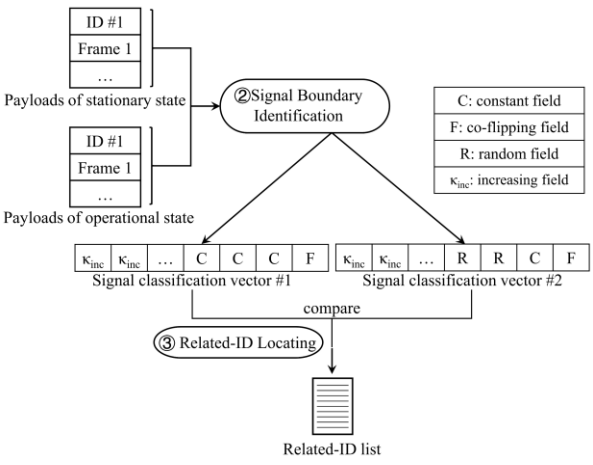


Fig. 3. Signal Boundary Identification and Related-ID Locating Workflow

Fuzzed data is then generated and injected into the CAN bus to provoke specific function controls or abnormal responses (in section III.*D*). Binary search techniques isolate the responsible frames and bits, enabling both validation and discovery of car-control actions, as shown in Figure 4. Finally, control bit analysis, comprising commonality analysis and selective bit testing, confirms the role of each individual bits in initiating specific actions (in section III.*E*). This approach achieves bit-level resolution, offering a significant advancement in reverse engineering techniques.

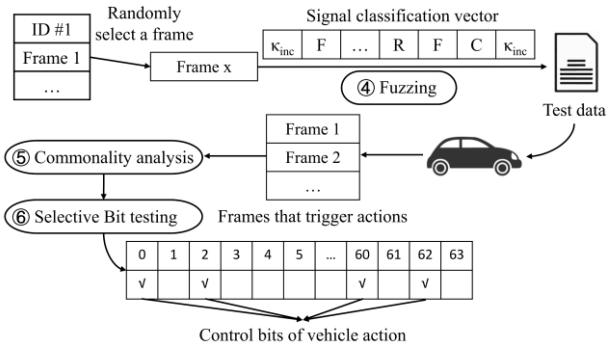


Fig. 4. Fuzz Testing and Control Bit Analysis Workflow

The notations used in this paper are listed in Table I.

TABLE I. NOTATIONS USED IN THIS PAPER

Notations	*Description*
$\boldsymbol{M_s}$	CAN data messages collected in the stationary state
$\boldsymbol{M_o}$	CAN data messages collected in the operational state
Ω_{op}	Set of specific car-control actions in the operational state, such as unlocking, wiper control
$\boldsymbol{P_{d,s/o}}$	A bit matrix representing the payloads of CAN ID d under stationary state (s) or operational state (o). $\boldsymbol{P_{d,s/o}}[i]$ is the i-th column of bits within $\boldsymbol{P_{d,s/o}}$ and $\boldsymbol{P_{d,s/o}}[i][j]$ is the i-th bit of the j-th frame within $\boldsymbol{P_{d,s/o}}$
$\boldsymbol{bfr_{d,s/o}}$	A bit-flipping rate vector for $\boldsymbol{P_{d,s/o}}$ and $\boldsymbol{bfr_{d,s/o}}[i]$ is the bit flipping rate for $\boldsymbol{P_{d,s/o}}[i]$ (The i-th column of bits within $\boldsymbol{P_{d,s/o}}$)
$\boldsymbol{\tau_{d,s/o}}$	A signal classification vector that maps each bit index i in the payload $\boldsymbol{P_{d,s/o}}$ to its types: constant (C), incrementing counter (κ_{inc}), decrementing counter (κ_{dec}), co-flipping (F_k), or random (R). The i-th element $\boldsymbol{\tau_{d,s/o}}[i]$ is the classification of the i-th bit in $\boldsymbol{P_{d,s/o}}$
$\Sigma_{related}$	Set of IDs with signal boundaries that differ between states, possibly associated with specific car-control actions
$\boldsymbol{P_{d,fuzzed}}$	A bit matrix of fuzzed payloads for CAN ID d generated through a mutation-based fuzzer. Each row $\boldsymbol{P_{d,fuzzed}}[i]$ represents a fuzzed payload as a 64-bit vector.
$\boldsymbol{M_{d,fuzzed}}$	Complete fuzzed data messages for CAN ID d, ready for injection
ω_i	A specific car-control action
Ω_{tr}	Set of all ω_i observed during the injection process
Φ_{d,ω_i}	Set of payload vectors for CAN ID d that independently trigger car-control action ω_i
$\boldsymbol{\chi_{d,\omega_i}}$	A marked vector produced by commonality analysis on Φ_{d,ω_i}.
$\boldsymbol{\bar{\chi}^{(i)}_{d,\omega i}}$	A test vector derived from $\boldsymbol{\chi_{d,\omega_i}}$ by flipping the i-th bit, while fixing all x-marked positions to 0. Used in selective bit testing.
$\boldsymbol{\mathcal{C}_{d,\omega_i}}$	Set of control bits for ω_i , representing the indices of bits in the payload of CAN ID d frames that directly control ω_i . This set is determined through selective bit testing.

B. Data Collection and Preprocessing

To gather the necessary data for reverse engineering, CAN messages are collected in two distinct states: a stationary state, denoted as $\boldsymbol{M_s}$, with no active car-control actions, and an operational state, denoted as $\boldsymbol{M_o}$, with car-control actions which is denoted as Ω_{op}, e.g., unlocking the car, controlling the windows, or activating the windshield wipers.

During the preprocessing phase, the collected CAN messages are grouped by CAN ID. For each CAN ID n, the payload, denoted as $\boldsymbol{P_{n,s}}$ or $\boldsymbol{P_{n,o}}$, is extracted under stationary or operational state, respectively. $\boldsymbol{P_{n,s/o}}$ is a bit matrix, where each row represents the 64-bit payload of a single frame, and each column corresponds to a specific bit position. The matrix has N rows, where N represents the number of frames with ID n in $\boldsymbol{M_{s/o}}$ ($\boldsymbol{M_s}$ or $\boldsymbol{M_o}$). In $\boldsymbol{P_{n,s/o}}$, the bit index i starts from 0 on the leftmost column, corresponding to the most significant bit (MSB), and increases to 63 on the rightmost column, corresponding to the least significant bit (LSB).

C. Bit-Flipping Analysis Model

This step leverages the bit-flipping rate (BFR) to classify and differentiate signal fields within CAN messages. For each ID n and its payload $\boldsymbol{P_{n,s/o}}$, the bit-flipping rate for $\boldsymbol{P_{n,s/o}}[i]$ (the i-th column of bits within $\boldsymbol{P_{n,s/o}}$) is denoted as $\boldsymbol{bfr_{n,s/o}}[i]$, representing the percentage of times bit i changes (flips) between consecutive data frames within $\boldsymbol{P_{d,s/o}}$. It is calculated as follows:

$$\boldsymbol{bfr_{n,s/o}}[i] = \frac{1}{N-1}\sum_{j=1}^{N-1} \mathbb{I}\left(\boldsymbol{P_{n,s/o}}[i][j] \neq \boldsymbol{P_{n,s/o}}[i][j+1]\right) \quad (1)$$

where:

- $\boldsymbol{N}$ is the total number of frames with ID n in $\boldsymbol{M_{s/o}}$,
- $\boldsymbol{P_{n,s/o}}[i][j]$ is the value of the i-th bit in the payload of the j-th frame for CAN ID n under stationary or operational states,
- $\mathbb{I}(\cdot)$ is an indicator function that equals 1 if the argument is true (indicating a flip between consecutive frames) and 0 otherwise.

The bit-flipping rate vector $\boldsymbol{bfr_{n,s/o}}$ summarizes the flipping probabilities for all bit indices within $\boldsymbol{P_{n,s/o}}$. A $\boldsymbol{bfr_{n,s/o}}[i]$ of 0% means the bits never changes within $\boldsymbol{P_{d,s/o}}[i]$, while 100% means the bit flips between every pair of consecutive frames. This bit-flipping rate is essential for identifying signal boundaries and categorizing fields based on their roles in car-control actions.

1) Signal Boundary Identification: To systematically classify and represent the fields within $\boldsymbol{P_{d,s/o}}$, a Signal Classification Vector $\boldsymbol{\tau_{d,s/o}}$ is defined, where each element $\boldsymbol{\tau_{d,s/o}}[i]$ represents the field type of the i-th bit in $\boldsymbol{P_{d,s/o}}$, categorized into the following types:

- C: Constant fields – Bits that remain unchanged across all frames.
- κ_{inc} and κ_{dec}: Incrementing and decrementing counter fields – Bits that exhibit structured sequential behavior.

- F_i: Co-flipping fields – Bits that flip simultaneously with other bits.
- R: Randomly changing fields – Bits that do not fit any other classification.

The classification of each bit is based on its behavior across frames in $\boldsymbol{P}_{d,s/o}$ determined by the following rules:

a) Constant Fields (C): If all bits in $\boldsymbol{P}_{d,s/o}[i]$ are identical, the i-th bit is classified as a constant bit, i.e. $\boldsymbol{\tau}_{d,s/o}[i]=C$.

b) Counter Fields (κ_{inc} and κ_{dec}): Consecutive bits $\boldsymbol{P}_{d,s/o}[i{\sim}j]$ form a counter field if they satisfy the following:

- The bit-flipping rate $\boldsymbol{bfr}_{n,s/o}[i]$ of each higher-order bit k ($i \le k < j$) is approximately half that of its adjacent lower-order bit k+1.
- During a higher-order bit flip: For incrementing counters (κ_{inc}): The lower-order bit flips from 1 to 0, indicating a carry. For decrementing counters (κ_{dec}): The lower-order bit flips from 0 to 1, indicating a borrow.

If these conditions hold, all bits in the range $[i, j]$ are classified as counter bits:

$$\boldsymbol{\tau}_{d,s/o}[k] = \kappa_{\text{inc}} \text{ or } \kappa_{\text{dec}}, \qquad \forall k \in [i,j] \tag{2}$$

c) Co-flipping Fields (F_i): Bits in $\boldsymbol{P}_{d,s/o}[i]$, $\boldsymbol{P}_{d,s/o}[j]$ and others form a co-flipping group if they:

- Have identical bit-flipping rates.
- Flip simultaneously across all frames in $\boldsymbol{P}_{d,s/o}$.

Each group is assigned an identifier F_i, where i is the lowest index within the group:

$$\boldsymbol{\tau}_{d,s/o}[k] = F_i, \qquad \forall k \in \{i, j, \dots\} \tag{3}$$

d) Randomly Changing Fields (R): Bits that do not fit any of the above categories are classified as randomly changing:

$$\boldsymbol{\tau}_{d,s/o}[i] = R \tag{4}$$

The following pseudocode summarizes the classification process:

Algorithm 1 Pseudo-Code of the Classification Step

```
For each bit position i in range(0, 63):
  If bfr_{d,s/o}[i] = 0:
    τ_{d,s/o}[i] ← C
  Else if bfr_{n,s/o}[i] ≈ 2 ×bfr_{n,s/o}[i+1] :
    If CheckIncrementingCounter(P_{d,s/o}, i):
      τ_{d,s/o}[i] ← κ_inc
    Else if CheckDecrementingCounter(P_{d,s/o}, i):
      τ_{d,s/o}[i] ← κ_dec
  Else if CoFlippingDetected(P_{d,s/o}, i):
    τ_{d,s/o}[i] ← F_k
  Else:
    τ_{d,s/o}[i] ← R
Return τ_{d,s/o}
```

For example, the classification result for CAN ID 273 in the stationary state ($\boldsymbol{\tau}_{273,s}$) is: $\boldsymbol{\tau}_{273,s}$ = [C, C, κ_{inc}, κ_{inc}, κ_{inc} , R, R, …]. This illustrates how each bit in $\boldsymbol{P}_{273,s}$ is assigned a type based on its observed behavior across frames.

2) Related-ID Locating: The signal classification vectors $\boldsymbol{\tau}_{d,s}$ and $\boldsymbol{\tau}_{d,o}$, representing the signal classifications for CAN ID d in stationary and operational states, are compared to determine whether d belongs to the related-ID set $\Sigma_{related}$. If there is any bit index i such that $\boldsymbol{\tau}_{d,s}[i] \neq \boldsymbol{\tau}_{d,o}[i]$, the ID is classified as related and included in $\Sigma_{related}$; otherwise, it is considered unrelated.

Figure 5 illustrates this process using $\boldsymbol{\tau}_{2C2,\,s}$ and $\boldsymbol{\tau}_{2C2,\,o}$ as an example. The vectors show differences in certain bit classifications, such as a transition from C to R, therefore $2C2 \in \Sigma_{related}$. In contrast, for $\boldsymbol{\tau}_{273,\,s}$ and $\boldsymbol{\tau}_{273,\,o}$, no differences are observed, so $273 \notin \Sigma_{related}$.

This direct comparison approach ensures that $\Sigma_{related}$ is constructed efficiently, capturing IDs most likely to be relevant for car-control actions.

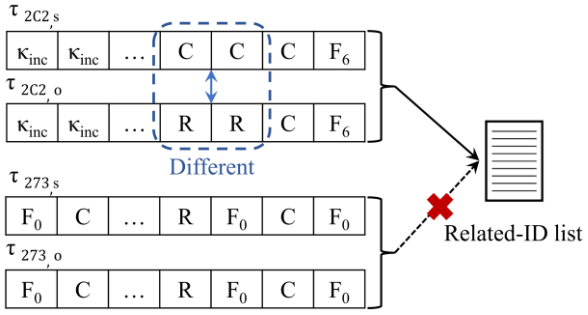


Fig. 5. Examples of Determining Related and Unrelated IDs Using Signal Classification Vectors

D. Fuzzed Data Generation and Injection

For each CAN ID d, fuzzed data generation begins by selecting an initial data frame from $\boldsymbol{P}_{d,o}$, representing the payloads collected in the operational state. This initial frame become the first row of the fuzzed data matrix, denoted as $\boldsymbol{P}_{d,fuzzed}[0]$. Each subsequent row $\boldsymbol{P}_{d,fuzzed}[n]$ ($n \ge 1$) is derived iteratively from the previous row $\boldsymbol{P}_{d,fuzzed}[n\text{-}1]$ by applying mutation rules based on $\boldsymbol{\tau}_{d,o}$.

Given the previous generated frame $\boldsymbol{P}_{d,fuzzed}[n\text{-}1]$, for each frame $\boldsymbol{P}_{d,fuzzed}[n]$, which represents the new frame to be generated based on the previously generated frame $\boldsymbol{P}_{d,fuzzed}[n\text{-}1]$, each bit i is mutated as follows:

If $\boldsymbol{\tau}_{d,o}[i] = C$, the i-th bit remains constant:

$$\boldsymbol{P}_{d,fuzzed}[n][i] = \boldsymbol{P}_{d,fuzzed}[n-1][i]. \tag{5}$$

If $\boldsymbol{\tau}_{d,o}[i] = R$, the i-th bit flips independently with a probability equal to $\boldsymbol{bfr}_{d,o}[i]$:

$$\boldsymbol{P}_{d,fuzzed}[n][i] = \begin{cases} 1 - \boldsymbol{P}_{d,fuzzed}[n-1][i], & \text{with probability } \boldsymbol{bfr}_{d,o}[i], \\ \boldsymbol{P}_{d,fuzzed}[n-1][i], & \text{otherwise.} \end{cases} \tag{6}$$

If $\boldsymbol{\tau}_{d,o}[i] = F_k$, all bits in the group F_k flip together, with a probability determined by $\boldsymbol{bfr}_{d,o}[k]$,consistent across the group:

$$\boldsymbol{P}_{d,fuzzed}[n][i] = \begin{cases} 1 - \boldsymbol{P}_{d,fuzzed}[n-1][i], & \text{if group flip occurs,} \\ \boldsymbol{P}_{d,fuzzed}[n-1][i], & \text{otherwise.} \end{cases} \tag{7}$$

If $\forall k \in [i, j]$, $\boldsymbol{\tau}_{d,o}[k] = \kappa_{inc}$ or κ_{dec}, the bits spanning $i \sim j$ are mutated according to the following rules. For the least significant bit ($k = j$), the flipping probability is determined by its bit-flipping rate $\boldsymbol{bfr}_{d,o}[j]$:

$$\boldsymbol{P}_{d,fuzzed}[n][j] = \begin{cases} 1 - \boldsymbol{P}_{d,fuzzed}[n-1][j], & \text{with probability } \boldsymbol{bfr}_{d,o}[j], \\ \boldsymbol{P}_{d,fuzzed}[n-1][j], & \text{otherwise.} \end{cases} \tag{8}$$

For higher-order bits ($i \le k \le j$), flipping is determined by whether a carry (for incrementing counters) or a borrow (for decrementing counters) propagates from lower-order bits:

$$\boldsymbol{P}_{d,fuzzed}[n][k] = \begin{cases} 1 - \boldsymbol{P}_{d,fuzzed}[n-1][k], & \text{if carry/borrow affects bit } k, \\ \boldsymbol{P}_{d,fuzzed}[n-1][k], & \text{otherwise.} \end{cases} \quad (9)$$

Using these mutation rules, the fuzzed data matrix $\boldsymbol{P}_{d,fuzzed}$ systematically explores variations in payloads for CAN ID *d*. After constructing $\boldsymbol{P}_{d,fuzzed}$, the payloads are combined with frame headers and footers to create the complete fuzzed dataset $\boldsymbol{M}_{d,\,fuzzed}$, which is injected into the vehicle's CAN bus.

If any $\boldsymbol{M}_{d,\,fuzzed}$ triggers a specific car-control action ω_i, binary search is performed to isolate the exact frames responsible. The payloads of these frames are recorded as $\Phi_{d,\omega i}$, an unordered set of payload vectors $\boldsymbol{P}$, an unordered set of payload vectors for CAN ID *d* that independently trigger ω_i.

The set of all triggered responses ω_i is denoted as Ω_{tr}, a set encompassing all car-control actions observed during the injection process.

E. Control Bit Analysis

This step identifies the exact bits in the payload that are responsible for triggering a specific car-control action ω_i. The analysis has two key steps: (1) commonality analysis, which isolates potentially relevant bits, and (2) selective bit testing, which refines these relevant bits to identify the exact control bits. Together, these steps form a precise methodology for identifying and verifying control bits for ω_i.

In the commonality analysis, $\Phi_{d,\omega i}$ is analyzed. Here, each vector $\boldsymbol{P}$ in $\Phi_{d,\omega i}$ represents the payload of a single CAN frame that triggered ω_i. The analysis produces a marked vector $\boldsymbol{\chi}_{d,\omega i}$, which captures the consistent and variable bit positions across all payloads $\boldsymbol{P}$. Each bit in $\boldsymbol{\chi}_{d,\omega i}$ is defined as:

$$\boldsymbol{\chi}_{d,\omega_i}[i] = \begin{cases} 1, & if\ \boldsymbol{P}[i] = 1,\ \forall \boldsymbol{P} \in \Phi_{d,\omega_i} \\ 0, & if\ \boldsymbol{P}[i] = 0,\ \forall \boldsymbol{P} \in \Phi_{d,\omega_i} \\ x, & if \oplus_{\boldsymbol{P} \in \Phi_{d,\omega_i}} \boldsymbol{P}[i] = 1. \end{cases} \quad (10)$$

Here, *x* indicates that the *i*-th bit varies across payloads, suggesting it is unrelated to ω_i. The marked vector $\boldsymbol{\chi}_{d,\omega i}$ eliminates irrelevant bits (*x*) and retains only those (0s and 1s) potentially related to ω_i.

In the selective bit testing, the marked vector $\boldsymbol{\chi}_{d,\omega i}$ is systematically tested to determine which bits directly influence ω_i. For each bit *i* marked 1or 0 in $\boldsymbol{\chi}_{d,\omega i}$, a test vector $\bar{\boldsymbol{\chi}}^{(i)}_{d,\omega i}$ is generated by flipping the *i*-th bit in $\boldsymbol{\chi}_{d,\omega i}$, while all *x* positions are set to 0. In total, *n* test vectors $\bar{\boldsymbol{\chi}}^{(i)}_{d,\omega i}$ are produced, where *n* is the number of 1s or 0s in $\boldsymbol{\chi}_{d,\omega i}$. Each test vector $\bar{\boldsymbol{\chi}}^{(i)}_{d,\omega i}$ is injected as a CAN frame, and its effect is observed. If the injection of $\bar{\boldsymbol{\chi}}^{(i)}_{d,\omega i}$ fails to trigger ω_i, the *i*-th bit is identified as a control bit. This implies that flipping this bit disrupts ω_i.

The final control bit set $C_{d,\omega i}$, storing the indices of all identified control bits, is defined as:

$$\mathcal{C}_{d,\omega_i} = \left\{ i \middle|\, \bar{\boldsymbol{\chi}}^{(i)}_{d,\omega i}\ fails\ to\ trigger\ \omega_i \right\} \quad (11)$$

Through these steps, the control bit analysis provides precise identification of the control bits for ω_i, enabling bit-level resolution reverse engineering of car-control actions.

IV. Performance Evaluation

In this section, we evaluate the performance of our CAN bus reverse engineering method. Our experiments were conducted on the Tesla Model 3 (2023) and the Leapmotor C11 and C10 (2024).

A. Experimental Setup

As described in section III. *B*, the data collection phase involves capturing CAN bus data ($\boldsymbol{M}_{s/o}$) in two distinct states: a stationary state and an operational state. During the operational state, a series of car-control actions (Ω_{op}, defined in section III. *B*) were manually triggered. These car-control actions are the actions that we aim to reverse engineer and they form the foundation for subsequent analysis.

In our experiment on Tesla Model 3, Ω_{op} included the following car-control actions, as shown in Table II:

TABLE II. Car-control Actions Triggered Manually (Ω_{op}) during Data Collection Phase

Window Up	Trunk Open	High Beam
Window Down	Trunk Close	Horn Activation
Multi-Level Wiper Control	Rearview Mirror Fold	Rearview Mirror Angle Adjustment
Left-turn Signal	Front Trunk Open	Door Lock
Right-turn Signal	Front Trunk Close	Door Unlock

B. Experimental Results

Through our methodology, we successfully obtained the control bits of a series of car-control actions, collectively defined as Ω_{tr}. The contents of Ω_{tr} are summarized in Table III. In our experiments, Ω_{op} consists of 8 manually triggered car-control actions, while Ω_{tr} includes a total of 43 actions, including both the predefined actions and additional function controls or abnormal responses discovered during the analysis.

TABLE III. Car-control Actions that Can Be Interpreted

Vehicle Function Controls	Abnormal Responses
Window Up/ Down	Window Sensors Deception
Front/Back Trunk Open/ Close	Front Seat Sensors Deception
Keyless Driving	Display Reboot
Turn Signals	Headlight Flicker
Multi-Level Wiper Control	Battery Warning Light
Long/ Short Horn Activation	Air Conditioning Shutdown
Silent Mode for Media	Climate Control Malfunction
Glove Box Open/ Close	Internal Circulation Off
High/ Low Beam	Defrost Function Error
Window Gap Open/ Close	Light Sensors Deception
Door Lock/ Unlock	Volume Display Flickering
Brake Light	Start Status Detection
Double-flash	Screen Display Error
Volume Control	A Tiny Slit of Window Landed
	Air Conditioning Airflow Malfunction

As described in section III.*B*, the data collection phase involves capturing CAN bus data ($\boldsymbol{M}_{s/o}$) in two distinct states: a stationary state and an operational state. During the operational state, a series of car-control actions (Ω_{op}, defined in section III.*B*) were manually triggered. These car-control actions are the actions that we aim to reverse engineer and they form the foundation for subsequent analysis.

The experimental results reveal several key insights:

1) Coverage: As shown in Table II and Table III, all 15 actions in Ω_{op} are fully contained within Ω_{tr}, confirming 100% coverage of the predefined car-control actions. This

demonstrates the effectiveness of our methodology in identifying control bits for these targeted actions.

2) Additional Car-control Actions: Beyond the 15 predefined actions, Ω_{tr} includes an additional 14 function controls and abnormal responses, bringing the total number of car-control actions to 43. These unexpected entries arise from the fuzzed data generation process, which efficiently produces plausible yet novel data frames beyond the original dataset. Such frames may activate related but untested actions or trigger conflicts within the ECU, resulting in abnormal responses. These anomalies are particularly significant as they reflect vulnerabilities in the ECU's processing logic, including insufficient error handling or flawed prioritization mechanisms.

C. Comparison with Existing Methods

To establish the efficiency and effectiveness of our method, we compared it against several existing approaches as shown in Table IV below.

TABLE IV. COMPARISON WITH EXISTING METHODS

Existing work	Experimental vehicles	Functions Controls	Abnormal Responses	Resolution
Fowler et al. [19]	1	×	×	Moderate (frame-level)
Zhang et al.[20]	1	×	×	Not mentioned
Jia et al. [21]	1	LED indicating car lock	√	Moderate (frame-level)
Fowler et al.[22]	×	Unlock the car door	√	High (bit-level)
Zixiang Bi et al. [23]	1	×	×	Low(ID-level)
Zixiang Bi et al. [24]	1	20 functions including lighting, wipers, etc.	×	High (bit-level)
Our work	3	43 actions and some high-risk actions involved (shown in Table III)	√	High (bit-level)

From Table IV, we can draw the following conclusions regarding our method's performance compared to others:

1) Car-control Actions Identification: Our method successfully reverse-engineered 43 car-control actions, achieving bit-level resolution for all identified actions. In contrast, most previous works, such as Fowler et al. [19, 22] and Jia et al. [21], primarily focused on frame-level or ID-level analysis, limiting their ability to identify precise control bits. Even Bi et al. [24], which achieved bit-level resolution, covered fewer actions than our method.

2) Unexpected Responses: Our approach extends beyond predefined car-control actions by efficiently uncovering additional related function controls and abnormal responses through fuzzed data generation. Previous works, such as Zhang et al. [20] and Bi et al. [24], did not explore unexpected responses, limiting their ability to detect unexpected ECU behaviors or potential conflicts.

3) Applicability Across Vehicles: Our method was validated on three real vehicle models, including the Tesla Model 3 and Leapmotor C11/C10, showcasing its robustness and generalizability. While some previous works, such as Fowler et al. [22] and Zhang et al. [20], were limited to a single vehicle, our broader applicability demonstrates its potential for diverse automotive platforms.

These results highlight the strengths of our methodology in achieving higher resolution and more extensive coverage, compared to existing approaches.

D. Overhead of Our Approach

The overhead of our approach includes a total time of 2 minutes for data collection, 30 seconds for data preprocessing, 3 seconds for bit-flipping analysis, 15 seconds for fuzzed data generation, and 20 minutes for control bit analysis (manual effort involved).

Resource utilization remains low, with an average CPU usage of 6% and memory consumption of approximately 73 MB, demonstrating the efficiency of the method on standard hardware.

E. Wireless Automatic Reverse Engineering Tool Development

We developed an automatic reverse engineering tool based on a Raspberry Pi 4b, equipped with a 2-CH CAN FD HAT for CAN communication. This tool is capable of receiving commands wirelessly from a laptop, reverse engineering, and performing car-control actions remotely via the CAN bus shown in Figure 6.



Fig. 6. The Wireless Automatic Reverse Engineering Tool Connected to Tesla Model 3in

The compact design and wireless capability make this tool portable and suitable for on-site experiments, enabling efficient execution of reverse-engineering tasks.

V. CONCLUSION

This paper presents a systematic framework for reverse engineering CAN bus messages, achieving bit-level resolution. The proposed method has been validated across multiple vehicle models, including the Tesla Model 3 and Leapmotor C10 and C11, demonstrating versatility across different vehicle platforms. On a single vehicle Tesla, it successfully identified 43 car-control actions, showcasing its extensive coverage of car-control actions. Furthermore, the method's low resource consumption allows seamless integration into compact platforms like the Raspberry Pi, facilitating practical deployment in real-world automotive systems. It establishes a practical and systematic method for precise bit-level identification.

REFERENCES

[1] Bozdal, Mehmet, et al. "Evaluation of can bus security challenges." Sensors, 2020, 2364

[2] Lee, H., Choi, K., Chung, K., Kim, J., & Yim, K. Fuzzing can packets into automobiles. IEEE Advanced Information Networking and Applications, 2015, 817-821.

[3] Greenberg, A. Hackers Remotely Kill a Jeep on the Highway—With Me in It. Wired, 2015. Available online: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[4] Weinmann, R.P.; Schmotzle, B. TBONE—A Zero-Click Exploit for Tesla MCUs. Kunnamon, 2021. Available online: https://kunnamon.io/tbone/tbone-v1.0-redacted.pdf.

[5] Ezeobi, Uchenna, et al. "Reverse engineering controller area network messages using unsupervised machine learning." IEEE Consumer Electronics Magazine, 2020, 50-56.

[6] Lee, H., Jeong, S. H., & Kim, H. K. OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame. Privacy, Security and Trust, 2017, 57-5709.

[7] Marchetti, M., & Stabili, D. READ: Reverse engineering of automotive data frames. IEEE Transactions on Information Forensics and Security, 2018, 1083-1097.

[8] Pesé, M. D., Stacer, T., Campos, C. A., Newberry, E., Chen, D., & Shin, K. G. LibreCAN: Automated CAN message translator. ACM Computer and Communications Security, 2019, 2283-2300.

[9] Choi, Wonsuk, et al. "An enhanced method for reverse engineering CAN data payload." IEEE Transactions on Vehicular Technology, 2021, 3371-3381.

[10] Ji, Cheongmin, Taehyoung Ko, and Manpyo Hong. "CA-CRE: Classification Algorithm-Based Controller Area Network Payload Format Reverse-Engineering Method." Electronics, 2021, 2442.

[11] Yang, K. ACRE: An Advanced CAN Signal Reverse Engineering Scheme. Consumer Electronics and Computer Engineering, 2023, 663-666.

[12] Karamolegkos, Michail. Automotive protocol reverse engineering & car Internal Network Reconnaissance. MS thesis. 2021.

[13] Kang, T. U., Song, H. M., Jeong, S., & Kim, H. K. Automated reverse engineering and attack for CAN using OBD-II. IEEE Vehicular Technology Conference, 2018, 1-7.

[14] Blaauwendraad, Bram, and Vincent Kieberl. "Automated reverse-engineering of CAN messages using OBD-II and correlation coefficients." 2020, Available online: https://rp.os3.nl/2019-2020/p103/report.pdf

[15] Song, Hyun Min, and Huy Kang Kim. "Discovering can specification using on-board diagnostics." IEEE Design & Test, 2020, 93-103.

[16] Buscemi, A., Castignani, G., Engel, T., & Turcanu, I. A data-driven minimal approach for CAN bus reverse engineering. IEEE Connected and Automated Vehicles Symposium, 2020, 1-5.

[17] Young, C., Svoboda, J., & Zambreno, J. Towards reverse engineering controller area network messages using machine learning. IEEE World Forum on Internet of Things, 2020, 1-6.

[18] Fowler, D. S., Bryans, J., Cheah, M., Wooderson, P., & Shaikh, S. A. A method for constructing automotive cybersecurity tests, a CAN fuzz testing example. IEEE Software Quality, Reliability and Security Companion, 2019, 1-8.

[19] Zhang, H., Huang, K., Wang, J., & Liu, Z. Can-ft: A fuzz testing method for automotive controller area network bus. Computer Information Science and Artificial Intelligence, 2021, 225-231.

[20] Jia, Y., Yang, J., Wang, S., Wang, Y., Chen, H., & Ma, Z. Algorithm Research on Fuzzy Test Method of CAN Bus Based on Field Weight. IEEE Power, Electronics and Computer Applications, 2022, 311-318.

[21] Fowler, D. S., Bryans, J., Shaikh, S. A., & Wooderson, P. Fuzz testing for automotive cyber-security. IEEE/IFIP Dependable Systems and Networks Workshops, 2018, 239-246.

[22] Bi, Zixiang, et al. "Bit-level automotive controller area network message reverse framework based on linear regression." Sensors, 2022, 981.

[23] Bi, Zixiang, et al. "A Method for Translating Automotive Body-Related CAN Messages Based on Labeled Bits." Applied Sciences, 2023, 1942.

MemSens: Significantly Reducing Memory Overhead in Adjoint Sensitivity Analysis Using Novel Error-Bounded Lossy Compression

Chenxi Li*, Yihang Feng*, Fuxing Deng*, Dingwen Tao†, Weifeng Liu* and Zhou Jin‡
*SSSLab, Dept. of CST, China University of Petroleum-Beijing, Beijing, China
† Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
‡College of Integrated Circuits, Zhejiang University, Hangzhou, China
Email: z.jin@zju.edu.cn

***Abstract*—Adjoint sensitivity analysis is an exceptionally efficient method for computing the gradient of an objective function with respect to given parameters, playing a crucial role in modern circuit design and verification. According to the principles of the adjoint method, it is necessary to store all essential system state information, such as state vectors and Jacobian matrices, at each time step during the forward integration process in order to construct the adjoint equations during the backward integration. Therefore, the memory overhead of the adjoint method is proportional to the system size and the number of time steps, resulting in prohibitive memory costs for solving large-scale dynamic systems.**

In this paper, we propose a novel, memory-efficient adjoint sensitivity analysis method that significantly reduces the memory overhead of storing system state information by employing error-bounded lossy compression techniques. Our compression algorithm effectively utilizes the spatiotemporal characteristics of data in circuit simulations and incorporates stringent error control mechanisms. This approach achieves a two-order-of-magnitude reduction in memory overhead during simulation while ensuring that the accuracy of the adjoint solution remains unaffected.

***Index Terms*—Lossy compression, Circuit simulation, Adjoint sensitivity analysis**

I. Introduction

Sensitivity analysis involves calculating the gradient of the objective function with respect to given parameters, serving as a powerful tool for evaluating how perturbations in system parameters impact system outputs. This method provides significant guidance in modern integrated circuit (IC) design and verification, making it widely applied in areas such as performance modeling [1], circuit optimization [2, 3], and yield estimation [4]. Sensitivity analysis encompasses two methods: the direct method and the adjoint method. The direct method is not suitable for dynamic systems with a large number of parameters [5], hence the adjoint method has become the standard in modern simulations.

A major drawback of the adjoint method is its high memory overhead [6]. During the forward solution process, it is necessary to store critical historical information at each time step, such as the state vectors, to construct differential equations during the adjoint process [7]. In large-scale simulations, this approach incurs substantial spatial and temporal costs. An alternative strategy involves storing additional state variables, such as the Jacobian matrix at each time step, to avoid recomputation during the reverse process and thereby reduce computational time. However, this approach further exacerbates the issue of memory overhead [8]. It is worth noting that during the simulation, Jacobian matrices can share indices [9], meaning that the primary memory overhead in adjoint sensitivity analysis stems from storing large-scale floating-point data.

One potential approach to reducing memory overhead during simulation is data compression. However, the mantissa of floating-point numbers often exhibits significant randomness, which limits the compression ratio achievable with traditional lossless compression algorithms [10], rendering them insufficient for the demands of large-scale simulations. Fortunately, advanced error-bounded lossy compression algorithms offer a promising solution to this challenge.

Error-bounded lossy compression, a current research hotspot in the field of high-performance computing (HPC), can significantly reduce data size while ensuring accuracy. It is widely regarded as the best solution for addressing the challenges posed by large-scale scientific data [11]. However, in circuit simulations, the complexity of circuit spatial topologies and the diversity of device parameters result in data exhibiting significant spikiness in the spatial dimension. Most existing compression algorithms heavily rely on data smoothness [12, 13], making them ineffective for handling circuit simulation data.

To address the low compression ratio of existing algorithms on circuit simulation data, we propose a novel error-bounded compression algorithm that efficiently compresses vector and matrix data in circuit simulations while ensuring high precision. This algorithm leverages the unique characteristics of circuit-related vectors and matrices to effectively smooth simulation data. It then applies a hybrid predictor based on the properties of data regions to decorrelate the data. Finally, a rigorous data quantization mechanism and an efficient encoding scheme are used to significantly reduce the size of simulation data while preserving simulation accuracy.

To the best of our knowledge, this is the first error-bounded compression algorithm capable of efficiently compressing both matrices and vectors in circuit simulations. Experiments demonstrate that the proposed compression algorithm can reduce simulation memory overhead by two orders of magnitude.

II. Background

A. Adjoint Sensitivity Analysis

In circuit design, sensitivity analysis is essential for evaluating the effects of parameter variations on performance. For sensitivity analysis, the circuit we are interested in can be represented by a set of differential algebraic equations (DAE) [14]:

$$\boldsymbol{g}(\boldsymbol{x}, t, \boldsymbol{p}) = \frac{d}{dt}\boldsymbol{q}(\boldsymbol{x}, \boldsymbol{p}) + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{p}) + \boldsymbol{b}(t, \boldsymbol{p}) = \boldsymbol{0} \quad (1)$$

where $\boldsymbol{x} \in \mathbb{R}^D$ is the state vector, t is time, $\boldsymbol{p} \in \mathbb{R}^M$ is the parameter vector with respect to which we require sensitivities, $\boldsymbol{q} \in \mathbb{R}^D$ and $\boldsymbol{f} \in \mathbb{R}^D$ represent the dynamic and static elements, respectively, $\boldsymbol{b} \in \mathbb{R}^D$ is the input. To obtain adjoint sensitivity, we take derivative of formula (1) with respect to p:

$$\frac{dO}{dp} = \sum_{i=0}^{n}\sum_{k=i}^{n}(\frac{\mathrm{d}\boldsymbol{O}}{\mathrm{d}\boldsymbol{x}})_k(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}})_k^{-1}\left(\prod_{j=k-1}^{i}(\frac{1}{h_t}\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{x}})_j(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}})_j^{-1}\right)(\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{p}})_i \quad (2)$$

This process is akin to the reverse of transient analysis, with the storage and reuse of matrices and vectors from the forward process significantly speeding up sensitivity analysis [15]. However, the requirement to preserve these matrices and vectors for numerical integration in the reverse process incurs substantial memory overhead. Thus, while sensitivity analysis is invaluable for design optimization, it also imposes significant computational resource demands.

B. Floating-Point Number

The IEEE 754 standard [16] defines floating-point (FP) data to be in 32-bit single-precision and 64-bit double-precision format. The single-precision includes one bit for the positive or negative sign, 8 bits for the exponent, and 23 bits for the mantissa. The double-precision format includes one sign bit, 11 exponent bits, and 52 mantissa bits, as demonstrated in Figure 1. The actual value of a floating-point data is represented as:

$$x = (-1)^{\text{sign}} \times 2^{(\text{exponent-bias})} \times 1.\text{mantissa}$$

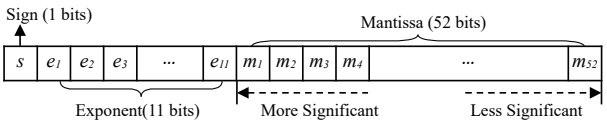


Fig. 1: IEEE 754 64-bit double-precision FP format.

C. Lossy Compression for Floating-Point Data

Error-bounded lossy compressors may have distinct designs and implementations. All existing error-bounded lossy compressors can be categorized into two groups: data-prediction based and domain-transform based.

Data-prediction-based compression model tries to predict each data point as accurately as possible based on its neighborhood in spatial or temporal dimension and then shrinks the data size by some coding algorithm, such as SZ [17] and FPZIP [18].

Domain-transform-based compression model needs to transform the original data to another non-orthogonal coefficient domain for decorrelation and then shrink the data size by applying some optimized coding algorithms such as embedded coding, e.g., ZFP [19].

However, all existing error-bounded lossy compressors are mainly designed to reduce storage overhead rather than memory overhead, considering error propagation. Only a few works [20–22] have utilized lossy techniques for memory compression, but these have not been optimized for the specific requirements of compression quality in circuit simulations.

III. Motivation

A. Lossy Compression Brings Higher Compression Ratio

Traditional lossless compression algorithms are less effective for domain-specific data due to low compression ratios. In contrast, lossy compression algorithms can achieve higher compression by allowing controlled information loss. As shown in Table I, lossy methods offer compression ratios 1 to 2 orders of magnitude higher than lossless ones [23]. However, some lossy algorithms, such as ZFP and FPZIP, do not adapt well to circuit simulation data, resulting in relatively low compression ratios.

TABLE I: Compression ratio comparison.

Dataset	Lossless Compression			Lossy Compression under 1×10^{-2} Relative Error			
	GZIP	ZSTD	BZIP2	ZFP [19]	FPZIP [18]	SZ1.4 [17]	SZ2.1 [17]
add20(vec)	5.23	6.03	4.33	8.52	189.8	1824.81	1783.85
add20(mat)	2.52	9.74	7.19	14.6	52.51	509.418	214.24
mem_plus(vec)	2.57	2.08	3.45	7.34	67.27	1933.368	1515.33

B. Existing Lossy Compression Algorithms Introduce Significant Errors in Simulation

Circuit simulation, particularly during transient analysis, requires high precision. While lossy compression algorithms like SZ improve compression ratios (see Table I), they introduce significant errors that accumulate over time, compromising simulation accuracy, as shown in Figure 2. ZFP controls errors better, but its compression ratio remains low. Moreover, the varying error levels across different lossy compression methods underscore the challenge of maintaining data integrity. Therefore, there is an urgent need to develop lossy compression algorithms that reduce data size while preserving the precision necessary for reliable simulations.

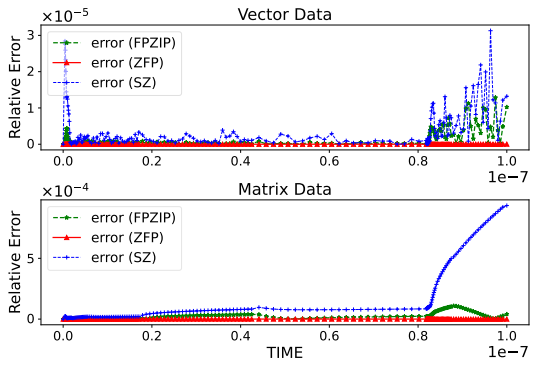


Fig. 2: The simulation error introduced by the compression algorithm under a relative error bound of 1×10^{-8}.

C. Existing Compressors Fail to Work Effectively on Circuit Simulation Data

As shown in Figure 3, the matrices and vectors generated by circuit simulation at each time step are spiky data with very low spatial autocorrelation. Existing compressors mostly rely on the smoothness of the data, so their performance will be greatly reduced on circuit simulation datasets. For example, ZFP, its domain transformation will lose effectiveness on simulation data, thus leading to low compression ratio. Due to the typically short time steps in numerical integration, data at adjacent time points exhibit strong correlations in the time dimension. This naturally suggests storing historical vectors or matrices as reference vectors or matrices to reduce data redundancy through prediction. However, this approach incurs significant memory overhead and struggles to handle dependencies in the data prediction process for lossy compression. Therefore, enhancing the smoothness of data in the spatial dimension is essential for simulation data compression.

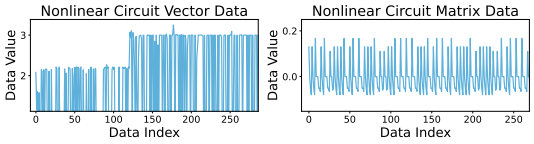


Fig. 3: The vector and matrix data of circuits exhibit pronounced spikiness.

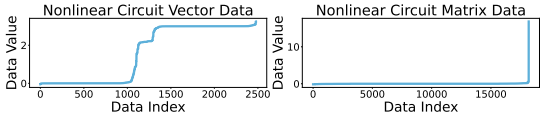


Fig. 4: After applying the reference sorting smoothing technique, the data exhibits strong autocorrelation.

IV. Error-Bounded Lossy Compression for Simulation

The proposed error-bounded compression algorithm efficiently compresses vectors and matrices during simulation with minimal overhead while ensuring precise floating-point error control. It effectively resolves the memory overhead challenges of traditional adjoint methods. As shown in Figure 5, the workflow consists of three main steps: (1) efficiently smooth the simulation data using **reference sorting**; (2) apply **RLE-FP** or **polynomial interpolation** predictors to decorrelate data based on regional characteristics; and (3) **quantize floating-point data** within a user specified error bound and encode it, achieving a significant data reduction.

A. Data Smoothing via Reference Sorting

Smoothing spiky data is essential for simulation data compression. Traditional smoothing techniques, such as wavelet transforms, are unsuitable due to their high time complexity [24]. To address this, we propose reference sorting, an efficient smoothing technique tailored for circuit simulation data. It consists of three main steps: (1) sort the floating-point queue in an ascending order at the initial time step, record the sorting indices, and use this sorting as a reference; (2) reorder the data in subsequent time steps based on the reference sorting; and (3) for longer integration processes, set validity check points to reinsert invalid data points into the sorted queue and update the reference sorting.

This approach is based on three key observations: (1) In circuits, nodes with large admittance (or small impedance) connections typically have identical or similar potentials [25]. Sorting the state vector groups these corresponding floating-point values together. This property is strongly linked to the circuit topology and device parameters, ensuring the reference sorting remains highly effective. (2) For matrices, elements with similar parameters contribute stamping values that are also similar [9], and sorting naturally clusters these corresponding floating-point values. (3) In time series data, values at adjacent time points are often similar or identical [26], leading to overall similarity in sorting across time steps.

Figure 4 illustrates the vector and matrix data of the nonlinear circuit at the 200th time step after applying the proposed smoothing technique. It can be observed that the data exhibits strong autocorrelation, which is highly advantageous for compression algorithms. The time overhead of reference sorting primarily arises from the sorting operation at the initial time step and the subsequent correction of a small number of out-of-order data points. The space overhead stems from storing the integer indices of a single time step, which is highly compressible and can be further reduced using integer compression algorithms. These time and space overheads are negligible compared to the original simulation costs.

B. Decorrelation by Hybrid Predictors

After data smoothing, we apply a hybrid predictor to decorrelate the data, significantly reducing its size. We observe that the smoothed data contains both numerically large steady regions and regions with small rapid numerical growth. To fully exploit the characteristics of these regional data patterns, we organize the entire dataset into different regions (or blocks) and apply distinct compression prediction techniques based on the data properties within each region.

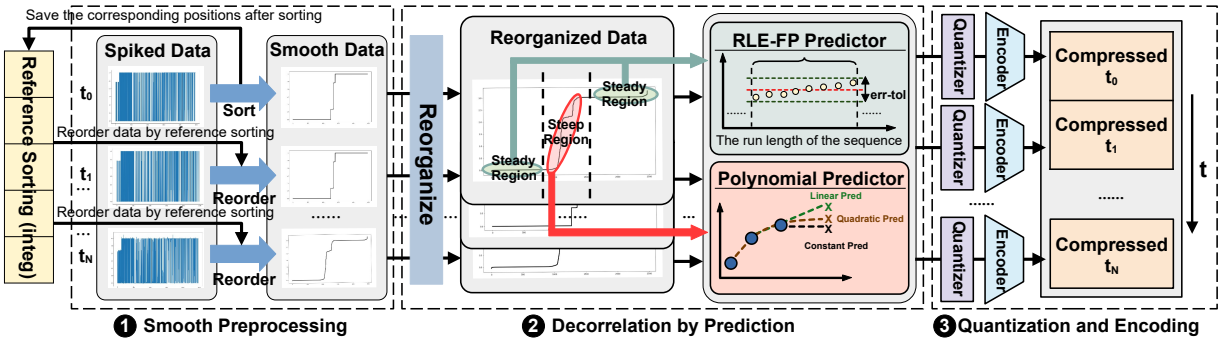


Fig. 5: The overall framework for error-bounded lossy compression of simulation data.

As shown in Figure 6, in the steady region, we introduce the RLE-FP technique for efficient decorrelation of the data. In the regions with rapid numerical growth, termed steep regions, we employ a prediction technique based on polynomial interpolation. This approach takes advantage of the varying data characteristics across different regions, optimizing the compression process.

We organize the sorted dataset into N regions. If the value at the right end of a region satisfies (3), we classify it as a steady region; otherwise, it is considered a steep region.

$$V_{right} < n\frac{V_{max} - V_{min}}{N} + V_{min} \tag{3}$$

In the text, V_{max} and V_{min} represent the maximum and minimum values in the dataset, respectively. n and N represent the total number of regions and the current region number from left to right, respectively. V_{left} represents the value at the leftmost end of the current region.

1) RLE-FP: The main idea of "RLE-FP" is to record consecutive values within the same error range in a floating-point sequence in order to achieve sequence compression.

Specifically, starting from the first floating-point number V_l at the left end of the steady region, we scan the floating-point sequence to identify the smallest $\hat{V}_m$ that satisfies $\hat{V}_m - V_l < e$ (where e is the user-specified error bound) and record this value as the center of the interval. From $\hat{V}_m$, we continue scanning to find the largest V_r such that $V_r - \hat{V}_m < e$, marking the right boundary of the interval. In other words, all floating-point numbers within this interval lie within a range centered at $\hat{V}_m$ with a radius of e. The process stops when a value outside the error range is encountered, and the length of the interval and $\hat{V}_m$ is recorded.

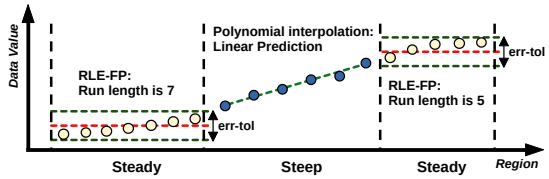


Fig. 6: Hybrid Predictors: RLE-FP for steady region and polynomial interpolation for steep region.

2) Polynomial Interpolation Prediction: The polynomial interpolation predictor is a predictor based on polynomial functions, which predicts future values by fitting historical data points. The current predicted value can be determined by the predicted values of the past n data points:

$$\hat{V}_n = \sum_{i=1}^{n} \alpha_{n-i} \hat{V}_{n-i} \tag{4}$$

To solve for the coefficient α, we substitute the historical predicted values into equation (4), yielding:

$$\hat{V}_n = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} \hat{V}_{n-i} \tag{5}$$

From equation (5), we can obtain the predicted value for each point, thereby achieving decorrelation in the steep regions. It is worth noting that using a polynomial interpolation predictor of order zero to two already provides sufficiently accurate predictions.

C. Error-Controlled Quantization and Data Encoding

A quantizer is responsible for quantizing the error between predicted values and true values, as well as handling unpredictable values. It significantly reduces the entropy of the original data while respecting the user-specified error boundaries.

We provide both absolute error bounds (denoted as ϵ) and relative error bounds (denoted as δ). To ensure simulation accuracy, both of these error bounds must be satisfied simultaneously. Therefore, there is a requirement between the true value V and the predicted value $\hat{V}$ (which is also the decompressed value):

$$\left|V - \hat{V}\right| \leq \epsilon \quad and \quad \frac{\left|V - \hat{V}\right|}{|V|} \leq \delta \tag{6}$$

So the final error bound e specified by the user is:

$$e = min(\epsilon, \delta|V|) \tag{7}$$

Clearly, when $|V| < \frac{\epsilon}{\delta}$, we focus solely on relative error; in contrast, we pay more attention to absolute error.

The quantizer calculates the error between the predicted value and the true value. If the result falls within the error bound e, the quantizer represents the true value with the predicted value. Otherwise, we refer to this true value as unpredictable.

For unpredictable values, we perform an XOR operation with their neighboring unpredictable values, use 2 bits to record the number of leading zero bytes in the residual, and finally truncate the insignificant bit planes based on the user-specified error bounds.

By analyzing the binary representation of floating-point numbers, it can be concluded that to satisfy absolute error bounds, the minimum number of mantissa bits required is:

$$R_{abs} = \begin{cases} 0, & p(V) - p(\epsilon) < 0 \\ 52, & p(V) - p(\epsilon) > 52 \\ p(V) - p(\epsilon), & otherwise \end{cases} \tag{8}$$

Algorithm 1: Compressing Simulation Data at t_n.

Input: Floating-point array of a matrix or vector $\{V_1, V_2, \ldots, V_M\}$ at t_n, user-specified error bound e
Output: compressed data stream in form of bytes

if $t_n = start_time$ **then**
  Sort the floating-point array.
  Save the sorting indices as reference sorting.
else
  if $t_n = checking_time$ **then**
    Check and maintain the validity of the reference sorting.
  end if
  Reorder the floating-point array by shared sorting.
end if
Organize data based on the regional characteristics of the sorted dataset.
for $(V_i,\ i = 1, 2 \cdots M)$ **do**
  if V_i is in a steady region. **then**
    if V_i is within the tolerance range. **then**
      $++run_length$
    else
      Save run_length and predicted value.
    end if
  else if V_i is in a steep region. **then**
    Determine the predicted value $\hat{V}_i$ using a polynomial predictor.
    Predict V_i using the $\hat{V}_i$.
  end if
  Quantize the unpredictable value and save the residual.
end for

Here, p represents the exponent of the floating-point number, and 52 denotes the number of mantissa bits for double-precision floating-points (23 for single-precision). To meet the absolute error bounds, larger floating-points require retaining more mantissa bits. Therefore, in implementations, the largest value within the range is used instead of V to determine the number of mantissa bits to retain. This approach not only improves precision but also ensures uniformity in the retained mantissa bits for all floating-points within the range.

To satisfy the relative error, the number of retained mantissa bits R_{rel} must satisfy $2^{R_{rel}} \leq \delta$. Therefore, we can derive:

$$R_{rel} = -p(\delta) \tag{9}$$

By calculating the significant bits to retain for floating-points using the above method and truncating the less important bit planes, we achieve quantization of unpredictable floating-points.

After quantization, we encode the indices generated during the compression process. Smaller integers can be represented with fewer bits, so we store only the differences between consecutive indices along with the first value. This reduces the magnitude of the integers, which are then encoded using variable-length encoding to significantly reduce the size of the indices. Finally, an advanced lossless compressor, ZSTD [27], is used for further compression of the data.

V. Integrating Compression in Sensitivity Simulation

As shown in Figure 7, we integrate the proposed error-bounded lossy compression algorithm into the simulation process, effectively reducing memory usage during both forward and backward propagation. In transient analysis, after successfully solving each time step, we store all the necessary state variables (e.g., state vectors and Jacobian matrices), denoted as D, and compress them to reduce storage. Subsequently, during the adjoint process, these variables are decompressed at the necessary moments to reconstruct the differential equations.

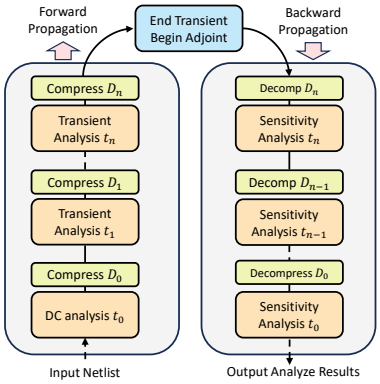


Fig. 7: Proposed memory-efficient simulation.

VI. Experimental Evaluation

TABLE II: Dataset information.

Dataset	Circuit Type	#CirElem	Dimension Size	Data Size
ibm1t (vec)	RLC	76934	54265×1252	518.34MB
ibm1t (mat)	RLC	76934	175022×1252	1.6GB
add20 (vec)	MOS	5091	2479×42799	809.47MB
add20 (mat)	MOS	5091	18189×42799	5.8GB
smult20 (vec)	MOS	46075	28759×9048	1.94GB
smult20 (mat)	MOS	46075	213185×9048	14.37GB

A. Experimental Setup

Platform and software. We implement our proposed compression algorithm in C++ and integrate it into the adjoint sensitivity simulation. The platform used is Xyce [28], an open-source, SPICE-compatible circuit simulator from Sandia National Laboratories. The experiments are conducted on an AMD Ryzen 7 4800H CPU operating at a clock speed of 2.9 GHz.

Datasets. We evaluate the proposed error-bounded lossy compression algorithm on six datasets, consisting of vector and matrix data generated from simulations of both linear and

TABLE III: Comparison of compression ratio, compression time (in seconds), and decompression time (in seconds) for different compression methods.

Dataset	ZSTD [27]			MASC [8]			ZFP [19]			SZ [17]			Our work		
	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}
ibm1t (vec)	1.45	4.49	1.85	1.98	5.23	2.72	4.08	1.3	1.38	11.6	2.02	1.4	12.78	1.97	1.34
ibm1t (mat)	29.59	1.73	1.67	7.17	11.35	8.61	3.54	3.62	0.65	196.23	5.45	2.24	970.11	5.29	1.85
add20 (vec)	6.03	3.63	2.09	2.95	6.79	3.97	5.02	1.79	1.77	76.08	2.67	1.33	72.28	2.76	1.45
add20 (mat)	9.74	10.68	4.92	11.90	36.72	29.28	4.48	12.6	12.72	80.46	18.6	8.52	125.85	18.01	5.52
smult20 (vec)	5.69	4.85	3.51	4.18	10.65	7.84	8.19	4.16	3.25	92.56	7.89	6.56	146.39	4.18	3.27
smult20 (mat)	12.18	25.23	15.72	11.07	86.51	75.06	16.97	29.82	18.68	169.25	50.28	25.72	205.63	45.42	18.17
Average	10.78	8.44	4.96	6.54	26.21	21.25	7.05	8.88	6.41	104.36	14.49	8.31	255.51	12.94	7.93

nonlinear circuits. As shown in Table II, the first column represents the name of the circuit and whether it is vector data or matrix data, the second column is the type of the circuit, the third column indicates the number of elements in the circuit, and the fourth and fifth columns respectively represent the dimensions and size of the data.

Baselines. We compare the proposed algorithm with the state-of-the-art lossless compressor ZSTD [27], the lossless compression algorithm MASC [8] specifically designed for matrices, and two cutting-edge lossy compressors, ZFP [19] and SZ [17]. All algorithms use the latest versions.

Configuration. In scientific computing, the absolute and relative error bounds for lossy compression are typically set to 10^{-5} and 10^{-3} [21], respectively, providing sufficient accuracy while significantly reducing simulation memory overhead. We adopt this configuration for our experiment as well.

B. Compression Performance

As shown in Table III, we compare the proposed error-bounded lossy compression algorithm with the aforementioned compression methods. In terms of compression ratio, our algorithm achieves average ratios of 23.70× and 39.07× higher than the state-of-the-art lossless compression methods ZSTD and MASC, respectively. This result highlights the superior capability of lossy compression in handling floating-point data. When compared to advanced lossy compression methods such as ZFP and SZ, the proposed algorithm achieves compression ratios of 36.24× and 2.44× higher, respectively. Thus, the simulation memory overhead can be reduced by two orders of magnitude.This improvement can be attributed to the algorithm's effective exploitation of the characteristics of circuit simulation data. By leveraging efficient smoothing techniques and predictive modeling, the algorithm effectively eliminates data redundancy in vectors and matrices. In terms of compression and decompression time, the time consumed by the five compression algorithms for both compression and decompression is negligible compared to the overall time cost of the sensitivity simulation itself.

C. Error Impact Evaluation

As shown in Figure 8, we integrate the proposed error-bounded lossy compression algorithm into the adjoint sensitivity simulation to analyze the impact of floating-point errors introduced by lossy compression on the accuracy of sensitivity simulation results. The absolute error bound is set to 10^{-5}, and the relative error bound to 10^{-3}. Using the compression algorithm, we reduce the memory overhead caused by storing vectors and matrices during the simulation process. It can be observed that the sensitivity analysis results at each time point remained within the predefined error bounds after integrating the lossy compression. This is attributed to the proposed algorithm's strict quantization of errors introduced during the compression process. Therefore, the proposed error-bounded lossy compression significantly reduces memory overhead while effectively preserving simulation accuracy.

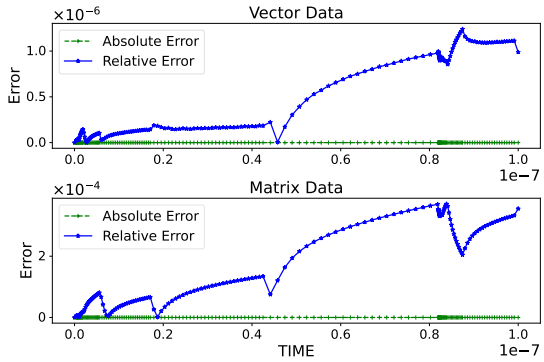


Fig. 8: Error comparison by integrating lossy compression algorithm into transient adjoint sensitive analysis with add20.

VII. Conclusion

This work introduces the first error-bounded lossy compression algorithm capable of efficiently compressing both vector and matrix data in circuit simulations. This approach significantly reduces memory overhead in adjoint sensitivity simulations. Based on the characteristics of data in circuit simulations, we propose an effective smoothing method tailored for simulation data. Subsequently, we design different data prediction algorithms for regions with distinct properties. Finally, through rigorous error quantization and efficient data encoding, the proposed approach significantly reduces the data size while preserving accuracy.

Acknowledgments

The corresponding author is Zhou Jin. This work was supported by the NSFC (Grant Nos. 92473107, 62204265, 62234010, U23A20301, 62032023, and T2125013), and the Innovation Funding of ICT, CAS (Grant No. E461050).

REFERENCES

[1] I. Stevanovic and C. McAndrew. Quadratic backward propagation of variance for nonlinear statistical circuit modeling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[2] B. Agrawal, F. Liu, and S. Nassif. Circuit optimization using scale based sensitivities. In *IEEE Custom Integrated Circuits Conference (CICC'06)*, 2006.

[3] J. Sun, X. Zha, C. Wang, X. Wu, D. Niu, W. Xing, and Z. Jin. Pseudo adjoint optimization: Harnessing the solution curve for spice acceleration. In *ACM/IEEE International Conference on Computer-Aided Design (ICCAD'24)*, 2024.

[4] C. Gu and J. Roychowdhury. An efficient, fully nonlinear, variability-aware non-monte-carlo yield estimation procedure with applications to sram cells and ring oscillators. In *Asia and South Pacific Design Automation Conference (ASPDAC'08)*, 2008.

[5] K. V. Aadithya, E. Keiter, and T. Mei. Dagsens: Directed acyclic graph based direct and adjoint transient sensitivity analysis for event-driven objective functions. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*, 2017.

[6] Y. Cao, S. Li, L. R. Petzold, and R. Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution. *SIAM Journal on Scientific Computing*, 2002.

[7] F. Liu and P. Feldmann. A time-unrolling method to compute sensitivity of dynamic systems. In *ACM/IEEE Design Automation Conference (DAC'14)*, 2014.

[8] C. Li, B. Zhang, Y. Duan, Y. Li, Z. Ye, W. Liu, D. Tao, and Z. Jin. Masc: A memory-efficient adjoint sensitivity analysis through compression using novel spatiotemporal prediction. In *ACM/IEEE Design Automation Conference (DAC'24)*, 2024.

[9] J. Roychowdhury. *Numerical Simulation and Modelling of Electronic and Biochemical Systems*, 2009.

[10] R. Li, Z. Li, Y. Wu, C. Chen, and Y. Zheng. Elf: Erasing-based lossless floating-point compression. *In VLDB Endowment*, 2023.

[11] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong. Full-state quantum circuit simulation by using data compression. *In international Conference for High Performance Computing, Networking, Storage and Analysis (SC'19)*, 2019.

[12] L. Li and W. Yu. Efficient and accuracy-ensured waveform compression for transient circuit simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[13] L. Li, W. Yu, G. Guo, and Z. Zhou. More efficient accuracy-ensured waveform compression for circuit simulation supporting asynchronous waveforms. In *Great Lakes Symposium on VLSI (GLSVLSI'23)*, 2023.

[14] W. Hu, Z. Ye, and Y. Wang. Adjoint transient sensitivity analysis for objective functions associated to many time points. In *ACM/IEEE Design Automation Conference (DAC'20)*, 2020.

[15] N. Sagan and J. Roychowdhury. Transient adjoint dae sensitivities: A complete, rigorous, and numerically accurate formulation. In *Asia and South Pacific Design Automation Conference (ASPDAC'22)*, 2022.

[16] IEEE Standards Committee et al. 754-2008 ieee standard for floating-point arithmetic, 2008.

[17] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *IEEE International Conference on Big Data (Big Data'18)*, 2018.

[18] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 2006.

[19] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 2014.

[20] A. Eldstål-Ahrens and I. Sourdis. Memsz: Squeezing memory traffic with lossy compression. *ACM Transactions on Architecture and Code Optimization (TACO'20)*, 2020.

[21] S. Jin, C. Zhang, X. Jiang, Y. Feng, H. Guan, G. Li, S. L. Song, and D. Tao. Comet: a novel memory-efficient deep learning training framework by using error-bounded lossy compression. *In VLDB Endowment*, 2021.

[22] Y. Huang, K. Zhao, S. Di, G. Li, M. Dmitriev, T.-L. D. Tonellot, and F. Cappello. Towards improving reverse time migration performance by high-speed lossy compression. In *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid'23)*. IEEE, 2023.

[23] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C.-S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 2013.

[24] J. Huang, J. Liu, S. Di, Y. Zhai, Z. Jian, S. Wu, K. Zhao, Z. Chen, Y. Guo, and F. Cappello. Exploring wavelet transform usages for error-bounded scientific data compression. In *IEEE International Conference on Big Data (BigData'23)*, 2023.

[25] Y. Su, F. Yang, and X. Zeng. Amor: An efficient aggregating based model order reduction method for many-terminal interconnect circuits. In *Design Automation Conference (DAC'12)*, 2012.

[26] P. Liakos, K. Papakonstantinopoulou, and Y. Kotidis. Chimp: Efficient lossless floating point compression for time series databases. *In VLDB Endowment.*, 2022.

[27] J. Chen, M. Daverveldt, and Z. Al-Ars. Fpga acceleration of zstd compression algorithm. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'21)*, 2021.

[28] J. Verley, E. Keiter, and H. Thornquist. Xyce: Open source simulation for large-scale circuits. Technical report, Sandia National Lab. (SNL-NM), 2018.

Megabits Down to Kilobits: Memory-Efficient Time-Aware Shaping for TSN

Xuyan Jiang[1], Wenwen Fu[1], Xiangrui Yang[1,*], Yingwen Chen[1], Wenfei Wu[2], Zhigang Sun[1]
[1] College of Computer, National University of Defense Technology, Changsha, China
[2] School of Computer Science, Peking University, Beijing, China
Email: {jiangxuyan, fuwenwen94, yangxiangrui11, ywch, sunzhigang}@nudt.edu.cn, wenfeiwu@pku.edu.cn

***Abstract*—Time-Sensitive Networking (TSN) provides bounded latency and low jitter for cyber-physical systems, such as industrial control. As a key component of TSN, the Time-Aware Shaper (TAS) applies gate control rules to control the transmission time of frames in critical flows. TAS stores the gate control rules for each frame in the gate control table. However, in typical industrial setups, the memory usage of the table could reach over tens of megabits and even exceed the total memory capacity of TSN switches.**

To address this issue, we propose a memory-efficient TAS design named METAS. It transitions from a per-frame to a *per-flow* approach. METAS stores one *persistent rule* for a flow and dynamically generates a *temporary rule* for a frame only when the frame arrives. We prototyped METAS on an FPGA, and experimental results show that METAS reduces memory usage from 14.34 Mbits to 288 Kbits when supporting 1,024 flows, using just 1.56% of the FPGA's logic resources while maintaining microsecond-level latency and nanosecond-level jitter for critical flows.

***Index Terms*—Time-Sensitive Networking, Time-Aware Shaper, Gate Control List, memory-efficiency, Push-In First-Out**

I. Introduction

Time-Sensitive Networking (TSN) is an emerging real-time Ethernet technology that guarantees bounded latency and minimal jitter for critical network flows [1]. This capability is particularly important in industrial internet of things and cyber-physical systems such as industrial automation [2], [3], automotive [4]–[7], and aerospace [8], [9], making TSN a promising solution in these fields [10], [11].

Time-Aware Shaper (TAS) is a key shaping mechanism in TSN designed to ensure bounded latency and minimal jitter for critical flows [12]. TAS stores a *gate control table*, with each row storing a *gate control rule*. These rules determine when gates should open or close, thus controlling the transmission time of each frame of critical flows [1]. Through the gate control table, TAS effectively shapes critical flows with precise timing control.

A gate control table stores gate control rules for a single *hyperperiod* instead of the system's entire lifetime. The hyperperiod in TSN describes the periodicity of all critical flows [13], and is defined as the least common multiple of the periods of all critical flows. By cyclically executing the gate control table over this hyperperiod, TAS can effectively control the transmission time of each frame throughout the system's lifetime.

A key issue with TAS is the high memory consumption of the gate control table. Each critical frame requires two gate control rules (one for open and another for close) in the table. Consequently, the per-frame approach results in the number of gate control rules in the table growing linearly with the number of critical frames within the hyperperiod. As *the number of frames within a hyperperiod can be substantial*, the gate control table results in significant memory consumption. In a typical industrial application [14], [15] involving the shaping of 400 flows comprising 72,040 frames, our evaluation on a FPGA shows that the gate control table requires 19.83 Mbits of static random-access memory (SRAM). This exceeds the FPGA's available SRAM capacity of 15.6 Mbits.

The substantial memory demand poses a significant challenge for deploying TSN, particularly given the limited on-chip memory available in TSN switch implementations. The gate control table is typically stored in on-chip memory to ensure rapid access and low latency [16]. However, on-chip memory resources in modern hardware platforms, such as FPGAs and ASICs, are constrained by considerations of area, cost, and power consumption [17], [18]. Consequently, the high memory usage of gate control tables can interfere with other critical switch functions, such as forwarding tables and packet buffering [19]. It may even exceed the available memory capacity of TSN switches.

To address the memory issue, we propose a memory-efficient TAS design called METAS. It transitions from a per-frame shaping approach to a per-flow shaping approach. Specifically, METAS introduces two types of rules: persistent rules for each flow and temporary rules for each frame. Only persistent rules are stored in the gate control table. Temporary rules are temporarily stored for frames that are currently within the switch because a temporary rule is dynamically generated from the persistent rule when a frame arrives and is released when the frame is transmitted. Since the number of frames buffered simultaneously in a switch is typically small (often fewer than a dozen) [16], [20]–[22], the memory consumption for temporary rules is minimal. This design trades off a small increase in logic resources required to compute temporary rules for significant memory savings in storing persistent rules.

We implemented the METAS prototype and integrated it into an open-source TSN switch [23] on the Xilinx Kintex-7 FPGA. Results show that METAS reduces the memory consumption of the gate control table from 14.34 Mbits to 288 Kbits when supporting 1,024 flows (51,968 frames) in a typical industrial application [14], [15], with only a 1.26% increase in logic resource usage relative to the total logic resources on the FPGA. When METAS is deployed to shape critical flows in a real TSN system, the latency of critical flows meets microsecond-level requirements, with jitter in the hundreds of nanoseconds, demonstrating that METAS ensures their real-time performance.

The main contributions of this paper are as follows:

- We propose a memory-efficient TAS design that significantly reduces the memory consumption of the gate control table.

* Xiangrui Yang is the corresponding author.

This work is supported by the National Foundation under Grant No. 145BHQ090003000X15, the NSF of Hunan under Grant No. 2023JJ40682, the YIISF of NUDT under Grant No. ZK2023-13, and the Foundation of National Key Laboratory under Grant No. 2023-KIWPDL-14.

[1] In this paper, a flow is defined as a sequence of Ethernet frames that share specific field values.

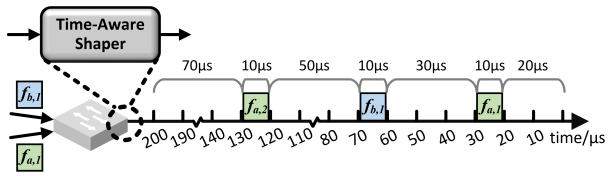


Fig. 1: TAS precisely controls the transmission time of each frame at the output port of a TSN switch.

- METAS shifts from a per-frame shaping approach to a per-flow shaping approach to achieve memory efficiency.
- We verified the significant reduction in memory usage achieved by METAS in a typical industrial TSN application while ensuring that the end-to-end latency and jitter requirements of the critical flows are still met.

II. Background and Motivation

This section introduces the TAS and its gate control table, followed by a simple example to illustrate the motivation of our work.

A. Time-Aware Shaper and Gate Control Table

Time-Aware Shaper (TAS), introduced by the IEEE TSN standard [12], is designed to shape critical flows, which are safety- and time-critical flows with strict periodicity, requiring bounded latency and ultra-low jitter. TSN also supports non-critical flows with relaxed constraints, but this work focuses exclusively on critical flows.

TAS controls the transmission time of each frame in critical flows at the output port of a TSN switch, as shown in Fig. 1. These times are pre-determined by a traffic [2] planning algorithm and expressed as *gate control rules*. The rules are written and statically stored in the TAS *gate control table* (Fig. 2). This configuration is completed prior to runtime (i.e., before any flows are transmitted).

During network runtime, the *gates* execute gate control rules stored in the gate control table to shape traffic, as shown in Fig. 2. Frames arriving at the TAS are dispatched to first-in first-out queues based on a pre-configured mapping. The gates after the queues execute the control rules, each consisting of two fields: an `interval` and the `states` of each gate. A gate transitions to the state (open or closed) specified by the `states` and remains in that state for the duration defined by the `interval`. When a gate is open, frames in the corresponding queue are transmitted. Gates execute the gate control rules cyclically, starting from the first rule, proceeding sequentially to the last, and then looping back to the first. This cyclic execution allows TAS to achieve precise control of frames' transmission times over the entire lifetime of the network.

The total duration covered by the gate control table is referred to as the hyperperiod. The hyperperiod, which models the cyclic transmission behavior of critical flows, is defined as the least common multiple (LCM) of all flow periods [13]. In Fig. 2, flows f_a and f_b have periods of 100 µs and 200 µs, respectively. Consequently, the hyperperiod is 200 µs, and the total duration of the gate control table (i.e., the sum of all `interval`) is also 200 µs.

B. Observations on TAS

Regarding the TAS defined in the TSN standard [12] (referred to as naive TAS in this paper), we make two observations:

First, each frame occupies two gate control rules: one to open the gate and another to close it. For example, in Fig. 2, the transmission of frame $f_{a,1}$ corresponds to the second and third rules, where the second rule opens gate2 for 10 µs and the third rule closes it. Similarly, frame $f_{b,1}$ corresponds to the fourth and fifth rules, and frame $f_{a,2}$ corresponds to the sixth and seventh rules.

Second, flows with shorter periods occupy more gate control rules within a hyperperiod due to having multiple frames. For instance, in Fig. 2, within the 200 µs hyperperiod, flow f_a (with a 100 µs period) has two frames, occupying four gate control rules in total.

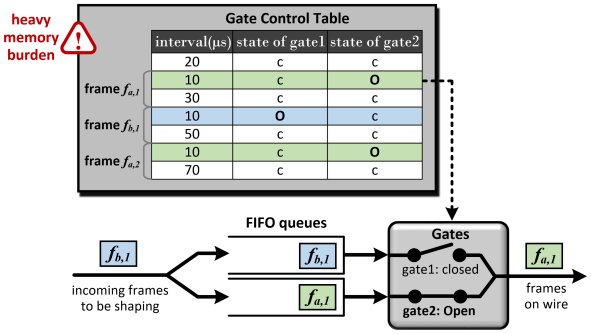


Fig. 2: Gate control table and gates in TAS. The gates executes gate control rules in the gate control table.

C. Memory Burden of Gate Control Table

The TAS described in Fig. 2 leads to a significant memory overhead in the gate control table. The primary reason for this heavy memory demand is that each frame requires two gate control rules, and the number of frames within a hyperperiod can reach a considerable amount. For example, consider a scenario with two flows: flow f_a and flow f_b. If their periods are 10 ms and 100 ms, respectively, the hyperperiod is 100 ms. During this hyperperiod, flow f_a has 10 frames, resulting in 20 gate control rules. However, if the period of flow f_a is 0.1 ms, it has 1000 frames within the hyperperiod, leading to 2000 gate control rules.

The large number of frames within the hyperperiod results in significant memory consumption for the gate control table. To assess this, we synthesized the gate control table on a Xilinx Kintex-7 FPGA. Our evaluation shows that, in a typical industrial application [14], [15] supporting 400 flows and a total of 72,040 frames, the gate control table requires 19.83 Mbits of memory. This memory demand exceeds the FPGA's maximum available memory capacity of 15.6 Mbits. Furthermore, in 62.94% of flow configurations within the industrial application, the memory usage of the gate control table surpasses the FPGA's memory capacity (§VI-B).

This heavy memory usage poses a significant challenge for deploying TSN due to the limited on-chip memory available in TSN switch implementations. Gate control tables are typically implemented using on-chip SRAM to ensure real-time performance. However, SRAM on-chip is severely constrained in hardware platforms such as FPGAs and ASICs due to cost, area, and power limitations [17], [18]. The excessive memory usage not only limits the resources available for critical functions such as forwarding tables and packet buffering [19], but also potentially exceeds the memory capacity of TSN switches. This makes memory consumption a critical bottleneck in TAS design.

III. Key Idea

We observe that the transmission times of a critical flow exhibit periodicity. Specifically, frames within the same flow are transmitted at consistent intervals. Such periodic behavior is frequently observed in the transmission schedule computed by many traffic planning algorithms [21], [24]–[26]. This periodicity serves as the foundation for our subsequent design.

[2]Traffic refers to the set of all critical flows.

Leveraging this periodicity, METAS shifts from a per-frame shaping approach to a per-flow shaping approach. The per-flow approach introduces two types of rules: persistent rules and temporary rules.

- A *persistent* rule records the period and transmission time of the *previous* frame for a flow, which are then used to compute the transmission time of the next frame upon arrival.
- A *temporary* rule records the transmission time of a frame that is currently within the switch. The temporary rule uses the transmission time to guide the gate on when to open or close.

METAS stores N persistent rules in the gate control table using SRAM, where N represents the number of flows. A temporary rule is dynamically generated from the persistent rule. It is created only when a frame arrives at the switch and is released once the frame is transmitted. Thus, a temporary rule only needs to be temporarily stored in the register while the corresponding frame is buffered in the switch. Since the number of frames buffered in the switch at any given time is typically small (usually fewer than a dozen) [16], [20]–[22], the temporary rules consume minimal register resources, which constitute only a small fraction of the total register resources on the hardware platform (§VI-A).

METAS significantly reduces the SRAM usage of the gate control table, with a slight increase in logic resources for dynamically computing temporary rules and a minimal amount of register resources for storing these temporary rules. Overall, METAS achieves a substantial decrease in total memory usage.

IV. Design

Following the key idea discussed in §III, METAS has two main focuses. The first is organizing the persistent rules to ensure efficient storage and fast access. The second addresses the temporary rules, detailing how to compute, manage, and execute them. The architecture of METAS, as illustrated in Fig. 3, consists of four components:

- *Gate Control Table with Persistent Rules*: Stores N persistent rules, one for each of the N flows. Each rule contains the flow's period and the transmission time of its *previous* frame.
- *Temporary Rule Generator*: For each arriving frame, it computes a temporary rule by calculating its transmission time. The transmission time is determined based on the flow's period and the transmission time of the previous frame.
- *Push-In First-Out Queue (PIFO)*: Temporarily stores and manages temporary rules. As a hardware priority queue, PIFO allows elements to be inserted at any position and permits output from the head [27]. In METAS, temporary rules are stored in the PIFO and sorted by transmission time, with the earliest placed at the head.
- *PIFO-based Gate*: Executes temporary rules by comparing the current time with the transmission time of the temporary rule at the PIFO head. When they match, the gate opens to transmit the corresponding frame.

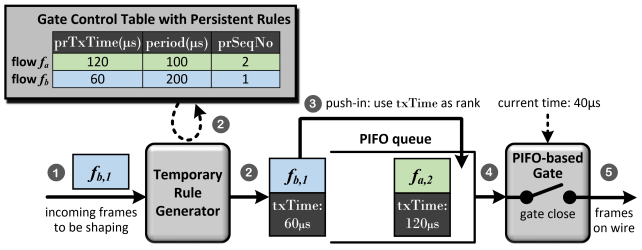


Fig. 3: Architecture of METAS.

A. Gate Control Table with Persistent Rules

The Gate Control Table with Persistent Rules stores N persistent rules, one for each critical flow, indexed by the unique flow identity. Fetching a rule based on the flow identity from the table has $\mathcal{O}(1)$ time complexity.

Each persistent rule contains three fields: `prTxTime`, which stores the transmission time of the previous frame of the flow; `period`, which represents the period of the flow; and `prSeqNo` records the sequence number of the previous frame to handle the rare occurrence of frame losses. The `prTxTime` and `prSeqNo` are updated by the Temporary Rule Generator upon the arrival of a new frame in the flow.

The table is configured before the switch begins operation. To initialize the table, a virtual "0-th" frame is created for each flow. This virtual frame records the transmission time of the frame preceding the first actual frame. Its transmission time is set to the first frame's transmission time minus the flow's period.

B. Temporary Rule Generator

The Temporary Rule Generator (referred to as Generator for simplicity) computes temporary rules for arriving frames. A temporary rule records the transmission time of a frame.

The Generator performs three actions for each frame. First, it calculates the transmission time `txTime` based on the persistent rule in the gate control table. After the transmission time is calculated, it is updated into the `prTxTime` field of the persistent rule. The sequence number of the current frame `seqNo` is also updated to the `prSeqNo` fields in the table. Finally, the frame, along with the temporary rule, is enqueued into the PIFO for shaping.

The transmission time calculation includes two scenarios: without frame loss (the typical case) and with frame loss (a rare edge case [14], [15], [28]).

Transmission Time Calculation Without Frame Loss. In the vast majority of cases without frame loss, the transmission time for each frame is calculated as:

$$f_{i,j}.\texttt{txTime} = (f_i.\texttt{prTxTime} + f_i.\texttt{period}) \bmod HP \quad (1)$$

This formula leverages the periodic nature of frame transmissions. By adding the flow's period to the `prTxTime` of the previous frame, the transmission time for the current frame $f_{i,j}$ is derived. The modulo operation with the hyperperiod (HP) ensures that the transmission time is represented as an offset within HP. Using offsets instead of absolute time aligns with TSN standards [29] and simplifies the representation and comparison of time values [21].

Transmission Time Calculation With Frame Loss. Although TSN employs robust reliability mechanisms, frame loss may still occur under rare conditions [28]. In such cases, the transmission time calculation must account for the lost frames.

We use the sequence number to detect frame loss. Each critical frame is assigned a sequence number [28], which increases sequentially within a flow. Frame loss is detected by comparing the sequence number of the newly arriving frame with the expected value, which is the previous frame's sequence number `prSeqNo` incremented by one. A mismatch indicates frame loss.

When frame loss occurs, the transmission time `txTime` is recalculated as follows:

$$f_{i,j}.\texttt{txTime} = (f_i.\texttt{prTxTime} + x \cdot f_i.\texttt{period}) \bmod HP \quad (2)$$

Here, x represents the number of skipped transmission cycles, which is the difference between the sequence number of the arriving frame and the sequence number of the previous frame.

Latency Analysis. The additional computation for transmission time introduced by the Generator may slightly increase the processing latency of each frame within the switch. In TSN, the upper bound of frame processing time in a switch can be considered during offline planning and is typically much smaller than the frame's sending interval. By appropriately adjusting this upper bound through offline planning, we can ensure that the increased processing time does not lead to performance degradation in terms of end-to-end latency or jitter [24]. In §VI-C, we evaluate the end-to-end latency and jitter in a real TSN system, showing that METAS effectively shapes traffic without compromising real-time performance.

C. PIFO for TAS

The PIFO queue temporarily stores and manages temporary rules, sorting them by transmission time with the earliest at the head. PIFO sorts incoming frames in just two clock cycles [27], outperforming traditional methods such as min-heap and binary comparator tree. This efficiency makes PIFO an optimal choice for TAS design.

Length of PIFO. In TSN, queues for critical flows are typically shallow because the number of frames buffered in a switch at any given time is small, often fewer than a dozen [16], [20]–[22]. Our design uses a PIFO queue depth of 10, which is sufficient to handle critical flows efficiently without over-provisioning.

PIFO uses flip-flops (the basic storage units of registers) to maintain the ordered list of temporary rules [27]. Under our queue length, experimental results show that the flip-flop consumption does not exceed 0.19% of the available resources on the FPGA (§VI-A). Therefore, the resource consumption for storing temporary rules is minimal.

Sorting of PIFO. The PIFO in our design follows the "earliest transmission time first" principle to sort temporary rules and their frames. However, since `txTime` values are represented as relative offsets within the hyperperiod rather than absolute times, directly ordering rules by `txTime` alone can lead to incorrect scheduling in certain cases.

For example, assume the current time is 15. A frame $f_{a,i}$ arrives with a `txTime` of 0, indicating it should be transmitted in the *next* hyperperiod. Meanwhile, a frame $f_{b,j}$ already in the queue has a `txTime` of 30. If we strictly order by `txTime`, $f_{a,i}$ would incorrectly be placed ahead of $f_{b,j}$, even though $f_{a,i}$ belongs to the next hyperperiod. This results in an invalid transmission order.

To address this, we adjust the `txTime` of temporary rules. If a rule's `txTime` is less than the current time, we add a hyperperiod HP to it, placing the rule in the next hyperperiod. This adjusted `txTime` is used only for sorting and is not updated in the persistent rule. When a new hyperperiod starts (i.e., when the current time resets to 0), the temporary rules in the queue are updated: if the `txTime` of a temporary rule exceeds a hyperperiod HP, we subtract HP from it. Since the PIFO queue is short, the cost of this update is negligible.

D. PIFO-based Gate

The PIFO-based Gate executes temporary rules and functions as follows.

If the PIFO queue is non-empty and the transmission time of the head rule matches the current time, the gate opens, allowing the critical frame to be transmitted immediately. Once the transmission is complete, the gate closes. In all other cases, the gate remains closed. This occurs when the PIFO queue is non-empty but the head rule's transmission time has not yet arrived, i.e., the current time is earlier than the transmission time, or when the PIFO queue is empty.

This simple mechanism ensures that frames are scheduled strictly according to their transmission times.

Interaction with Other Shaping or Scheduling Mechanisms. The PIFO-based Gate operates independently of other scheduling algorithms in TSN, such as credit-based shaping or strict-priority scheduling [29]. These algorithms, which manage non-critical traffic, are active only during idle slots, when the PIFO-based Gate is not scheduling critical flows. As a result, there is no interference between the PIFO-based Gate and other mechanisms.

E. Memory Usage Analysis

METAS stores persistent rules in the gate control table. The memory usage of the table is given by:

$$\mathbb{M}(\text{METAS}) = (w_{\texttt{prTxTime}} + w_{\texttt{period}} + w_{\texttt{prSeqNo}}) \times N \tag{3}$$

where $w_{\texttt{prTxTime}}$, $w_{\texttt{period}}$, and $w_{\texttt{prSeqNo}}$ are the widths of the respective fields in a persistent rule, and N is the total number of critical flows, which also represents the depth of the gate control table in METAS. According to the equation above, the space complexity of the gate control table in METAS is $\mathcal{O}(N)$.

For comparison, the memory usage of the naive TAS is as follows:

$$\mathbb{M}(\text{naiveTAS}) = (w_{\texttt{interval}} + w_{\texttt{states}}) \times D(\text{naiveTAS}) \tag{4}$$

where $w_{\texttt{interval}}$ and $w_{\texttt{states}}$ are the widths of the interval and gates states fields in each gate control rule of the naive TAS. The total number of gate control rules $D(\text{naiveTAS})$ is given by:

$$D(\text{naiveTAS}) = 2 \times \sum_{i=1}^{N} \frac{\text{LCM}(P_1, P_2, \ldots, P_N)}{P_i} + 1 \tag{5}$$

Here, P_i is the period of flow f_i, and the sum represents the total number of frames within a hyperperiod, where the hyperperiod is the least common multiple (LCM) of the periods $P_1, P_2, \ldots, P_N$. Thus, the space complexity of naive TAS is $\mathcal{O}(M \cdot N)$, where M is the average number of frames for flows within a hyperperiod.

METAS reduces the space complexity from $\mathcal{O}(M \cdot N)$ to $\mathcal{O}(N)$, making it a memory-efficient solution.

F. Limitation

Scalability of PIFO. Scalability concerns can arise with PIFO queues due to their resource consumption. Prior studies have shown that when the queue length reaches 1,024, PIFO implementations consume significant logic and register resources [27], [30]. However, since the PIFO queue in our design is shallow, scalability limitations are effectively mitigated, ensuring resource-efficient implementation.

Requirements for Planning Algorithms. Our approach leverages the periodicity of frame transmission times, which imposes a requirement on offline traffic planning algorithms: the transmission times of frames belonging to the same flow traversing the same link must exhibit periodicity. Fortunately, this condition holds for the majority of existing algorithms [24]–[26]. For algorithms that do not inherently meet this requirement, adapting them to METAS is straightforward, as simply adding a periodicity constraint is sufficient.

V. Implementation

We prototyped METAS on a Xilinx Kintex-7 FPGA, and integrated it into OpenTSN [23], an open-source TSN switch.

The gate control table is implemented using dual-port RAM. The fields `prTxTime`, `period`, and `prSeqNo` have widths of 20 bits, 20 bits, and 16 bits, respectively. The values of `prTxTime` and `period` are in multiples of 1 microsecond, as the minimum

planning granularity is typically 1 microsecond on 1 Gbps network links [24]. Thus, the 20-bit width for these fields is sufficient to represent durations up to one second, which is adequate for most TSN applications [14], [31]. The 16-bit width of `prSeqNo` corresponds to the frame sequence number used in the switch integrated with our prototype. The gate control table has a depth of 1024 entries, allowing the prototype to support up to 1024 critical flows.

PIFO is implemented using the classic parallel comparator-and-shift architecture [27], [32], supporting one push and one pop operation per cycle. To improve efficiency, two PIFO queues are used: one for temporary rules and one for frame metadata. This separation reduces the bit width of data that needs to be compared, as rules are sorted based on their transmission time. The depth of both queues is set to 10, which is typical for TSN designs [16], [20]–[22]. The rule queue has a width of 20 bits, matching the size of `prTxTime`, while the metadata queue has a width of 70 bits. The 70-bit frame metadata represents the frame itself, as the OpenTSN switch processes metadata instead of full frames throughout the entire pipeline to enhance efficiency.

The generator is implemented using two parallel circuits to compute `TxTime` for both frame loss and no frame loss scenarios, ensuring minimal processing time. In the case of frame loss, we use a straightforward iteration approach: repeatedly adding `period` to `txTime` to account for the lost frames, avoiding the use of multiplier digital signal processors. Since frame loss is rare and the number of consecutive frames lost typically only ranges from 3 to 4 [14], [15] due to the reliability mechanisms of TSN [28], the iteration count remains minimal. The modulus operation in Equation (1) is implemented using a comparator and subtractor. Since the value to be modded can exceed at most one hyperperiod, we simply subtract one hyperperiod to obtain the correct result, avoiding the need for more complex components.

VI. Evaluation

This section details the FPGA resource usage of the METAS prototype, evaluates memory savings in industrial settings, and evaluates its real-time performance in a TSN system.

A. Prototype

The METAS prototype was implemented on a Xilinx Kintex-7 FPGA [33], which provides 203,800 lookup tables (LUTs), 407,600 flip-flops (FFs), 15.6 Mbps of block random-access memory (BRAMs) and four 1 Gbps Ethernet interfaces. Each of the four Ethernet ports on the FPGA requires TAS deployment. Therefore, we evaluated the total resource consumption of the prototypes deployed in all four ports. For comparison, a naive TAS was also implemented as a baseline.

METAS supports 1,024 flows with arbitrary periods. To describe the capability of the naive TAS, we consider two flow periods commonly found in industrial applications: 100 µs and 20 ms [14], [15]. Based on these flow periods, the naive TAS can support 1,024 flows, but only if no more than 25% have a 100 µs period (equivalent to no more than 51,968 frames).

Table I summarizes the resource usage of the METAS and naive TAS prototypes, including BRAMs, FFs, and LUTs. These resources represent the consumption of SRAM, register and logic on the FPGA, respectively. The table also lists the utilization of these resources as a percentage of the FPGA's total capacity.

The BRAM consumption of the gate control table in METAS is only 288 Kbits. In contrast, the naive TAS nearly exhausts all available memory resources on the FPGA, consuming 14.34 Mbits, which accounts for 91.69% of the total memory capacity. METAS uses more FFs than the naive TAS, as the PIFO queue relies on FFs to maintain a sorted list [30]. However, FF usage increases by only 0.11% of the total available FFs on the FPGA. Similarly, LUT usage increases by 1. 26% due to dynamic computation and management of temporary rules in METAS. These results demonstrate that METAS achieves a highly efficient trade-off between SRAM savings and increased logic and register utilization.

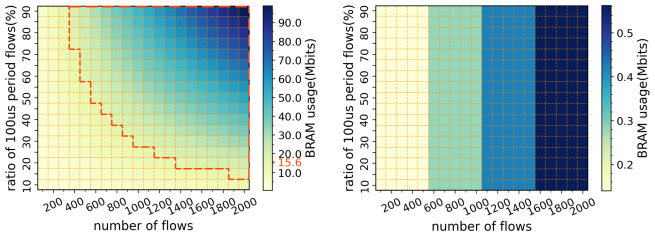
TABLE I: Footprint of naive TAS and METAS on a Xilinx Kintex-7 FPGA

Metric	BRAM	FF	LUT
Naive TAS	14.34 Mbits (91.69%)	313 (0.08%)	607 (0.30%)
METAS	288 Kbits (1.80%)	770 (0.19%)	3179 (1.56%)

B. Memory Usage

This section presents the synthesis results on the selected FPGA, focusing on the memory consumption of the gate control table in METAS and the naive TAS. These results were obtained in different flow setups. For flow setups, we selected typical traffic characteristics from an industrial application [14], [15], considering various combinations of flow periods and flow counts, which result in different numbers of frames within the hyperperiod. Specifically, we chose two representative periods: 100 µs and 20 ms, with flow counts ranging from 100 to 2,000 in increments of 100. The proportion of flows with a 100 µs period was progressively increased from 10% to 90%.

The results, shown in Fig.4a and Fig.4b, illustrate the memory resource usage across different flow counts and period ratios. The reported memory consumption is the total resource usage of METAS or naive TAS across all four Ethernet ports on the FPGA. Each Ethernet port deploys a separate instance of the METAS or naive TAS prototype, so the values in the figures represent the sum across all four ports. And the widths of the gate control tables in METAS and naive TAS are consistent with the descriptions in §V.



(a) naive TAS (b) METAS

Fig. 4: Memory usage comparison of naive TAS and METAS for different flow configurations

The memory usage of METAS consistently remained below 576 Kbits across all setups, whereas the memory consumption of naive TAS reached up to 99 Mbits. In approximately 62.94% of the setups, naive TAS exceeded the available BRAM capacity of the selected FPGA, which is 15.6 Mbits. The setups where the memory usage of naive TAS exceeds the available memory capacity are highlighted in Fig. 4a. A red dotted line marks the area in the upper right corner. The BRAM usage in this area was estimated using the fixed primitives algorithm, the same BRAM concatenation method. These results highlight the high memory efficiency of METAS.

As the proportion of 100 µs period flows increases from 20% to 80%, the memory usage of METAS remains constant, while naive

TAS's memory usage grows significantly (Fig. 5). This difference arises because METAS's resource consumption depends only on the total number of flows, whereas naive TAS's consumption scales with the number of frames within the hyperperiod. As the proportion of 100 µs period flows increases, the number of frames within the hyperperiod rises, leading to higher memory usage for naive TAS. This highlights a key distinction: METAS employs a per-flow approach, while naive TAS uses a per-frame approach.

Another notable observation is the "layered" pattern in METAS's memory usage. For example, the BRAM usage is consistently 144 Kbits for 100, 200, 300, ..., 500 flows, and 288 Kbits for 600, 700, ..., 1000 flows. This pattern results from BRAM concatenation in FPGA, where larger memory is created by combining basic BRAMs of fixed size. As a result, the synthesized memory usage exceeds the theoretical requirement, as shown in Fig. 6.

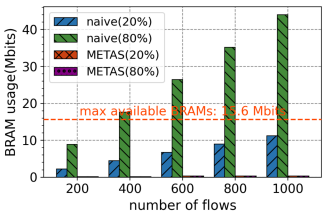


Fig. 5: BRAM usage of naive TAS and METAS under varying flow parameters

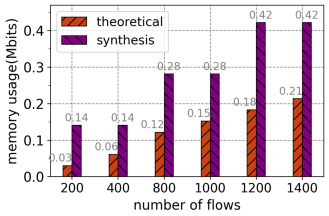


Fig. 6: Theoretical memory usage and synthesized memory usage on FPGA

C. Real-Time Performance

To evaluate the impact of METAS on real-time performance, we conducted tests in a real network built with OpenTSN switches, with METAS integrated into each switch. The network topology, shown in Fig. 7, is a dumbbell shape, consisting of four endpoints and two switches. The network contains 10 critical flows, five of which have a period of 100 µs and the remaining five have a period of 20 ms. The end-to-end latency requirements for each flow are 100 µs and 200 µs, respectively, with a jitter requirement of less than 1 µs. Additionally, each endpoint generates a background flow at line rate to serve as interference. The paths for these flows are randomly assigned. Each frame is 64 bytes in size, and the link rate is set to 1 Gbps. The precision of time synchronization is 48 nanoseconds.

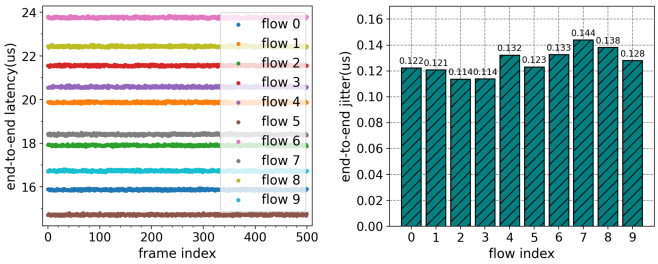


Fig. 7: Diagram and real system of dumbbell topology

We collected end-to-end latency data for 500 frames per flow, as shown in Fig. 8a. The results show that all ten flows meet their end-to-end latency requirements, all within the microsecond range. The jitter is minimal, as illustrated in Fig. 8b. For all ten flows, the jitter is approximately 120-140 nanoseconds, satisfying the stringent jitter requirements of the most demanding applications [34]. These results demonstrate that METAS improves memory efficiency without compromising the system's real-time performance.

VII. Related Work

Optimization for Gate Control Table in TSN. Current TSN research has primarily focused on generating gate control rules for TAS, i.e., traffic planning algorithms [21], [24]–[26], [35], with limited research addressing the memory usage of these gate control



(a) End-to-end latency (b) End-to-end jitter

Fig. 8: End-to-end latency and jitter of flows in the real system

rules. The work in [36] proposes an algorithm that reduces the number of gate control rules by merging multiple frames into a single rule, namely, transmitting them consecutively (back-to-back). While this approach reduces the rules, it has two key drawbacks: it doesn't address the underlying memory challenges in TAS, and it introduces jitter, as the order of the merged frames is non-deterministic [25]. Another work, [37], optimizes policing rules for critical flows in TSN. These rules are used to determine whether a frame arrives within the expected time window. They also face memory challenges due to a similar underlying cause as TAS. Although this design also uses a per-flow approach similar to METAS, it is limited by the number of flows with distinct periods. For example, it can support 1,024 flows with 8 distinct periods, but cannot support 9 flows with 9 periods. The same limitation is observed in a commercial TSN switch [38], one of the few for which TAS design details are publicly available.

Priority Queue. The METAS uses the priority queue, PIFO, to achieve fast sorting. PIFO provides a queue abstraction that allows elements to be inserted at any position based on their priority, while dequeuing elements from the head of the queue [27]. The time complexity for PIFO's enqueue and dequeue operations is $\mathcal{O}(1)$, requiring just one clock cycle to perform two push operations and one pop operation. However, its scalability is limited, particularly when the queue depth exceeds 1,024 [30]. To address this limitation, subsequent works such as AIFO [39], SP-PIFO [40] and BMW-Tree [41] have focused on improving PIFO's scalability. Push-In Extract-Out queue [30] and P-Heap [42] are priority queues that allow sorting of elements based on multiple priority levels. However, since METAS only requires sorting based on the transmission time of data frames, PIFO is sufficient for our purposes. In addition to PIFO, other data structures like min-heaps and binary comparator tree can also be used to implement priority queues. However, these structures have higher time complexity for dequeue operations, $\mathcal{O}(logn)$, where n is the queue length, making them less efficient than PIFO for fast shaping.

VIII. Conclusion

TAS in TSN controls the output timing of frames in critical flows by executing the gate control rules stored in the gate control table, ensuring bounded latency and low jitter. However, the traditional per-frame approach of TAS leads to considerable memory consumption in the gate control table, which can surpass the total memory capacity of TSN switches. Our proposed solution, METAS, achieves a memory-efficient TAS design by adopting a per-flow approach instead of the per-frame approach. METAS stores only one persistent rule for each flow in the gate control table and dynamically generates the corresponding temporary rules upon the arrival of each frame. With this design, METAS significantly reduces the memory usage of TAS at the cost of a slight increase in logic resources, while maintaining microsecond-level latency and nanosecond-level jitter.

REFERENCES

[1] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, 2018.

[2] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proceedings of the IEEE*, 2019.

[3] T. Zhang, G. Wang, C. Xue, J. Wang, M. Nixon, and S. Han, "Time-Sensitive Networking (TSN) for Industrial Automation: Current Advances and Future Directions," *ACM Computing Surveys*, 2025.

[4] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-Sensitive Networking in Automotive Embedded Systems: State of the Art and Research Opportunities," *Journal System Architecture*, 2021.

[5] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Transactions on Industrial Informatics*, 2019.

[6] S. Samii and H. Zinner, "Level 5 by Layer 2: Time-Sensitive Networking for Autonomous Vehicles," *IEEE Communications Standards Magazine*, 2018.

[7] "P802.1DG - TSN Profile for Automotive In-Vehicle Ethernet Communications," https://1.ieee802.org/tsn/802-1dg/, [Accessed 31-10-2024].

[8] S. F. Bush, "Toward Efficient Time-Sensitive Network Scheduling," *IEEE Transactions on Aerospace and Electronic Systems*, 2022.

[9] "P802.1DP - TSN for Aerospace Onboard Ethernet Communications," https://1.ieee802.org/tsn/802-1dp/, [Accessed 31-10-2024].

[10] R. Mahfouzi, A. Aminifar, S. Samii, P. Eles, and Z. Peng, "Security-aware Routing and Scheduling for Control Applications on Ethernet TSN Networks," *ACM Transactions on Design Automation of Electronic Systems*, 2020.

[11] V. Gavriluţ and P. Pop, "Traffic-type Assignment for TSN-based Mixed-criticality Cyber-physical Systems," *ACM Transactions on Cyber-Physical Systems*, 2020.

[12] "IEEE Std 802.1Qbv-2015: IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," https://standards.ieee.org/ieee/802.1Qbv/6068/, [Accessed 31-10-2024].

[13] J. Lin, W. Li, X. Feng, S. Zhan, J. Feng, J. Cheng, T. Wang, Q. Li, Y. Wang, F. Li *et al.*, "Rethinking the Use of Network Cycle in Time-Sensitive Networking (TSN) Flow Scheduling," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2022.

[14] "Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types." [Online]. Available: https://hub.iiconsortium.org/tsn-converged-traffic-types

[15] "IEC/IEEE 60802 TSN Profile for Industrial Automation," https://1.ieee802.org/tsn/iec-ieee-60802/, [Accessed 31-10-2024].

[16] J. Sanchez-Garrido, B. Aparicio, J. G. Ramírez, R. Rodriguez, M. Melara, L. Cercós, E. Ros, and J. Diaz, "Implementation of a Time-Sensitive Networking (TSN) Ethernet Bus for Microlaunchers," *IEEE Transactions on Aerospace and Electronic Systems*, 2021.

[17] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," *IEEE Circuits and Systems Magazine*, 2021.

[18] J.-S. Seo, "Advances and Trends on On-Chip Compute-in-Memory Macros and Accelerators," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.

[19] J. Yan, W. Quan, X. Yang, W. Fu, Y. Jiang, H. Yang, and Z. Sun, "TSN-Builder: Enabling Rapid Customization of Resource-Efficient Switches for Time-Sensitive Networking," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[20] Z. Li, H. Wan, Y. Deng, X. Zhao, Y. Gao, X. Song, and M. Gu, "Time-Triggered Switch-Memory-Switch Architecture for Time-Sensitive Networking Switches," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[21] J. Yan, W. Quan, X. Jiang, and Z. Sun, "Injection Time Planning: Making CQF Practical in Time-Sensitive Networking," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[22] W. Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2010.

[23] W. Quan, W. Fu, J. Yan, and Z. Sun, "OpenTSN: an Open-Source Project for Time-Sensitive Networking System Development," *CCF Transactions on Networking*, 2020.

[24] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless, and S. Han, "Real-time Scheduling for 802.1 Qbv Time-Sensitive Networking (TSN): A Systematic Review and Experimental Study," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024.

[25] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *ACM RTNS*, 2016.

[26] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Optimizing Message Routing and Scheduling in Automotive Mixed-Criticality Time-Triggered Networks," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017.

[27] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable Packet Scheduling at Line Rate," in *ACM SIGCOMM*, 2016.

[28] "IEEE Std 802.1CB-2017: IEEE Standard for Local and Metropolitan Area Networks - Frame Replication and Elimination for Reliability," https://ieeexplore.ieee.org/document/8091139, [Accessed 31-10-2024].

[29] "IEEE Std 802.1Q-2022: Bridges and Bridged Networks," https://standards.ieee.org/ieee/802.1Q/10323/, [Accessed 31-10-2024].

[30] V. Shrivastav, "Fast, Scalable, and Programmable Packet Scheduler in Hardware," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019.

[31] "Summary of Aerospace Use Cases," https://grouper.ieee.org/groups/802//1/files/public/docs2021/dp-Jabbar-Aerospace-UseCase-Summary-0521-v01.pdf, [Accessed 31-10-2024].

[32] S.-W. Moon, J. Rexford, and K. Shin, "Scalable Hardware Priority Queue Architectures for High-Speed Packet Switches," *IEEE Transactions on Computers (TC)*, 2000.

[33] "AMD Kintex™ 7 FPGAs," https://www.amd.com/zh-cn/products/adaptive-socs-and-fpgas/fpga/kintex-7.html, [Accessed 18-11-2024].

[34] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," *IEEE Communications Surveys & Tutorials*, 2019.

[35] X. He, X. Zhuge, F. Dang, W. Xu, and Z. Yang, "DeepScheduler: Enabling Flow-Aware Scheduling in Time-Sensitive Networking," in *IEEE Conference on Computer Communications (INFOCOM)*, 2023.

[36] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding," in *IEEE RTAS*, 2018.

[37] X. Jiang, X. Yang, T. Zhou, W. Fu, W. Quan, Y. Jiao, Y. Sun, and Z. Sun, "Towards Memory-Efficient Traffic Policing in Time-Sensitive Networking," https://arxiv.org/abs/2403.01652, 2024, arXiv preprint arXiv:2403.01652.

[38] NXP, "SJA1105: 5-Port Automotive Ethernet Switch," https://www.nxp.com.cn/docs/en/data-sheet/SJA1105.pdf, [Accessed 31-10-2024].

[39] Z. Yu, C. Hu, J. Wu, X. Sun, V. Braverman, M. Chowdhury, Z. Liu, and X. Jin, "Programmable Packet Scheduling With a Single Queue," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2021.

[40] A. G. Alcoz, A. Dietmüller, and L. Vanbever, "SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[41] R. Yao, Z. Zhang, G. Fang, P. Gao, S. Liu, Y. Fan, Y. Xu, and H. J. Chao, "BMW Tree: Large-Scale, High-Throughput and Modular PIFO Implementation using Balanced Multi-Way Sorting Tree," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2023.

[42] R. Bhagwan and B. Lin, "Fast and Scalable Priority Queue Architecture for High-Speed Network Switches," in *IEEE Conference on Computer Communications (INFOCOM)*, 2000.

3D-SubG: A 3D Stacked Hybrid Processing Near/In-Memory Accelerator for Subgraph GNNs

Guoxiang Li[1,2], Runnan Xu[1], Ruohang Xu, Yikan Qiu, Renati Tuerhong, Muhan Zhang[1], Le Ye[2], Yufei Ma[1,2*]
[1]Institute for Artificial Intelligence, Peking University, Beijing, China
[2]School of Integrated Circuits, Peking University, Beijing, China

***Abstract*—Subgraph Graph Neural Networks (GNNs) are emerging as a promising approach to enhance GNN expressiveness, but their more complex graph structures with numerous independent and irregular subgraphs pose significant hardware deployment challenges. In this work, we propose 3D-SubG, a 3D stacked hybrid processing-near/in-memory accelerator for subgraph GNNs. With hybrid bonding packaging technology, a logic die is 3D stacked with a DRAM die for highly parallel memory accesses. The logic die employs digital SRAM-based processing-in-memory (PIM) macros to boost computation density and minimize data transfer. We further propose a bit-level non-zero gathering method to exploit graph sparsity for PIM, a workload-balanced mapping strategy for subgraph allocation onto different logic-to-DRAM blocks, and a distributed global pooling approach to reduce inter-block data movements. Experimental results show that 3D-SubG achieves average improvements of 146.11× in performance, 934.18× in area efficiency, and 1171.80× in energy efficiency compared to RTX 3090Ti.**

***Index Terms*—Processing in Memory, Graph Neural Networks, Near-Memory Processing, 3D Stacked Architecture**

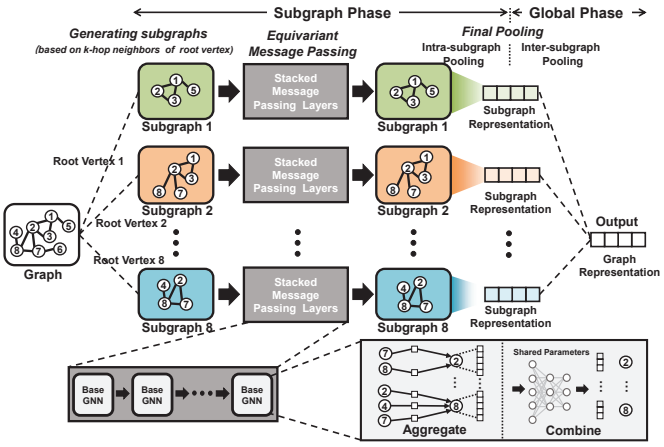


Fig. 1. An illustration of general subgraph GNN architecture.

I. Introduction

Graph Neural Networks (GNNs) have exhibited exceptional capabilities in handling graph-structured data across diverse domains, including graph mining [1], [2], recommendation systems [3], biology [4], chemistry [5], and so on. However, there is a growing consensus that traditional GNNs suffer from limited expressive power [6]–[9], constrained by the first-order Weisfeiler-Leman (1-WL) test [10], which is a common measure of GNN expressiveness. Studies indicate that GNNs equivalent to the 1-WL test fall short in capturing essential structural concepts, like motif counting (e.g., cycles, triangles) [9], [11], [12], which are proven to be pivotal in fields such as bioinformatics and chemoinformatics [13].

Recently, subgraph GNNs [9], [14]–[17], as illustrated in Fig. 1, are emerging to surpass the expressive power of traditional GNNs, approaching the level of 3-WL [9], [14], [18]. A general subgraph GNN operates in two phases: the subgraph phase, which confines computations to individual subgraphs, and the global phase, which processes all subgraphs. By decomposing a graph into multiple subgraphs processed by independent base GNNs, subgraph GNNs boost expressive power. However, they demand significantly higher hardware complexity due to the need to process numerous subgraphs with vertices often vastly exceeding the original graphs by orders of magnitude, as shown in Table II. It is worth noting that the same vertices shared among different subgraphs cannot reuse vertex feature vectors, because after message passing, the feature values of the same vertex differ across subgraphs.

Subgraph GNNs pose greater challenges for hardware deployment. Both CPUs and GPUs are inadequate for efficiently deploying GNN models [19]–[21], let alone the more complex subgraph GNNs. Existing GNN accelerators [19]–[22] are optimized for conventional GNNs. However, subgraph GNNs involve numerous independent subgraph operations performed by stacked base GNNs, requiring not only high computing power and memory bandwidth but also the ability to handle subgraph-level parallelism with irregular data accesses. The conventional GNN accelerators, optimized for single-graph processing, struggle with the parallel processing of subgraphs and the accompanying challenges, such as load balancing and massive uneven data movements.

To facilitate the efficient deployment of subgraph GNNs, we propose 3D-SubG, an accelerator that integrates 3D processing-near-memory (PNM) architecture with digital processing-in-memory (PIM) macros. The PNM architecture [23], [24] based on 3D hybrid bonding packaging technology achieves a tight coupling between DRAM and logic dies through numerous metal pillars. It boasts dense I/Os with a density about 110,000/mm² by around 3μm pitch [23], ensuring exceptionally high bandwidth communication be-

*Corresponding Author: Yufei Ma. (yufei.ma@pku.edu.cn)

This work was supported in part by National Key R&D Program of China (No. 2023YFB4404603), in part by National Natural Science Foundation of China (No. 92164301, 62204003, 62225401, and 61927901).

tween the two dies. By deploying subgraph GNNs using the 3D PNM architecture, 3D-SubG not only benefits from extremely high off-chip bandwidth but also leverages the parallel processing capabilities of multiple distributed logic-to-DRAM (L2D) blocks. Each logic block can independently interact with its corresponding DRAM block to fully exploit the inherent parallelism of subgraphs. Furthermore, we employ PIM macros [25], [26] on the logic die to execute matrix multiplications (MMs) during the combination stage of base GNNs, enhancing the computation density and minimizing weight (W) movements. In addition, the aggregation in base GNNs, formulated as a substantial MM between an ultra-sparse adjacency matrix A and a comparatively sparse feature matrix X, is remarkably irregular and sparse. This makes it more suited for SIMD-based engines, which are designed to flexibly handle such aggregation stage and other vector operations. The contributions are summarized as follows:

- Through comprehensive analysis and profiling, we propose 3D-SubG, a 3D stacked hybrid processing near/in-memory accelerator tailored for subgraph GNNs.
- PIM macros with fixed and regular computation pattern can hardly exploit sparsity in base GNNs. Thus, we employ a Digital PIM (DPIM) macro featuring flexible dataflow and propose bit-level non-zero gathering technique to further leverage data sparsity.
- To efficiently manage the varied workloads of subgraphs, we propose a greedy-based subgraph mapping algorithm, which mitigates workload imbalances among L2D blocks during the subgraph phase.
- The final global pooling step involves aggregating information across all subgraphs, which can lead to extensive inter-block data movement. We introduce a distributed pooling approach, which reduces the substantial inter-block data transfers and alleviates inter-block load imbalance.
- We have implemented 3D-SubG using 22nm technology. In comparison to RTX 3090Ti GPU, 3D-SubG demonstrates an average performace improvement of 146.11×, an area efficiency improvement of 934.18×, and an energy efficiency improvement of 1171.80×.

II. Basis of Subgraph GNNs

Fig. 1 illustrates the execution process of Subgraph Graph Neural Network (Subgraph GNN) models. After extracting the subgraphs from the initial graph based on the k-hop neighbors of the root vertices, the subgraph GNNs operate two phases: subgraph phase and global phase.

Subgraph phase comprises equivariant message-passing and intra-subgraph pooling. Stacked equivariant message-passing layers consist of base GNNs for subgraphs, which serve as the primary components in subgraph GNNs. The t-th iteration of the base GNN process within the k-hop subgraph G_v^k of vertex v can be expressed as follows:

$$h_v^t = \sigma^{(t)}\left(h_v^{(t-1)}, \sum_{w \in \mathcal{N}^1_{G_v^k}(v)} h_w^{(t-1)}\right) \tag{1}$$

TABLE I
NOTATIONS OF SUBGRAPH GNNS

Notation	Meaning	Notation	Meaning
$\mathbf{G}$	graph $\mathbf{G} = (\mathcal{V}, \mathcal{E})$	$\mathcal{V}$	vertices of $\mathbf{G}$
$N_G^k(v)$	k-hop neighbors of v in G	v	vertex of $\mathbf{G}$
G_v^k	k-hop subgraph of v	$\mathcal{E}$	edges of $\mathbf{G}$
h_v	feature of vertex v	$h_{\mathbf{G}}$	representation of $\mathbf{G}$

TABLE II
DATA VOLUME OF ORIGINAL GRAPHS AND SUBGRAPHS.

Dataset	Num. Graphs	Sum Vertices	Num. Subgraphs	Sum Vertices (Subgraph)			
				1-hop	2-hop	3-hop	4-hop
MUTAG	188	3371	3371	10813	21669	33181	43385
BZR	405	14479	14479	45549	97819	164545	234879
ENZYMES	600	19578	19578	94144	193130	289884	375652
PROTEINS	1113	43471	43471	205559	435963	701931	979121

where $\sigma^{(t)}$ is an arbitrary continuous function, h_v^t is the feature of vertex v at t-th iteration and $\mathcal{N}^1_{G_v^k}(v)$ denotes the 1-hop neighbors of v in its k-hop subgraph. Intra-subgraph pooling involves pooling the vertex features within each subgraph to derive the corresponding subgraph representation.

Global phase generates the final graph representation through inter-subgraph pooling. Specifically, the pooling is executed across all subgraphs to combine their representations into the final graph representation. This process can be expressed as:

$$h_{\mathbf{G}} = P^G\left(\left\{h_{G_v^k} \mid v \in \mathbf{G}\right\}\right) \tag{2}$$

Here, P^G represents the graph pooling, $h_{G_v^k}$ is the subgraph representation and $h_{\mathbf{G}}$ is the final graph representation.

III. Architecture and Mapping of 3D-SubG

A. Overview of 3D-SubG Hardware Architecture

The overall architecture of 3D-SubG is depicted in Fig. 2. It consists of a DRAM die featuring a 6 × 6 block configuration, with a logic die densely integrated beneath it using uniformly distributed copper metal pads at the bond interface. Similarly, the logic die is segmented into 6 × 6 blocks, constrained by packaging technology. Each logic block is required to be vertically aligned with its respective DRAM block, enabling direct high-bandwidth data exchanges within this logic-to-DRAM (L2D) block. Additionally, the logic blocks are interconnected through an on-chip 2D mesh network, facilitating efficient data communication among all L2D blocks.

A DPIM-based GNN engine is implemented on the logic die within a L2D block to deliver efficient parallel computing capabilities. Each GNN engine is equipped with eight DPIM macros, segmented into four combination lanes for MM operations required in GNN combination calculations. The combination calculations are characterized by their regular and static memory access patterns suitable for DPIM macros. The engine also includes SIMD cores for handling irregular and dynamic memory accesses in aggregation tasks and other vector computations, such as pooling. To harness the inherent

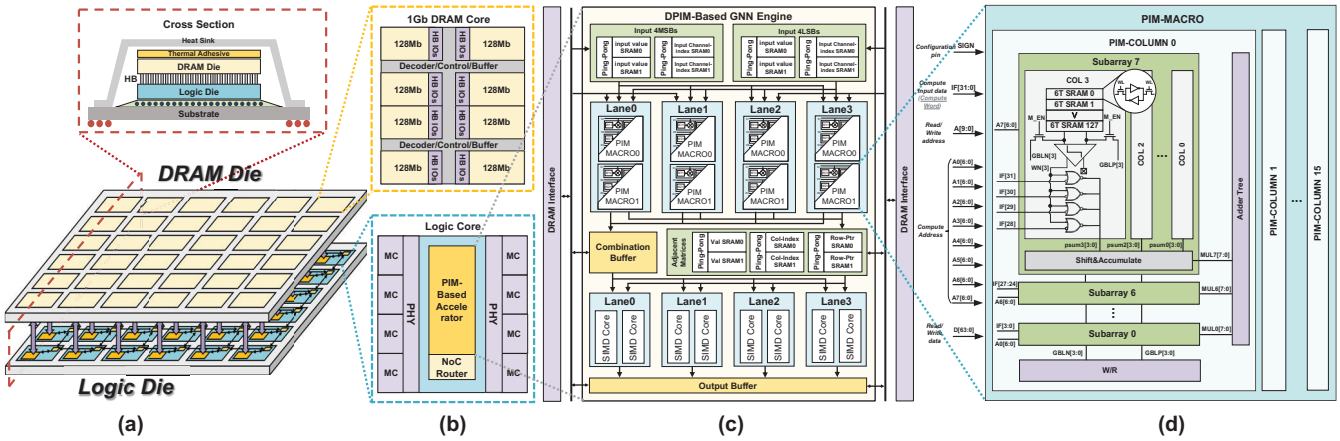


Fig. 2. An overview of the proposed 3D-SubG architecture: (a) the hybrid bonding PNM architecture, (b) a logic-to-DRAM block with a DRAM block and a logic block, (c) micro-architecture of DPIM-based GNN engine inside a logic block, and (d) the micro-architecture of DPIM.

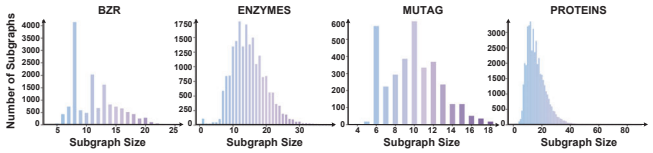


Fig. 3. The count of subgraphs (hop=3) of specific sizes across four datasets reveals an uneven distribution of subgraph sizes.

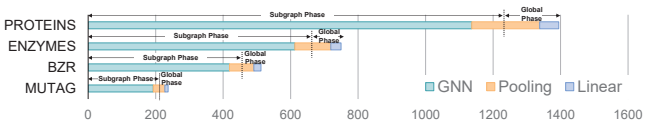


Fig. 4. Runtime breakdown of a subgraph GNN (NGNN) on four different datasets with over 95% attributed to base GNN and pooling.

sparsity of GNNs, a specialized channel-independent DPIM is used in conjunction with our bit-level non-zero (NZ) value gathering method depicted in Fig. 5. For aggregation, the Compressed Sparse Row (CSR) format is used for the adjacency matrix A with 4 flexible SIMD lanes performing sparse matrix multiplication to enhance efficiency in sparse data processing.

During the subgraph processing phase of subgraph GNN, workloads are distributed to L2D blocks at the granularity of subgraphs. This mapping strategy maximizes the parallel execution of subgraphs by leveraging the spatially distributed nature of L2D blocks. Specifically, the 6 × 6 array of L2D blocks enables concurrent processing of up to 36 subgraphs. Once the subgraph phase is completed, the central block collects the computational results from its neighboring blocks and proceeds to execute the final global pooling phase.

B. Sparse-Aware DPIM with Bit-Level Non-Zero Gathering

The primary operator of subgraph GNN is the base GNN, as depicted in Fig. 4. The input matrix X exhibits high sparsity, as evidenced in Table III. Traditional PIM macros, constrained by their rigid dataflow, often find it challenging to effectively

TABLE III
SPARSITY STATISTICS OF THE GNN PART IN NGNN [14]

Matrix		BZR	MUTAG	ENZYMES	PROTEINS
A		0.954	0.882	0.914	0.928
X	Element-wise	0.8562	0.4039	0.2017	0.1906
	4 MSBs	0.8945	0.4696	0.3570	0.2896
	4 LSBs	0.8739	0.5302	0.3190	0.3241
W		0.103	0.0167	0.0653	0.0341

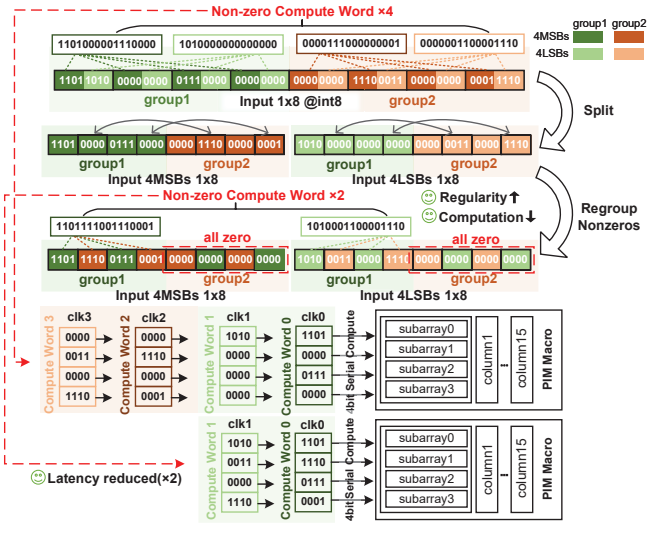


Fig. 5. The proposed bit-level non-zero gathering method for DPIM.

manage sparse data. As illustrated in Table III, among the two matrices involved in the combination stage, weight matrix W has very low sparsity and can be treated as dense, whereas input X displays significantly higher sparsity. Therefore, a weight-stationary DPIM can directly utilize W without additional optimization. The primary focus of optimization efforts should be on how to efficiently exploit the sparsity of X.

This work introduces a channel-independent digital PIM (DPIM) macro for sparse combination computations. Conven-

tional DPIMs are constrained by their reliance on a single subarray address signal, limiting them to selecting weights only from the same position within each subarray. Our DPIM macro, however, allows each input channel to have its unique subarray address signal. This enables concurrent feeding of data from multiple input channels into the DPIM macro for multiplication with different weights stored at various addresses within the subarrays. This design provides an opportunity to leverage sparsity by exchanging input data within the same input channel across different compute words. Different input channels can independently select weights from distinct subarray addresses, ensuring computational accuracy after data exchange. For clarity, we use the term "compute word" to describe the input data that a DPIM macro can process within one clock cycle, analogous to the "word" in SRAM, which denotes the data that can be written or read in a single clock cycle. Furthermore, our DPIM macro employs a 4-bit serial computing approach instead of the conventional 1-bit serial computing, significantly boosting its computing capability.

The bit-level non-zero (NZ) gathering method we propose is illustrated in Fig. 5. The principle of this method involves exchanging data among different compute words within the same input channel. This process gathers non-zero values into specific compute words, thereby increasing the number of all-zero compute words that can be bypassed during computation. It is important to note that compute words composed of different bits from the same data cannot be exchanged, as this would disrupt the proper functioning of the accumulator. Fig. 5 illustrates this method using a DPIM macro with a 4×4 compute word as an example. Before optimization, all four compute words need to be computed. However, after applying NZ gathering, only two compute words remain for processing. While element-level NZ gathering is effective, Table III demonstrates that higher bit-level sparsity makes bit-level NZ gathering even more advantageous, as will be further evidenced in the evaluation results.

C. Load-Balanced Mapping of Subgraphs onto L2D Blocks

In the subgraph phase, the workload is allocated to the L2D blocks at the subgraph granularity. The significant size differences among subgraphs, as presented in Fig. 3, can lead to an imbalanced workload across blocks during parallel computation. To address this issue, we introduce a balanced mapping strategy, as shown in Algorithm 1. It is based on the assumption that the workload of a subgraph generally correlates positively with its vertex count, although the irregular sparsity of the subgraph also plays a role. To estimate the workload of a subgraph, we consider its vertex count and employ a greedy algorithm for subgraph assignment. Initially, we sort the subgraphs in descending order based on their vertex counts. Subsequently, we sequentially assign each subgraph to a target L2D block. The target block is determined based on the number of tasks already assigned to each block. The block with the fewest tasks is the target block. After each subgraph assignment, the record of the block with the fewest tasks is updated, ensuring next subgraph to be assigned in a load-balanced manner.

Algorithm 1: Load-Balanced Mapping of Subgraphs

Input: Subgraph Array $SubG_{array}$, Adjacency matrices array A_{array}, L2D Block Array $Block_{array}$, Gragh Array G_{array}
Output: Block Workload Array BW_{array}

for *each graph* $g \in G_{array}$ **do**
  Update $SubG_{array}$, A_{array}
  for *each subgraph* $subg \in SubG_{array}$ **do**
    $Size[subg] = Number\ of\ rows\ in\ A_{array}[subg]$
  end
  Sort $SubG_{array}$ by $Size$ in descending order
  $Minblock$: the block with the fewest allocated workloads
  for *each subgraph* $subg \in SubG_{array}$ **do**
    for *each block* $b \in Block_{array}$ **do**
      if $BW_{array}[b] < BW_{array}[Minblock]$ **then**
        $Minblock = b$
      else
        Continue
      end
    end
    $BW_{array}[Minblock] = BW_{array}[Minblock] + Size[subg]$
  end
end

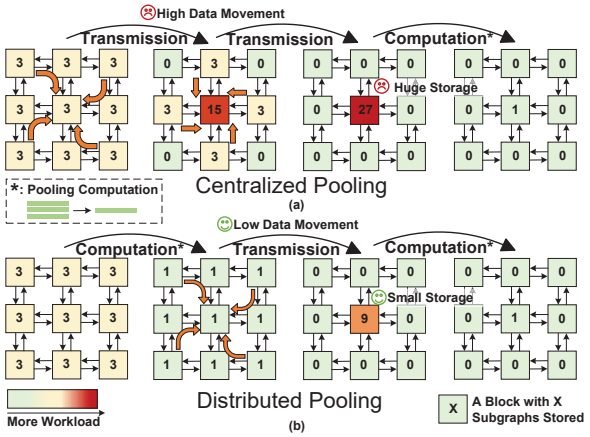


Fig. 6. (a) Centralized global pooling lead to massive data tranfer and imbalanced computation. (b) The proposed distributed pooling method reduces the inter-block data transfer and alleviates workload imbalance.

D. Distributed Global Pooling Approach

Although pooling is often overlooked, global pooling in subgraph GNNs involves significant inter-block data movement, making its optimization crucial. During the subgraph phase, subgraphs are distributed to different L2D blocks. Global pooling requires aggregating computational data from all subgraphs. The most straightforward approach is to use NoC to transmit all subgraph data to a central block, which then performs the final pooling. However, this approach incurs an $O(n)$ data transmission cost through the NoC, where n is the number of subgraphs. The Table II demonstrates that the

number of subgraphs is exceptionally large and continues to increase with the addition of hops. Furthermore, as illustrated in the Fig. 6(a), this method concentrates all computational loads on the central block, resulting in a highly unbalanced load distribution.

Considering the characteristics of pooling, local pooling can be performed within a block first, followed by global pooling. After local pooling, the data volume of each block is reduced to that of a single subgraph. Consequently, the total amount of data that needs to be transferred shifts from being proportional to number of subgraphs to being proportional to number of blocks, thereby significantly mitigating load imbalance and reducing data movement overhead, as shown in Fig. 6(b).

IV. Evaluation

A. Experimental Setup

Architecture Configuration. The specification of 3D-SubG is shown in Table IV, which features a configuration of 6×6 L2D blocks. Every logic block is equipped with eight PIM macros, each comprising a 1024×64-bit SRAM cell array. These SRAM cells are organized into an 8×16 subarray structure, capable of processing 8×16 INT8-by-INT4 operations every two clock cycles. Additionally, each logic block incorporates a total of 276 kB ping-pong SRAM buffers. The NoC on logic die employs a mesh topology with a 64-bit flit width. The DRAM specifications are based on previous research [24] and appropriately scaled to align with this work. The DRAM blocks are arranged in a 6×6 configuration, mirroring the layout of logic blocks. Each DRAM block has a capacity of 16 MB and provides a die-to-die bandwidth of 4.8 GB/s.

Hardware Implementation. The DPIM macro on logic die is evaluated using Cadence Virtuoso for area and power consumption. The on-chip SRAM is generated with ARM Artisan SRAM Compiler, which also supplies data on area and power. The remaining digital components are synthesized using Cadence Genus. The logic die is synthesized using commercial 22nm process, operating at clock frequency of 200 MHz and voltage of 0.8V. The energy cost of DRAM access is estimated as 0.88 pJ/bit [25]. The energy cost of NoC is estimated as 1.1 pJ/bit/hop [27]. Moreover, a C++ simulator is developed to assess the performance, data transfer rates, and utilization of the overall 3D-SubG system.

Benchmarks and Datasets. To comprehensively evaluate 3D-SubG, we deployed and evaluated the Nested GNN (NGNN) [14] on four widely-used datasets in the subgraph GNN domain: BZR [28], MUTAG [29], ENZYMES [30], and PROTEINS [31], all of which belong to TU datasets [32]. NGNN adheres to the general subgraph GNN structure. No specific optimizations were implemented, thereby enabling an unbiased evaluation of 3D-SubG's compatibility with the general subgraph GNN framework. For the model configuration, we adopted the setup outlined in [14], utilizing NGNN with GCN as the base GNN, a hop value of 3, and a hidden dimension of 32. We trained the floating-point model and then quantized it, where the adjacency matrix A and weights W were quantized to INT4, while the graph representation matrix X and model outputs were quantized to INT8. We demonstrated that this quantization scheme has a negligible impact on model accuracy across all four datasets examined, as shown in Table V.

TABLE IV
Specifications of Logic Die and DRAM Die

Logic Die	Specification
Area	98.22 mm^2
Technology	22nm
Block	6×6 blocks
DPIM	12.8GOPS/macro (int8 mul int4)
SRAM Capacity	276kB + 64kB(DPIM) / block
NoC	mesh, 64-bit flit width
Frequency	200MHz
Voltage	0.8V

DRAM Die	Specification
Area	98.22 mm^2 (DRAM 72 mm^2 + Peri 26.22mm^2)
Technology	25nm
Capacity	16MB/block
Bandwidth	4.8GB/s / block
Frequency	150MHz
Voltage	1.1V

TABLE V
Quantization precision testing

Dataset	Claimed in [14] (%)	Floating Point[1](%)	Quantized[1](%)	Accuracy Drop (%)
BZR	–	79.5±3.9	79.0±3.3	0.6
MUTAG	82.9 ± 11.1	81.1 ± 8.3	81.1 ± 8.3	0.0
ENZYMES	31.2 ± 6.7	34.2 ± 12.5	33.8 ± 12.1	1.3
PROTEINS	73.3 ± 4.0	71.8 ± 5.7	71.8 ± 5.7	0.0

[1] The training process employed the 10-fold cross-validation method.

Baselines. To the best of our knowledge, 3D-SubG is the pioneering accelerator tailored for subgraph GNNs. For fairness, we refrain from comparing it with conventional GNN accelerators, whose target task is traditional GNNs. To validate the efficacy of the proposed techniques, we utilize an unoptimized variant of 3D-SubG as the benchmark for comparison. Lastly, we also conducted a comparative analysis of latency, power consumption, and area efficiency with an NVIDIA RTX 3090Ti utilizing PyTorch Geometric.

B. Evaluation Results

Efficiency Evaluation for Bit-Level Non-Zero Gathering. As in Fig. 7, bit-level NZ gathering achieves an average reduction of 2.5×, 1.5×, 1.53×, and 1.93× compute words on BZR, MUTAG, ENZYMES, and PROTEINS, respectively, with the reduction directly proportional to DPIM's computation time. Additionally, we compared bit-level versus element-level NZ gathering, observing reductions of 1.38×, 1.08×, 1.13×, and 1.12× across the datasets. The technique's effectiveness depends on data sparsity and distribution, with BZR showing the greatest improvement due to its highest sparsity as in Tabel III.

Efficiency Evaluation for Load-Balanced Subgraph Mapping. The benefits of load-balanced subgraph mapping are shown in Fig. 8. On the BZR, MUTAG, ENZYMES, and

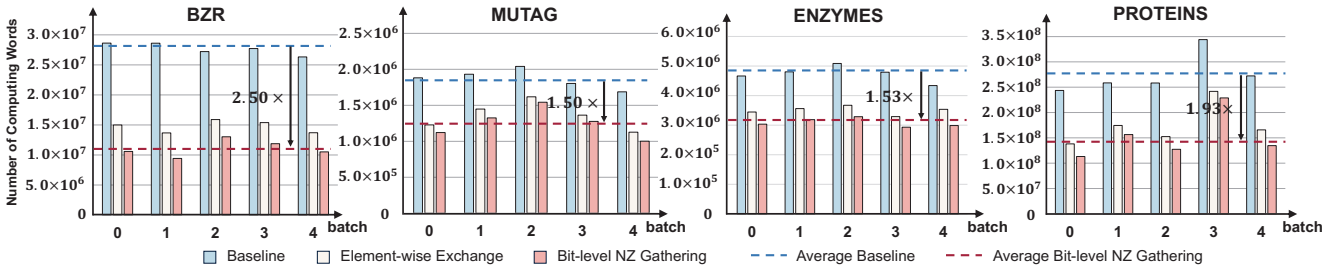


Fig. 7. Number of compute words for DPIM on four datasets.

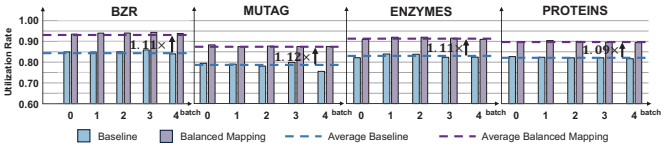


Fig. 8. Average utilization rate of L2D blocks in the subgraph phase.

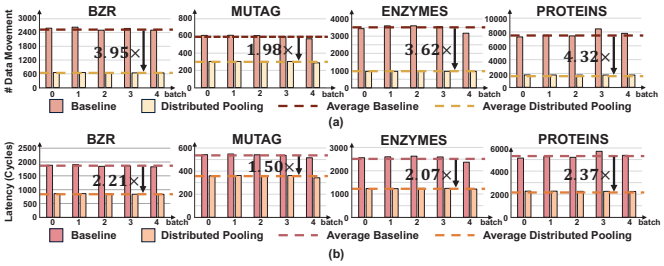


Fig. 9. (a) Amount of data transferred on NoC during the global phase; (b) The latency of global pooling.

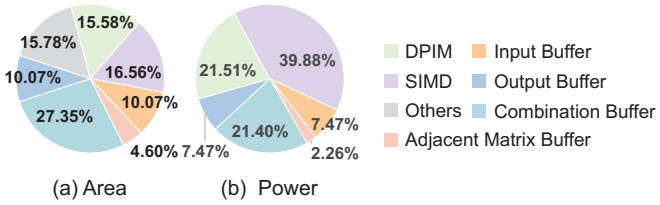


Fig. 10. Area and power breakdown of logic die.

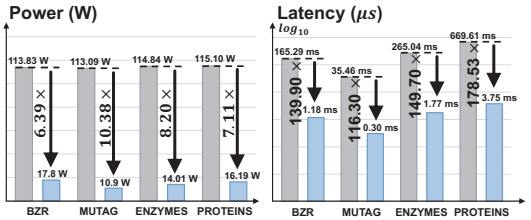


Fig. 11. Comparison results of 3D-SubG with RTX 3090Ti GPU.

PROTEINS datasets, this technique improved the L2D block utilization by 1.11×, 1.12×, 1.11×, and 1.09×, respectively, increasing the overall utilization from around 80% to approximately 90%.

Efficiency Evaluation for Distributed Global Pooling. Distributed global pooling theoretically diminishes the data transfer required for global pooling from $O(n)$ to $O(m)$, where n and m denote the number of subgraphs and L2D blocks, respectively. Experiments confirm this, indicating that on the BZR, MUTAG, ENZYMES, and PROTEINS datasets, distributed pooling attains reductions in inter-block data movement by 3.95×, 1.98×, 3.62×, and 4.32×, respectively, as shown in Fig. 9. By distributing the computational load across all blocks and markedly decreasing data transfer, the total latency of global pooling is also decreased, achieving reductions of 2.21×, 1.50×, 2.07×, and 2.37× on four datasets.

Area and Power Overhead. The implementation results show that the total area of the logic die is 98.22mm^2, with a total power consumption of 4.17W. The area and power breakdowns are shown in Fig. 10.

Overall Performance. We compare 3D-SubG with RTX 3090Ti GPU using PyTorch Geometric, with results presented in Fig. 11. This comparison includes power consumption related to die-to-die and NoC data transmission, as well as DRAM static power consumption. Across four datasets, 3D-SubG outperforms GPU in latency performance by 139.90×, 116.30×, 149.70×, and 178.53×, respectively. Simultaneously, it achieved significant reductions in power consumption by 6.39×, 10.38×, 8.20×, and 7.11×. On average, 3D-SubG exhibited 146.11× improvement in performance, 1171.80× increase in energy efficiency, and 934.18× enhancement in area efficiency compared to RTX 3090Ti GPU.

V. Conclusion

We develop a 3D stacked accelerator, named 3D-SubG, tailored for subgraph GNNs. 3D-SubG is built upon a processing-near-DRAM architecture enabled by hybrid bonding packaging to achieve parallel memory accesses, where the DRAM die is 3D stacked with logic die equipped with PIM macros. For the subgraph phase and global phase in subgraph GNNs, we have proposed load-balanced subgraph mapping and distributed global pooling strategies, respectively. To efficiently manage sparse computations to benefit PIM, we introduced a bit-level non-zero gathering method. On average, 3D-SubG outperforms RTX 3090Ti GPU by 146.11× in performance, 934.18× in area efficiency, and 1171.80× in energy efficiency.

REFERENCES

[1] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *NeurIPS*, vol. 31, 2018.

[2] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *ICML*. PMLR, 2019, pp. 3835–3845.

[3] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, 2018, pp. 974–983.

[4] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," *NeurIPS*, vol. 30, 2017.

[5] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *KDD*, 2019, pp. 750–760.

[6] M. Haggai, B. Heli, S. Nadav, and L. Yaron, "Invariant and equivariant graph networks," in *ICLR*, 2019.

[7] X. Keyulu, H. Weihua, L. Jure, and J. Stefanie, "How powerful are graph neural networks?" in *ICLR*, 2019.

[8] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *AAAI*, vol. 33, no. 01, 2019, pp. 4602–4609.

[9] Z. Lingxiao, J. Wei, A. Leman, and S. Neil, "From stars to subgraphs: Uplifting any gnn with local structure awareness," in *ICLR*, 2022.

[10] A. Leman and B. Weisfeiler, "A reduction of a graph to a canonical form and an algebra arising during this reduction," vol. 2, no. 9, 1968, pp. 12–16.

[11] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" *NeurIPS*, vol. 33, 2020.

[12] V. Arvind, F. Fuhlbruck, J. Kobler, and O. Verbitsky, "On weisfeiler-leman invariance: Subgraph counts and related graph properties," *Journal of Computer and System Sciences*, vol. 113, pp. 42–59, 2020.

[13] D. C. Elton, Z. Boukouvalas, M. D. Fuge, and P. W. Chung, "Deep learning for molecular design—a review of the state of the art," *Molecular Systems Design & Engineering*, vol. 4, no. 4, pp. 828–849, 2019.

[14] M. Zhang and P. Li, "Nested graph neural networks," *NeurIPS*, vol. 34, pp. 15 734–15 747, 2021.

[15] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," in *ICLR*, 2022.

[16] L. Cotta, C. Morris, and B. Ribeiro, "Reconstruction for powerful graph representations," in *NeurIPS*. Curran Associates Inc., 2024.

[17] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 10 737–10 745.

[18] F. Frasca, B. Bevilacqua, M. Bronstein, and H. Maron, "Understanding and extending subgraph gnns by rethinking their symmetries," *NeurIPS*, vol. 35, 2022.

[19] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *HPCA*, 2020.

[20] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt *et al.*, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing," in *MICRO*, 2020, pp. 922–936.

[21] R. Hwang, M. Kang, J. Lee, D. Kam, Y. Lee, and M. Rhu, "Grow: A row-stationary sparse-dense gemm accelerator for memory-efficient graph convolutional neural networks," in *HPCA*, 2023, pp. 42–55.

[22] Z. Zhu, F. Li, G. Li, Z. Liu, Z. Mo, Q. Hu, X. Liang, and J. Cheng, "Mega: A memory-efficient gnn accelerator exploiting degree-aware mixed-precision quantization," in *HPCA*, 2024, pp. 124–138.

[23] Z. Yue, H. Wang, J. Fang, J. Deng, G. Lu, F. Tu, R. Guo, Y. Li, Y. Qin, Y. Wang, C. Li, H. Han, S. Wei, Y. Hu, and S. Yin, "Exploiting similarity opportunities of emerging vision ai models on hybrid bonding architecture," in *ISCA*, 2024.

[24] D. Niu, S. Li, Y. Wang, W. Han, Z. Zhang, Y. Guan, T. Guan, F. Sun, F. Xue, L. Duan, Y. Fang, H. Zheng, X. Jiang, S. Wang, F. Zuo, Y. Wang, B. Yu, Q. Ren, and Y. Xie, "184qps/w 64mb/mm23d logic-to-dram hybrid bonding with process-near-memory engine for recommendation system," in *ISSCC*, vol. 65, 2022.

[25] Y. Chi *et al.*, "16.4 an 89tops/w and 16.3 tops/mm2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *ISSCC*, vol. 64, 2021, pp. 252–254.

[26] A. Guo, X. Si, X. Chen, F. Dong, X. Pu, D. Li, Y. Zhou, L. Ren, Y. Xue, X. Dong *et al.*, "A 28nm 64-kb 31.6-tflops/w digital-domain floating-point-computing-unit and double-bit 6t-sram computing-in-memory macro for floating-point cnns," in *ISSCC*, 2023, pp. 128–130.

[27] J. Wang, H. Du, B. Ding, Q. Xu, S. Chen, and Y. Kang, "Ddam: Data distribution-aware mapping of cnns on processing-in-memory systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 3, 2023.

[28] J. J. Sutherland, L. A. O'Brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships," *Journal of chemical information and computer sciences*, vol. 43 6, pp. 1906–1915, 2003.

[29] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *ICML*, 2012, p. 291–298.

[30] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg, "Brenda, the enzyme database: updates and major new developments," *Nucleic Acids Research*, vol. 32, no. suppl_1, 01 2004.

[31] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, pp. i47–i56, 2005.

[32] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *ArXiv*, vol. abs/2007.08663, 2020.

SynCircuit: Automated Generation of New Synthetic RTL Circuits Can Enable Big Data in Circuits

Shang Liu†, Jing Wang†, Wenji Fang, Zhiyao Xie*
Hong Kong University of Science and Technology
{sliudx, jwangjw, wfang838}@connect.ust.hk, eezhiyao@ust.hk

***Abstract*—In recent years, AI-assisted IC design methods have demonstrated great potential, but the availability of circuit design data is extremely limited, especially in the public domain. The lack of circuit data has become the primary bottleneck in developing AI-assisted IC design methods. In this work, we make the first attempt, SynCircuit, to generate new synthetic circuits with valid functionalities in the HDL format.**

SynCircuit automatically generates synthetic data using a framework with three innovative steps: 1) We propose a customized diffusion-based generative model to resolve the Directed Cyclic Graph (DCG) generation task, which has not been well explored in the AI community. 2) To ensure our circuit is valid, we enforce the circuit constraints by refining the initial graph generation outputs. 3) The Monte Carlo tree search (MCTS) method further optimizes the logic redundancy in the generated graph. Experimental results demonstrate that our proposed SynCircuit can generate more realistic synthetic circuits and enhance ML model performance in downstream circuit design tasks.

I. Introduction

The ever-increasing demands for chip performance have caused escalating integrated circuit (IC) complexity, challenging traditional Electronic Design Automation (EDA) methodologies. In recent years, AI-assisted IC design techniques have demonstrated remarkable potential in accelerating the chip design process. Notable AI for EDA applications include automated chip design generation [1], [2], [3], fast chip quality prediction [4], [5], [6], [7], [8], [9], [10], and automated chip design planning [11].

SynCircuit: Pathway to Big Data in Circuits. Compared with generating datasets with limited circuits in the public domain, we believe the automated generation of a large number of *synthetic* circuits is the most promising way to *completely solve* the circuit data availability problem in the foreseeable future. In this work, we demonstrate the feasibility of this promising direction with a novel synthetic circuit generation framework named SynCircuit.

To the best of our knowledge, SynCircuit proposed in this work is the first technique to automatically generate brand-new synthetic circuits with valid functionalities. Specifically, we achieve this with novel customized graph learning algorithms and follow-up refinement techniques. The synthetic circuits are in the RTL stage and support hardware description language (HDL) code format. They can be automatically synthesized into regular netlists and turned to layouts with commercial tools. These synthetic designs can not only enhance the training and robustness of AI models by providing diverse and extensive datasets but also may serve as benchmarks for assessing the performance of existing design algorithms [12], [9], [13]. In addition, our proposed directed graph generation framework potentially applies to other tasks, such as neural architecture search (NAS) and Bayesian optimization [14], [15], [16].

SynCircuit generates synthetic circuits with 3 phases:

Phase 1. Generation of Large Directed Cyclic Graph (DCG). Circuit design code (i.e., HDL code) can be equivalently represented as directed cyclic graphs (DCG), allowing the utilization of graph generation algorithms to create synthetic circuit designs. However, most machine learning (ML) research on graph generation has focused on undirected graphs [17], [14], [18], [19], [20], leaving unsolved challenges on generating *directed graphs*. For instance, edge predictions in these prior works are determined by applying a *symmetric operator* to the pairwise node embeddings. Also, spectrum-based generation techniques like Spectre [21] and DiGress [22] rely on properties specific to undirected graphs, such as *symmetric Laplacian matrices*. Some works generate directed graphs in an autoregressive manner following the topological order [15], [23]. However, they do not apply to directed *cyclic* graphs, where a topological sort does not exist. In summary, the generation of DCG is not a well-solved problem, not only for synthetic circuit data generation in EDA applications, but also in the general AI community.

† Equal Contribution
∗ Corresponding Author

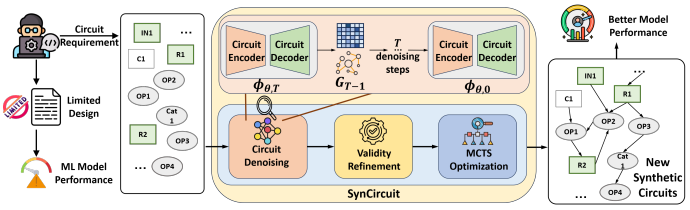


Fig. 1: The overview of SynCircuit. It addresses the serious design scarcity issue by generating new synthetic circuits. Through a three-stage process consisting of denoising, validity refinement, and optimization, SynCircuit can generate an unlimited number of synthetic circuits suitable for downstream ML tasks.

In response to the challenges posed by large directed cyclic graphs in circuit designs, we propose a graph generation algorithm for digital circuits. We employed a diffusion generation framework for graph generation and designed an efficient and low-cost denoising network that includes an encoder and a decoder. For the graph encoder, a directed messaging-passing neural network is utilized to better capture the local features of graphs while only requiring a relatively low memory consumption. For the decoder, we model the directed edge relation based on learnable translated embeddings.

Phase 2 & 3. Refinement of Synthetic Circuits. Additionally, how to ensure the graph validness, i.e., satisfying the circuit constraints, is not also fully explored in graph generation, especially in a one-shot manner. Given the fanin constraints and combinational loop prevention requirements, we propose a probability-guided graph post-processing strategy to further refine the invalid synthetic graph obtained from the denoising process. This technique sequentially processes the fan-in edges of each node, which not only ensures that the resulting circuit satisfies the constraints but also preserves the generative information of the diffusion model as much as possible.

We also point out the logic redundancy problem in the synthetic circuits. To further alleviate the redundancy of the generated designs and improve their utility as data augmentation in downstream tasks, we also employ Monte Carlo Tree Search (MCTS) to further fine-tune the generated graphs. This approach allows us to obtain higher-quality graphs that are better suited for real applications.

We summarize our contributions as follows:

- To the best of our knowledge, we are the first to propose a framework for generating digital circuits at the RTL level from a graph generation perspective. This framework comprises three components in the generation phase: directed cyclic graph generation, probability-guided postprocessing, and MCTS-based optimization.
- We propose a graph generative model solution targeting the challenging directed cyclic graph generation task. We utilize the diffusion framework and reconstruct the edge direction information through an asymmetric edge decoder.

- To further alleviate the redundancy of the generated designs and improve their utility as data augmentation in downstream tasks, we employ MCTS to further fine-tune the generated graphs. This approach obtains higher-quality graphs with optimized logic redundancy which are better suited for real applications.
- We conducted experiments on graph structural property similarity evaluation and demonstrated that SynCircuit generates more realistic graphs. Experiments on early RTL-stage PPA prediction show that SynCircuit can help alleviate the data availability problem in AI-based solutions for EDA tasks.

II. Problem Formulation

DCG representation of HDL code. Given a circuit design HDL code D, it can be mapped to a circuit graph representation G through a bijection function $f : D \leftrightarrow G$ using our developed parser. Here, G is a directed cyclic graph represented as (V, E, X), where V denotes the set of nodes, E represents the set of edges with $e_{i,j} \in E$ indicating a directed edge from v_i to v_j, and X represents the attributes of the nodes. The node attributes include node *type* and *width*. The node types are primarily categorized into IO port, arithmetic operator, register (reg), bit selection, and concatenate operator. The width attribute reflects the output signal width of the corresponding node.

Circuit constraints $\mathcal{C}$. To ensure that the graph can be parsed back into HDL code, valid G must satisfy two types of constraints:

- The node type uniquely determines the number of parent nodes. For example, a node of the type "mux" requires three parent nodes, while the type "add" requires two.
- The graph must not contain any combinational loops. The combinational loop is defined as a cycle that does not include any registers, which would cause timing violations.

Generate new large circuits. In order to provide generation flexibility, we will use the generative model to produce edges E conditioned on the specified node number V and attributes X by users. Our objective is to learn the probability distribution $P(G \mid V, X)$ from a set of real circuit HDL codes through their graph representations $\{G_i\}$. This will enable the generation of a series of new synthetic circuit graphs $\{G_i^{syn}\}$. We aim for these synthetic circuit design graphs to closely resemble real designs in terms of topological structure and satisfy all the circuit constraints $\mathcal{C}$. Moreover, considering most registers in real designs will not be removed in the synthesis stage, we require the generated circuits to have as low logic redundancy as possible.

III. Overview of SynCircuit

Initially, the realistic designs are converted into directed cyclic graph representation. We subsequently trained a diffusion model $P(G \mid V, X)$ on these designs. In the generation stage, as Figure 2 shows, our proposed circuit generation process consists of three phases: initial graph generation, validness post-process, and optimization refinement.

$$P(G) \xrightarrow{\textcircled{1}} G^{ini} \xrightarrow{\textcircled{2}} G^{val} \xrightarrow{\textcircled{3}} G^{opt}$$

In the generation phase of ❶, we adopt the trained diffusion model and apply the denoising process to obtain an initial synthetic graph G^{ini} with a corresponding edge probability matrix $P_E^{(t=0)}$. The value of (i, j) in P_E represents the directed edge $e_{i,j}$ existence probability. In phase ❷, we propose an autoregressive method to determine the edges based on the G^{ini} to ensure that the refined graph G^{val} satisfies all the circuit constraints $\mathcal{C}$. In the final phase ❸, we address the issue of significant logical redundancy in the generated circuits. G^{val} is fine-tuned by the MCTS-based method, reducing logic redundancy and thereby enhancing the performance of downstream tasks. In the following subsections, we will explain each stage in detail.

IV. Phase 1. DCG Generation Technique

Limitation of prior works on DCG generation. Many prior graph generation works utilize a symmetric operator or semi-positive-definite Laplacian matrix to assign edges. This symmetric assumption does not support the asymmetric edge in the directed cyclic graph. For directed acyclic graphs (DAGs), previous work [23], [15] addresses DAG generation by sorting node order based on the topological structure and then generating nodes in a layerwise manner. Topological sorting ensures that a node's parent nodes always precede it in the order. Consequently, the edge direction is automatically determined. However, there is no topological node ordering in the DCG graph since the nodes in a loop share the same topological level.

The proposed DCG generation technique overview. To the best of our knowledge, we are the first to address the task of generating directed cyclic graphs using a diffusion model framework. The method involves a forward diffusion process and a reverse denoising process. In the **forward process**, we gradually corrupt the adjacency matrix A by adding noise, transforming it from the original data distribution into a noisy distribution. The goal of the reverse process is to reconstruct the original adjacency matrix from the noisy version by reversing this corruption. During the **reverse denoising process**, a denoising neural network ϕ_θ with an *encoder* and a *decoder* is trained with real circuits. The *encoder* processes the node attributes and the noisy adjacency matrix to generate node representations that consist of rich graph structural information at each denoising step. The *decoder* utilizes these node representations to effectively reconstruct the ground truth edge connections. By training ϕ_θ in the reverse process, the model learns to progressively denoise the graph structure, ultimately generating synthetic circuits similar to realistic ones.

A. Forward Process

In the forward diffusion process, we progressively corrupt the adjacency matrix A by applying a series of predefined transition matrices, introducing noise at each time step t. Specifically, starting from the initial graph $G^{(0)} = (X, A^{(0)})$, the corrupted adjacency matrix $A^{(t)}$ at time t is obtained via a Markov process:

$$A^{(t)} = A^{(t-1)}\tilde{Q}_A^{(t)}$$

where $\tilde{Q}_A^{(t)}$ is the noise transition matrix at time t. By iteratively applying these transitions, we gradually transform $A^{(0)}$ into a noisy distribution. To control the degree of corruption, we employ a scheduling strategy (e.g., cosine schedule [24]) to adjust the noise level over time. The scheduling ensures that the corrupted adjacency matrices $A^{(t)}$ smoothly transit from the original data distribution to a predetermined noise distribution.

B. Denoising Process

In the denoising diffusion process, our goal is to recover the original adjacency matrix $A^{(0)}$ from the noisy matrix[1] $A^{(T)}$. Based on the node attributes[2] X, we model the conditional probability $p_\theta(G^{(t-1)}) = p_\theta(A^{(t-1)}|A^{(t)}, X, t)$, parameterized by a denoising network $\phi_{\theta,t}$. In the generation stage, the $\phi_{\theta,t}$ will reconstruct a probability matrix P_E^{t-1} given $G^{(t)}$. The new graph G^{t-1} will be updated by sampling the edge existence probability matrix P_E^{t-1}.

C. Encoder Design for Large Graphs

In the denoising network architecture $\phi_{\theta,t}$, the encoder is to capture the structural information of the input noisy graph $G^{(t)}$. Handling large-scale graphs with more than 10K nodes poses computational challenges. To address this, we design an efficient encoder based on Message Passing Neural Networks (MPNNs), which have a computational complexity linear to the number of edges $|E|$.

The encoder generates the representation for each node. When initializing the node embeddings, we use an MLP to obtain a time embedding for the current step t and combine it with the node attributes. By introducing the time step information t which conditions the encoder at different noise levels, we can improve the denoising

[1] In training, the $A^{(T)}$ is obtained from the forward process. In inference, we randomly sample a completely noisy adjacency matrix

[2] In training, the X is obtained directly from the real circuits. In inference, we can either use the $P(X)$ distribution from the real design or set it according to the user's specifications.

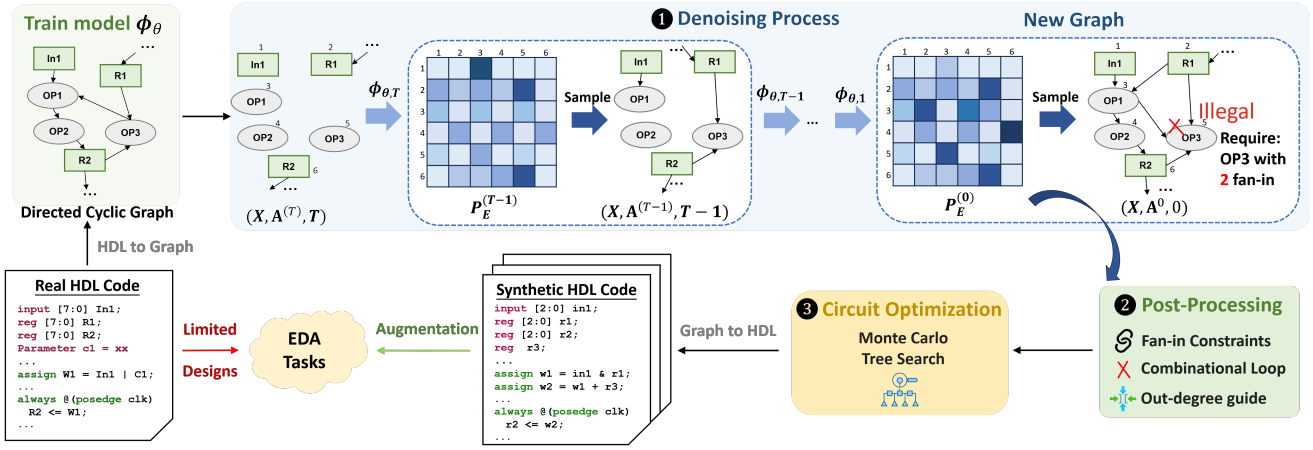


Fig. 2: Overview of SynCircuit framework. The circuit generation process starts with the specified node number V and attributes X. In **Phase 1**, the model performs a reverse diffusion process, where it predicts the probability of an edge existing between any pair (i, j) to obtain $P_E^{(t)}$. Then we sample from $P_E^{(t)}$ to update a graph $G^{(t)}$ as a new time state. After T denoising steps, the $G^{ini} = G^{(0)}$ is obtained. However, the generated graph may not necessarily be valid. In **Phase 2**, during post-processing, we sequentially assign parents to each node based on $P_E^{(t=0)}$, ensuring at each step that the circuit satisfies the constraints. In **Phase 3**, considering that some synthetic circuits exhibit significant logic redundancy, we employ an MCTS-based optimization technique, thereby reducing the generated design redundancy and better serving the downstream tasks.

performance at different diffusion steps. The node representation of each node j is updated through several layers of message passing:

$$H_j^{l+1} = \sigma\left(W_h^l H_j^l + \sum_{i \in \mathcal{P}_j} \frac{1}{|\mathcal{P}(j)|} W_m^l H_i^l\right)$$

where H_j^l is the representation of node j at encoder layer l, $\mathcal{P}(j)$ denotes the parents of node j, W_h^l and W_m^l are learnable weight matrices of the MPNN-based encoder, and σ is Relu function.

D. Decoder Design

In the denoising model $\phi_{\theta,t}$, the decoder will decide the existence of directed edges between pairs of nodes based on their encoded representations H_i from the encoder. In the context of directed graphs, conventional symmetric operations are insufficient to distinguish the directionality of edges. For example, the dot product [25] or Euclidean distance [15] of node embeddings H_i, H_j are commutative (i.e., $H_i^\top H_j = H_j^\top H_i$ and $\|H_i - H_j\|_2 = \|H_j - H_i\|_2$), failing to differentiate an directed edge from node i to j or from node j to i.

To address this limitation, we model the directed edge $e_{i,j}$ by associating H_i and H_j with a learnable relation embedding $r(t)$ [26], capturing the inherent directionality of the edge. In an asymmetric relation, the fundamental idea is that the embedding of the source node H_i, when translated by the relation embedding $r(t)$, is supposed to be close to the embedding of the target node H_j. We define the decoder's prediction for the existence probability of $e_{i,j}$ at time step t as follows:

$$\begin{aligned} P_E^{(t-1)}(i,j) = \hat{p}_{ij}^{(t-1)} &= p_\theta\left(A_{ij}^{(t-1)} = 1 \mid H_i^{(t)}, H_j^{(t)}, t\right) \\ &= \mathrm{MLP}\left(\left\{\left(H_i^{(t)} + r(t)\right) \odot H_j^{(t)}\right\} \oplus d(t)\right) \end{aligned}$$

where $H_i^{(t)}$ and $H_j^{(t)}$ are the representations of nodes i and j at time step t from the encoder, $r(t)$ is the learnable relation embedding obtained by $r(t) = MLP_r(t)$, $\odot$ denotes element-wise multiplication, $\oplus$ denotes vector concatenation, $d(t)$ is the learnable embedding of the time step t obtained by $d(t) = MLP_d(t)$.

V. Phase 2. Probability-Guided Graph Post-Processing

In phase 1, we obtain the G^{ini} and the edge existence probability $P_E^{(t=0)}$. But the G^{ini} will mostly likely violate circuit constraints $\mathcal{C}$. Therefore, we need to utilize the edge existence probability $P_E^{(t=0)}$ to refine G^{ini} and produce designs G^{val} that meet all predefined circuit constraints. We propose a probability-guided sequential post-processing approach to leverage the $P_E^{(t=0)}$ for parent node selection while ensuring both fanin limitations and combinational loop prevention which are required by constraints $\mathcal{C}$.

The post-processing algorithm iteratively processes each node in the circuit graph. For each node i, if its parent edges in G^{ini} are valid, then we will skip this node. If this node violates $\mathcal{C}$, we leverage the predicted edge probabilities from the previous diffusion model, sorting potential parent nodes by their connection probabilities in descending order. Before adding an edge from a candidate parent j, we need to check if the combinational loops would be created by this new edge. It can be achieved by checking if there exists a path from i to j in the subgraph that excludes register-type nodes. This sequential processing continues until the required number of valid parents is found.

VI. Phase 3. Refinement on Circuit Redundancy

Logic redundancy problem. Another important yet rarely studied challenge is logic redundancy. In real circuits, human designers will not likely introduce too much redundant logic into the circuit, which can be reflected by the sequential cell count in the optimized netlist. We define a metric, sequential cell preservation ratio (SCPR) which is calculated by dividing the number of sequential cells in the synthesized netlist by the total number of bits in sequential signals in the pre-synthesis HDL design. Our experiments show that the SCPR is usually between 70% to 100% in real designs. But some synthetic designs G^{val} can not even reach a 10% SCPR, which shows a serious logic redundancy problem.

Our objective in ❸. To better align the generated circuits with real ones not only in the topological aspects but also in the redundancy perspective, we aim to refine G^{val} and obtain G^{opt}, thus reducing the ratio of removed registers by the synthesis tool.

A. Challenges and optimization overview.

To optimize synthesized circuit designs, we face the challenges of an immense exploration space. To tackle this issue, we optimize the driving cone for the target register[3] one by one. In the cone optimization procedure, we introduce an MCTS-based effective strategy for navigating the large design space and finding circuits with better logic preservation quality.

[3]The term "driving cone for a register" refers to the set of nodes obtained by performing a reverse breadth-first search starting from a register node. This search traces back through the parent nodes until nodes of type "const", "in", or other "reg" nodes are encountered. The nodes traversed during this process collectively constitute the driving cone for our target particular register.

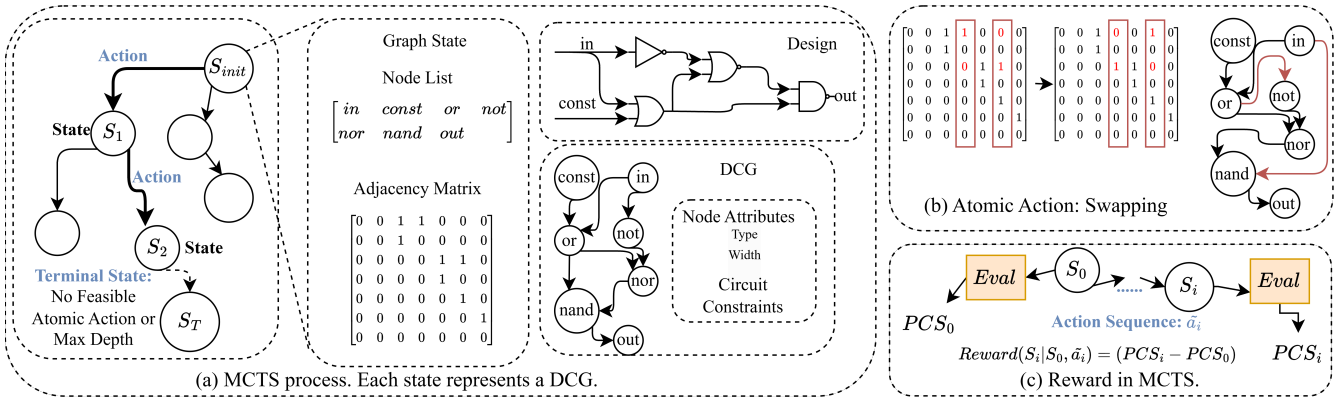


Fig. 3: MCTS-based circuit optimization flow overview. (a) The search tree is on the left and each state node represents an adjacency matrix. (b) A swapping atomic operation on the adjacency matrix facilitates state transformations during the search. (c) We propose a new metric, post-synthesis size (PCS) as the reward model to guide the search process.

B. Monte Carlo tree search-based exploration.

We detail the MCTS process in Figure 3. It guides the exploration and optimization of the design space, driving the search toward solutions with reduced hardware redundancy. Within this framework, each state node represents an adjacency matrix. We define a swapping atomic operation on the adjacency matrix, facilitating state transformations during the search.

Hardware redundancy metric as MCTS reward. To reflect the cone redundancy, we defined the post-synthesis circuit size (PCS). It is calculated by dividing the post-synthesis area by the number of nodes before synthesis. The rationale is that a larger PCS indicates fewer components have been optimized out, implying less redundancy in the synthetic circuit design. The overarching goal of the MCTS is to maximize the PCS while preserving the validity of the circuit.

Action space. The action space for transforming the circuit design during the MCTS process is defined by our proposed atomic swapping operation. For State S_i with $A(i,j) = 1$ and $A(p,q) = 1$, the new state S_{i+1} can be obtained by simply swapping the parents of node j and q, i.e., S_{i+1} with $A(p,j) = 1$ and $A(i,q) = 1$. Each atomic action should be checked if it violates $\mathcal{C}$. The advantage of this atomic operation lies in its ability to ensure that the number of edges in the graph remains unchanged, while also maintaining the out-degree and in-degree of each node in G^{val}.

Selection and expansion. In the selection phase, the MCTS algorithm traverses down the tree, selecting the most promising state which also meets circuit constraints $\mathcal{C}$ to expand based on the UCB1 criterion [27]:

$$a = \arg\max_{a} \left[Q(S,a) + \sqrt{2} \cdot \sqrt{\frac{\ln N(S)}{N(S,a)}} \right]$$

where $Q(S,a)$ is the empirical mean reward of action a in state S, $N(S)$ is the total number of visits to state S, $N(S,a)$ is the number of times action a has been chosen in state S. This formula balances the exploitation of known high-search reward actions with the exploration of less-tried actions. During the expansion step, the MCTS algorithm generates new child nodes by applying the atomic operation to the current node.

Simulation and backpropagation. Due to our objective of identifying the optimal state encountered during the search trajectory rather than optimizing for terminal states, we modify the traditional MCTS simulation and backpropagation mechanisms. In our approach, the reward for each simulation is defined as the maximum state reward $Reward_{max}$ encountered during that search path, rather than the terminal state value. During backpropagation, we update $Q(S,a)$ [27] using the $Reward_{max}$.

VII. Experimental Results

Application of SynCircuit in our experiments. The register-transfer level (RTL) stage offers maximum optimization flexibility in VLSI flow. Recently, machine learning-based RTL-level Power, Performance, and Area (PPA) prediction methods have been proposed, which can directly predict the performance of designs without requiring logic synthesis [6] [5] [28], thereby significantly accelerating the design iteration process. However, the application of these methods is limited by the insufficient availability of open-source RTL code.

Experiments objectives. In this experiment, our primary objective is to explore the effectiveness of new synthetic circuits for PPA modeling at the RTL level. Additionally, we are also interested in investigating the differences between synthetic circuits and real designs from not only the graph structure but also the logic redundancy perspectives. The comprehensive evaluation for SynCircuit may help us gain more insights into the digital circuit generative model.

A. Experimental Setup

Circuit design preparation. Firstly, we developed a dataset comprising 22 designs based on open-source RTL. The detailed dataset information is provided in Table I. It encompasses a diverse range of digital circuit modules, representing some of the high quality designs available within the open-source community.

Source Benchmark	#. of Designs	Original HDL Type	Design Scale (#K Gates) {Min, Median, Max}
ITC'99[29]	6	VHDL	{9, 19, 45}
OpenCores[30]	8	Verilog	{2, 6, 35}
Chipyard[31]	8	Chisel	{12, 19, 52}

TABLE I: Dataset composition and design size information.

Design label preparation. To obtain netlist labels including design area, register slack (SL), worst negative slack (WNS), and total negative slack (TNS), we employed Synopsys Design Compiler® 2021 with the NanGate 45nm technology library. To align with real-world scenarios, multiple parameters within the Design Compiler were adjusted, and a set of the PPA values along the Pareto frontier were utilized as ground truth labels.

Training-testing data splitting. We randomly selected 7 designs from the dataset as the test set, while the remaining 15 designs were used as the training set in the downstream PPA prediction task. To prevent data leakage, all the graph generative models mentioned in this paper were only trained on these 15 training designs.

Graph generative model baselines. We selected several representative and most closely related baselines, including GraphRNN [17], DVAE [15], SparseDigress [32] and GraphMaker [25]. These 4 baselines all need to be adapted to the circuit generation task. GraphRNN [17] and DVAE [15] are node ordering-based autoregressive

Model	1-Wasserstein distance $W_1 \downarrow$						$\mathbb{E}_{\hat{G}\sim p_\theta}\left[\frac{M(\hat{G})}{M(G)}\right] \to 1$					
	Out Degree		Cluster		Orbit		# Triangle		$\hat{h}(A,Y)$		$\hat{h}(A^2,Y)$	
	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core
GraphRNN [17]	1.03	0.762	0.935	0.136	1.94	0.978	0.016	0.0238	5.33	3.27	2.13	2.53
DVAE [15]	1.31	0.832	0.912	0.146	1.33	1.24	0.241	0.105	6.66	5.19	3.17	2.96
GraphMaker-v [25]	0.678	0.621	0.957	0.112	1.24	1.05	**0.262**	0.501	2.31	1.96	1.53	1.62
Sparse Digress-v [32]	0.598	0.652	0.972	0.135	1.21	0.953	0.163	0.312	4.89	4.32	2.56	2.13
SynCircuit w/o diff	0.373	0.323	0.925	0.0807	1.09	0.926	0.0501	**0.760**	0.629	0.361	0.561	0.321
SynCircuit w/ diff	**0.236**	**0.226**	**0.876**	**0.0452**	**0.344**	**0.231**	0.146	1.34	**0.713**	**0.670**	**0.624**	**0.487**

TABLE II: Evaluation for structural properties similarity with original realistic circuits, best results are in **bold**. Lower W_1 and $\left|\mathbb{E}_{\hat{G}\sim p_\theta}\left[M(\hat{G})/M(G)\right]-1\right|$ reflect a better graph similarity with the realistic ones. All 4 baselines were adapted to generate directed circuit graphs without violating the constraints.

generative approaches. These methods cannot be directly applied to generate directed cyclic graphs. We have to break the cycles in the training circuits and use the topological order of nodes as the sequence for the model training and inference. During generation, edge directions are automatically determined by the topological order. In the sequential generation process, we introduced a validity checker for circuits to ensure that the resulting digital design is valid. **But since these two models can only be applied on DAGs, the generated graph contains no cycles which is very different from the real designs.**

For GraphMaker [25] and SparseDigress [32], these are one-shot approaches that ignore edge direction information, generating only undirected graphs. We applied the Gravity-Inspired Graph Autoencoders [33] method to process the generated undirected graph by assigning the direction for each edge. To ensure the validity of the generated graphs by GraphMaker [25] and SparseDigress [32], we must refine the parent edges in a specific node order. The synthesis tool removed most part of the generated circuits, so these two baseline models can only be used for comparing graph characteristics and cannot support valid designs for downstream tasks.

SynCircuit setup. For our proposed graph generative model, we set the diffusion steps to 9 and utilized a 5-layer MPNN. Both the node attribute embeddings and hidden embeddings were set to 256. In the circuit optimization phase, to accelerate the evaluation process, we replaced the slow synthesis tool with a trained discriminator to approximate the PCS. In the MCTS process, we set the number of simulations to 500 and the maximum exploration depth to 10 for each register cone. All these experiments were conducted on a platform with an Intel(R) Xeon(R) Gold 6438Y+ processor and 8*4090 GPUs.

B. Experimental Results

1) Graph structural properties evaluation: To evaluate the similarity of the generated graph structures with realistic ones, we follow GraphRNN [17] and GraphMaker [25] and report distance metrics for node out degree, clustering coefficient, and four-node orbit count distributions. We use the 1-Wasserstein distance to model statistic distributions from the original and generated graphs, respectively. A lower W_1 value indicates better similarity.

Beyond distribution distance metrics, we also directly compare several scalar-valued statistics [25], [34]. Let $M(G)$ be a non-negative statistic; we report $\mathbb{E}_{\hat{G}\sim p_\theta}\left[M(\hat{G})/M(G)\right]$, where $\hat{G}$ is a generated graph. A value closer to 1 indicates better performance. For the M statistics, we choose triangle count, $\hat{h}(A,X)$ [25], [34] and two-hop correlations $\hat{h}(A^2,X)$. $\hat{h}(A,X)$ reflects the correlation between graph structure and node types.

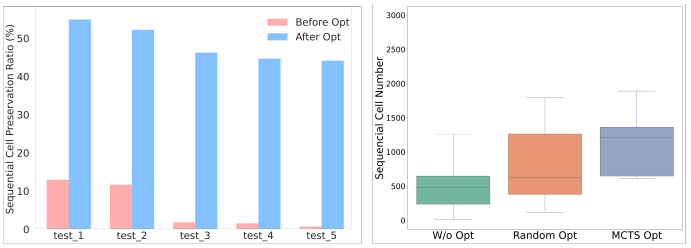
The results are shown in Table II. We can see that our SynCircuit has the best performance among 5 out of the 6 metrics. Benefiting from the diffusion model and the out-degree guidance in the postprocessing phase, the degree distribution of our synthetic circuits is more similar to the real designs. This feature is important because the digital circuits are indeed scale-free networks[4] [35].

[4]A Scale-Free Network is a type of network characterized by a degree distribution that follows a power law. This means that in such networks, the probability $P(k)$ that a node has k connections (or degree k) decreases polynomially as k increases

SynCircuit w/o diff is an ablation study where we remove the diffusion model and randomly construct edges when generating G^{ini} and $P_E^{(0)}$, but applying post-processing to ensure certain predefined constraints. We observe that its performance is greatly inferior to the complete SynCircuit, demonstrating the effectiveness of our designed generative diffusion model.

2) Logic redundancy and timing evaluation: We employ Monte Carlo Tree Search (MCTS) to optimize the register cones within G^{val}. For comparison, we implement an ablation study of randomly altering edge connections on G^{val} while still ensuring every step is valid. We utilize the same number of simulations as MCTS and adopt the optimal solution identified throughout the process as the results. The SCPR enhancement is illustrated in Figure 4(a), demonstrating that MCTS can significantly reduce logic redundancy.

Figure 4(b) displays the number of sequential cells saved during the logic synthesis using different optimization techniques. We can see that by introducing our MCTS-based refinement, the registers saved in the netlist have been greatly improved over no optimization and the optimization performance is also better than that using a random optimization method.



(a) SCPR before and after MCTS-based optimization

(b) No. of Sequential Cell Saved in the Netlist Stage

Fig. 4: (a) Logic redundancy metric SCPR is calculated by dividing the number of sequential cells in the synthesized netlist by the total number of bits in sequential signals in the pre-synthesis HDL design. Here we select the five G^{val} examples with the most significant logic redundancy. The register ratio (SCPR) cannot even reach 20% if we do not optimize G^{val}. After the MCTS optimization, the SCPR is greatly improved and exceeds 50% in some synthetic circuits. (b) The number distribution of registers preserved after logic synthesis. The logic redundancy is greatly reduced by our proposed MCTS optimization method.

Netlist timing statistics comparison We use two metrics to evaluate the design's timing characteristics: Worst Negative Slack (WNS) and Total Negative Slack divided by the number of violated paths (TNS/NVP).

- WNS represents the most severe timing violation in the design, indicating the largest negative slack among all timing paths, which points to the longest delay and the most critical bottleneck in signal propagation.

Model	Register Slack			WNS			TNS			Area		
	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓
Basic training data (No-pseudo circuits)	0.70	27%	0.83	0.86	20%	0.83	0.81	50%	0.97	0.89	30%	0.62
GraphRNN [17]	0.70	27%	0.83	0.88	21%	0.83	0.80	54%	0.97	0.84	44%	0.75
DVAE [15]	0.69	29%	0.94	0.88	24%	0.86	0.78	50%	0.97	0.84	61%	0.86
SynCircuit w/o opt	0.72	24%	0.79	**0.90**	23%	0.85	0.80	48%	0.86	0.86	39%	0.72
SynCircuit w/ opt	**0.77**	**16%**	**0.70**	0.89	**20%**	**0.80**	**0.97**	**45%**	**0.64**	**0.95**	**25%**	**0.34**

(a) Basic training dataset contains 15 real designs. Lower $|R-1|$, MAPE and RRSE reflect better model prediction performance.

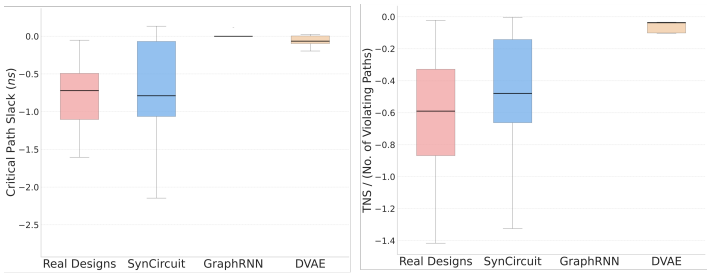
Model	Register Slack			WNS			TNS			Area		
	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓
Basic training data (No-pseudo circuits)	0.52	34%	1.1	NA	52%	2.1	NA	67%	1.1	0.65	66%	1.3
GraphRNN[17]	0.52	34%	1.1	0.71	42%	1.7	-0.30	74%	1.1	0.51	77%	1.6
DVAE[15]	0.49	36%	1.3	0.75	77%	2.6	0.76	93%	1.1	0.70	86%	2.4
SynCircuit w/o opt	0.56	35%	1.2	0.65	47%	1.9	0.72	70%	0.85	0.63	62%	1.1
SynCircuit w/ opt	**0.67**	**25%**	**0.76**	**0.87**	**34%**	**1.2**	**0.98**	**61%**	**0.63**	**0.95**	**36%**	**0.61**

(b) Basic training dataset contains 5 real designs. Lower $|R-1|$, MAPE and RRSE reflect better model prediction performance.

TABLE III: Model Performance on the Register Slack, WNS, TNS, and area prediction. The basic training dataset in (a) and (b) contains 15 and 5 real designs respectively. In (a) and (b), the augmentation datasets are added to the basic training set, each always with 25 pseudo-circuits, generated from SynCircuit, GraphRNN [17], and DVAE [15]. R with NA means that the model prediction is the same for all testing designs

- The metric TNS/NVP provides the average negative slack per violated path, reflecting the overall severity of timing violations and offering insights into the distribution of delays.

From Figure 5, We observe that the graphs generated by GraphRNN [17] and DVAE [15] exhibit very small WNS (critical path slack) and TNS/NVP values, failing to capture the inherent delay characteristics of circuits. In contrast, our SynCircuit demonstrates a more similar distribution of the two metrics to real designs. This suggests that SynCircuit is more effective in modeling the diverse timing behaviors present in real circuits.



(a) Critical Path Slack (ns) (b) TNS / No. of Violating Paths (ns)

Fig. 5: Netlist statistics for the three synthetic datasets and real benchmarks. The distributions of two statistic metrics: Critical Path Slack (WNS) and the ratio of Total Negative Slack to the Violating Path Numberare shown in (a) and (b), respectively. The dataset generated by SynCircuit exhibits *more similar statistics distributions* to those of the real designs compared to GraphRNN [17] and DVAE[15].

3) SynCircuit application in RTL tasks: In this part, we explore the application of synthetic circuit generation for ML-based RTL-level PPA prediction model, primarily referencing the overall design evaluation methods (i.e., area, WNS, and TNS) proposed by MasterRTL [5] and the fine-grained timing slack evaluation by RTL-Timer [28]. We employ three metrics to assess model performance: correlation coefficient (R), Mean Absolute Percentage Error (MAPE), and Root Relative Squared Error (RRSE). Lower MAPE and RRSE values indicate better model performance.

We randomly selected 5 and the full 15 training real designs to create two basic training datasets. For each basic training dataset, we augmented it with different synthetic datasets (three sets of 25 designs each), generated respectively by SynCircuit, GraphRNN [17], and DVAE [15], to study how these synthetic designs affect model performance.

The results are shown in Table III. We can see that in both training settings, models trained on datasets augmented with Synthetic-generated data consistently outperformed those trained solely on real designs, achieving the best results across all metrics. And we have more model performance gain when the basic training dataset contains 5 designs. The Register Slack MAPE is reduced by 10% in both basic training settings. And we have a 30% area MAPE reduction in the 5 basic training dataset settings.

Notably, models augmented with DVAE [15]-generated and GraphRNN [17]-generated data performed worse, regardless of the training dataset size. This may indicate a significant gap between the data generated by these baselines and the real designs due to logic redundancy. As observed in Figure 5, the synthetic data generated from GraphRNN [17] and DVAE [15] contain very few paths with large delays. This discrepancy may have caused the model's learning to deviate from the normal timing features for Register Slack, WNS, and TNS tasks.

We also included non-optimized circuits SynCircuit w/o opt (G^{val}) as part of an ablation study. It can be observed that the excessive design logic redundancy introduced may have adversely affected the model's predictive performance, rendering it inferior to scenarios involving only real training circuits. This observation indirectly demonstrates the importance of optimizing logic redundancy in generating synthetic circuits that more closely resemble real-world designs.

VIII. Conclusion And Future Work

Data-driven automation techniques have become increasingly prevalent in digital circuit design in recent years. Nevertheless, the availability of open-source circuits is often limited. To address this issue, we propose the SynCircuit, an automatic digital circuits generation framework. It consists of three stages: directed cyclic graph generation, probability-guided postprocessing, and MCTS-based optimization. Our experiments demonstrate that the designs generated by SynCircuit not only exhibit structural properties that closely resemble those of real designs, but also enhance model performance in ML-based power, performance, and area (PPA) prediction tasks at the RTL early stage.

IX. Acknowledgement

This work is supported by the Hong Kong Research Grants Council (RGC) ECS Grant 26208723, CRF Grant C6003-24Y, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and the National Supercomputing Center in Guangzhou Nansha Subcenter for computational resources.

REFERENCES

[1] Z. Pei, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[2] C.-C. Chang, Y. Shan, S. Fan, J. Li, S. Zhang, N. Cao, Y. Chen, and X. Zhang, "Lamagic: Language-model-based topology generation for analog integrated circuits," *arXiv preprint arXiv:2407.18269*, 2024.

[3] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," *arXiv preprint arXiv:2312.08617*, 2023.

[4] Y. Bai, A. Sohrabizadeh, Z. Qin, Z. Hu, Y. Sun, and J. Cong, "Towards a comprehensive benchmark for high-level synthesis targeted to fpgas," *Advances in Neural Information Processing Systems*, vol. 36, pp. 45 288–45 299, 2023.

[5] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[6] C. Xu, C. Kjellqvist, and L. W. Wills, "SNS's not a synthesizer: a deep-learning-based synthesis predictor," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, pp. 847–859.

[7] Z. Qin, Y. Bai, A. Sohrabizadeh, Z. Ding, Z. Hu, Y. Sun, and J. Cong, "Cross-modality program representation learning for electronic design automation with high-level synthesis," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024, pp. 1–12.

[8] S. Yang, Z. Yang, D. Li, Y. Zhang, Z. Zhang, G. Song, and J. Hao, "Versatile multi-stage graph neural network for circuit representation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 313–20 324, 2022.

[9] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[10] J. Zou, X. Wang, J. Guo, W. Liu, Q. Zhang, and C. Huang, "Circuit as set of points," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[11] C. Bai, J. Huang, X. Wei, Y. Ma, S. Li, H. Zheng, B. Yu, and Y. Xie, "Archexplorer: Microarchitecture exploration via bottleneck analysis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 268–282.

[12] M. Yoon, Y. Wu, J. Palowitch, B. Perozzi, and R. Salakhutdinov, "Graph generative model for benchmarking graph neural networks," *arXiv preprint arXiv:2207.04396*, 2022.

[13] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, "A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda," *arXiv preprint arXiv:2504.03711*, 2025.

[14] S. Jiang, D. Jiang, M. Balandat, B. Karrer, J. Gardner, and R. Garnett, "Efficient nonmyopic bayesian optimization via one-shot multi-step trees," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 039–18 049, 2020.

[15] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs," *Advances in neural information processing systems*, vol. 32, 2019.

[16] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "Brp-nas: Prediction-based nas using gcns," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 480–10 490, 2020.

[17] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *International conference on machine learning*. PMLR, 2018, pp. 5708–5717.

[18] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer, 2018, pp. 412–422.

[19] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," *Advances in neural information processing systems*, vol. 32, 2019.

[20] X. Chen, J. He, X. Han, and L.-P. Liu, "Efficient and degree-guided graph generation via discrete diffusion modeling," *arXiv preprint arXiv:2305.04111*, 2023.

[21] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer, "Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators," in *International Conference on Machine Learning*. PMLR, 2022, pp. 15 159–15 179.

[22] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard, "Digress: Discrete denoising diffusion for graph generation," *arXiv preprint arXiv:2209.14734*, 2022.

[23] M. Li, V. Shitole, E. Chien, C. Man, Z. Wang, Y. Zhang, T. Krishna, P. Li *et al.*, "Layerdag: A layerwise autoregressive diffusion model of directed acyclic graphs for system," in *Machine Learning for Computer Architecture and Systems*, 2024.

[24] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International conference on machine learning*. PMLR, 2021, pp. 8162–8171.

[25] M. Li, E. Kreačić, V. K. Potluru, and P. Li, "Graphmaker: Can diffusion models generate large attributed graphs?" *arXiv preprint arXiv:2310.13833*, 2023.

[26] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Advances in neural information processing systems*, vol. 26, 2013.

[27] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[28] W. Fang, S. Liu, H. Zhang, and Z. Xie, "Annotating slack directly on your verilog: Fine-grained rtl timing evaluation for early optimization," in *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024, pp. 1–6.

[29] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers (ITC)*, 2000.

[30] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[31] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, 2020.

[32] Y. Qin, C. Vignac, and P. Frossard, "Sparse training of discrete diffusion models for graph generation," *arXiv preprint arXiv:2311.02142*, 2023.

[33] G. Salha, S. Limnios, R. Hennequin, V.-A. Tran, and M. Vazirgiannis, "Gravity-inspired graph autoencoders for directed link prediction," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 589–598.

[34] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 887–20 902, 2021.

[35] X. F. Wang and G. Chen, "Complex networks: small-world, scale-free and beyond," *IEEE circuits and systems magazine*, vol. 3, no. 1, pp. 6–20, 2003.

MAGE: A Multi-Agent Engine for Automated RTL Code Generation

Yujie Zhao*, Hejia Zhang*, Hanxian Huang, Zhongming Yu, Jishen Zhao
University of California San Diego
La Jolla, CA 92093, USA
{yuz285, hez024, hah008, zhy025, jzhao}@ucsd.edu

***Abstract*—The automatic generation of RTL code (e.g., Verilog) through natural language instructions has emerged as a promising direction with the advancement of large language models (LLMs). However, producing RTL code that is both syntactically and functionally correct remains a significant challenge. Existing single-LLM-agent approaches face substantial limitations because they must navigate between various programming languages and handle intricate generation, verification, and modification tasks. To address these challenges, this paper introduces MAGE, the first open-source[1] multi-agent AI system designed for robust and accurate Verilog RTL code generation. We propose a novel high-temperature RTL candidate sampling and debugging system that effectively explores the space of code candidates and significantly improves the quality of the candidates. Furthermore, we design a novel Verilog-state checkpoint checking mechanism that enables early detection of functional errors and delivers precise feedback for targeted fixes, significantly enhancing the functional correctness of the generated RTL code. MAGE achieves a 95.7% rate of syntactic and functional correctness code generation on VerilogEval-Human v2 benchmark, surpassing the state-of-the-art Claude-3.5-sonnet by 23.3%, demonstrating a robust and reliable approach for AI-driven RTL design workflows.**

I. INTRODUCTION

Digital hardware design usually requires engineers to define the architecture and functionality of hardware by writing code in hardware description languages (HDLs), such as Verilog and VHDL. As VLSI designs become more complex, hardware design workflows are increasingly time-consuming and error-prone [1]–[3]. Although electronic design automation (EDA) tools have advanced to support this workflow [4]–[7], the demand for more efficient and automated hardware design solutions remains crucial.

Large Language Models (LLMs) [8], [9] have recently shown promising potential in assisting and improving hardware design [10]–[13]. Recent studies explored leveraging LLMs to improve the correctness of RTL code generation through model fine-tuning [10], [14], [15], training of domain-adaptive models [16], and single-agent methods [13], [17], [18] that augment LLMs by incorporating results and feedback from compilers or simulators to refine the generated code.

However, previous studies using a single LLM or a single agent face substantial limitations. First, these approaches fall short of the context-switch between tasks, programming languages, and knowledge domains. As a result, the quality of RTL design suffers: for instance, a single Claude-3.5-sonnet agent [9] has a functionality pass rate of only 75.0% even for simple design tasks [19]–[21]. Second, HDLs are specifically designed to describe the logic and architecture of digital hardware at a low level, focusing on the flow of data between registers, the timing of operations, and hardware characteristics such as propagation delays and signal dependencies. Directly adopting general-purpose LLM or multi-agent designs will produce low-accuracy RTL code that fails to meet timing constraints, resulting in wrong designs [10], [16], [19], [20].

To address these challenges, we propose **MAGE**, the first open-source **M**ulti-**AG**ent **E**ngine to achieve automatic high-quality RTL code generation. Different from the existing single-agent RTL code generation frameworks [13], [17], [18], MAGE enables specialized agents to handle distinct aspects in the RTL development pipeline. Inspired by real-world RTL design workflows, where specialists focus on distinct stages, our approach, MAGE, employs a specialized multi-agent architecture composed of four types of key agents: the RTL code generation agent, testbench generation agent, judge agent, and debug agent. Each type plays a specific role, working collaboratively within our tailored recursive framework to generate optimized and reliable RTL code.

MAGE consists of three key **design principles**. First, mimicking the iterative nature of human RTL design teams in addressing complex design challenges, we developed a high-efficiency collaboration workflow with a delicate design context communication protocol. Second, we propose a novel high-temperature RTL candidate sampling and debugging system, which leverages both high-temperature sampling and simulation-based scoring to identify promising candidates. Our key insight is that higher temperatures result in more diverse LLM outputs, while also increasing the likelihood of including the highest-quality candidates for optimization in the next stage. Finally, we propose a novel Verilog-state checkpointing and validation scheme, which substantially improves the functional correctness. Unlike conventional methods that provide feedback only on the final output mismatch [13], [17], [18], MAGE validates an RTL design by comparing state values with expected outputs at each clock edge. This enables early detection of functional errors and provides precise feedback

* Equal Contribution

[1]MAGE is open-sourced at https://github.com/stable-lab/MAGE-A-Multi-Agent-Engine-for-Automated-RTL-Code-Generation.

for targeted fixes.

In summary, we make the following contributions:

- We design MAGE, the first open-source LLM-based multi-agent system for robust and accurate Verilog RTL code generation. By decomposing a complex hardware design into manageable sub-tasks, we design LLM agents to handle sub-tasks and design the system to enable effective agent communication and collaboration.
- We propose a novel high-temperature RTL candidate sampling system, notably enhancing generated code quality by exploring the benefits of high-randomness generation.
- We propose a novel Verilog-state checkpoint checking mechanism, which provides precise feedback for targeted fixes, significantly improving the RTL code quality.
- MAGE achieves a **95.7%** rate of syntactic and functional correctness code generation on the VerilogEval- Human v2 benchmark, surpassing all existing methods, representing a critical step toward automating and optimizing hardware design workflows, offering a more robust methodology for AI-driven RTL design.

II. Background and Motivation

A. Characteristics and challenges of LLM-based RTL design

Decomposing Complex Traditional RTL Design. Traditional digital hardware design flows necessitate hardware engineers to iteratively perform: (1) implement Verilog code to specify hardware architectures and behaviors, (2) customize test benches to rigorously verify the correctness of these hardware descriptions (3) iterate between Verilog simulations, signal waveform reasoning, and code refining until all output signals match expected behavior. This iterative loop among design, verification, and refinement makes the RTL design process not only challenging but also highly demanding in terms of both time and expertise. The complexity of traditional Verilog RTL design calls for decomposing the whole process into multiple manageable sub-stages and adopting different specialists (agents) for different sub-stages.

Related Work on LLM-based RTL Design. Recent works [10], [15], [16], [22], [23] train or fine-tune general code generation LLMs by incorporating RTL domain knowledge to enhance the correctness of RTL code generation. Single-agent methods [13], [17], [18] further integrate simulation results and introduce more stages in the code generation process, such as planning, verification, and code refining based on the feedback from simulation. However, this process involves generating both synthesizable and non-synthesizable code for RTL generation and verification, and context switches among different sub-tasks in multiple iterations, necessitating different domain knowledge and problem-solving abilities. Thus, adopting a single agent for all these tasks in the hardware design process leads to sub-optimal results, as they have to process such complex contexts in each interaction and maintain consistency through a long conversation history.

In contrast, multi-agent systems distribute tasks among agents with independent conversation histories, enabling specialized task handling and greater modularity. However, Aivril [24] only implements a basic two-agent division between code generation and review, still requiring a single agent to handle both synthesizable RTL and non-synthesizable verification code, thus failing to address the fundamental context-switching challenges. Verilogcoder [25] further constrains accessibility and system adaptability through its closed-source implementation and reliance on proprietary components (e.g., its Abstract Syntax Tree and Waveform Tracing Tool).

Temperature Sampling for LLM-based RTL Design. Given an input requirement, LLMs rely on a specific decoding strategy to generate the code auto-regressively. Temperature sampling method [26] uses a temperature coefficient T (usually $\in [0, 1]$) to control the sampling randomness. Increasing the temperature avoids overly conservative results and promotes diversity in code generation, thus enhancing the chance of exploring the correct answers. However, this comes at the cost of introducing more noise and errors in the generation results. Recent studies [27], [28] show that high-temperature sampling can improve correctness in software code generation through multiple sampling iterations. However, higher temperatures tend to result in poorer performance for RTL design as explored in recent studies [10], [21]. We found a key reason for this is that single-agent mechanisms restrict the ability to design independent and efficient sampling and feedback, thereby hindering the optimization potential of high-temperature sampling for RTL code.

B. Opportunity on Effective Code Generation

To tackle the aforementioned limitations of both existing single-agent and multi-agent systems for RTL code generation, we identify potential opportunities to develop effective RTL code generation frameworks to improve correctness.

First, to further reduce the mutual influence between non-synthesizable testbenches and synthesizable RTL code generation, as well as to decrease the complexity of tasks assigned to each agent, we recognize the necessity of appropriately distributing tasks among multiple agents and organizing the workflow effectively. Specifically, previous works (e.g., [17], [24]) employed a single agent to generate both non-synthesizable testbenches and synthesizable RTL code simultaneously. However, [29] found that generating testbenches and code together within the same conversation may lead to a corresponding decrease in effectiveness. Moreover, test cases generated in this manner can become biased and influenced by the code, resulting in a loss of objectivity and diversity.

Second, based on our analysis in Sec. II-A, we thoroughly reviewed existing multi-agent systems [24], [25], where we found that all of them are closed-source and dependent on third-party proprietary tools that are not LLM-directly-adapted. This inevitably limits system extensibility and transparency. Therefore, we identify the need for an open-sourced multi-agent system.

Finally, to address the limitation of high-temperature sampling for RTL code generation described in Sec. II-A, we summarize the key to efficiently sampling RTL code candidates process to mitigate the impact of randomness while leveraging

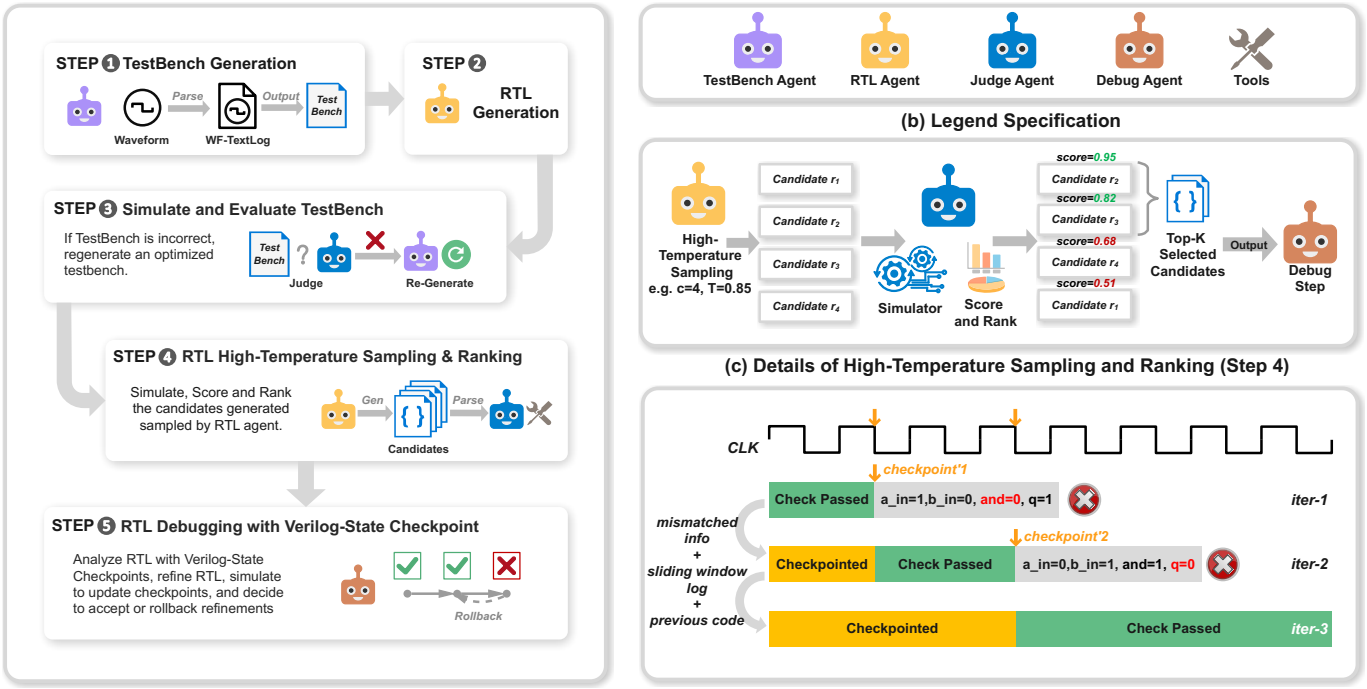


Fig. 1: (a) The overview of MAGE, (b) the roles of four types of agents, (c) state check module, and (d) sampling and debugging module.

the exploration benefits. Given that RTL code is synthesizable and a practical miscount scoring method is applicable, we prioritize candidate selection at an early stage by identifying top-scoring candidates. This approach allows us to reduce the exploration cost by filtering suboptimal results early.

C. Opportunity on Effective Code Debugging

Most existing LLM approaches for RTL code generation [13], [17], [18] rely on the direct application of original golden testbenches, which can only provide pass rates. This provides very limited feedback to LLMs in fine-tuning RTL code as pass rates alone lack detailed insights into crucial aspects such as timing analysis, signal interactions, variable values, and mismatch information, all of which are essential for ensuring high-quality RTL design. VerilogCoder [25] applies closed-source Abstract Syntax Tree (AST) analysis, which significantly restricts flexibility and transparency. Furthermore, the introduction of tools that output in graphical form, which LLM can not directly apply, increases the complexity of tasks for LLMs and reduces their effectiveness. Therefore, we identify the requirement for an optimized testbench, which will output a log resembling a simulated waveform in text form, which can be directly adaptable by LLMs. This shifts the reliance on closed-source tools to a fully open-source LLM-based textual protocol, alongside significant improvements in LLM RTL code debug effectiveness, improving both the extension of the framework and the quality of the analysis.

III. MAGE Design

We present MAGE, a multi-agent engine designed specifically for RTL. This system integrates an RTL-specific context communication protocol with high-efficiency, LLM-adapted tools for fine-tuning RTL, all within a productive process.

A. Multi-Agent System Overview

Figure 1 depicts a comprehensive overview of the workflow implemented in MAGE. Inspired by the collaborative division of work approaches of human RTL design teams, our framework incorporates four types of agents, each playing a specific and concrete role in the automation process. Figure 1 (b) specifically delineates the responsibilities and tasks assigned to each agent, ensuring a clear understanding of the workflow. (1) A Testbench Generation Agent is responsible for creating optimized test benches in a textual-waveform-output format based on the natural language specifications and any available golden test benches. (2) A RTL Generation Agents convert these specifications along with the optimized test bench into Verilog code, incorporating syntax checking to ensure code validity. (3) A Judge Agent then takes over by simulating and evaluating the generated RTL code against the optimized test bench. It scores the RTL code candidates and decides whether any require debugging or if the test bench needs to be regenerated. (4) A Debug Agent performs iterative refinements on any code that fails initial tests, using textual waveform-like simulation outputs as feedback for improvements. Our method mimics the division of labor in human teams and enhances the efficiency and accuracy of automated RTL design. It should be noted that, whether in RTL code generation or debugging, the agent will perform at most $s = 5$ iterations to automatically fix syntax errors.

Based on the distinct and specific roles of multiple agents, we have designed an efficient workflow to ensure that each agent's unique contributions are seamlessly integrated, fa-

cilitating a coherent progression throughout the workflow. Figure 1 (a) demonstrates the entire process is systematically divided into five main steps as depicted as follows:

Step ❶ – Generate initial textual-waveform-output testbenches. To reduce the limitations of pass-rate-output golden testbenches, which is analyzed in Sec. II-C, we directly utilize natural language specifications to generate optimized testbenches that can output State Checkpoints, which will be used in Step ❺ (see Figure 1 (d) and Sec. III-C). Furthermore, since natural language specifications may contain ambiguities, we combine the input with the golden testbench if available.

Step ❷ – Based on the natural language specifications and the optimized testbench, we generate an initiate RTL code.

Step ❸ – If the initial RTL code cannot pass the optimized testbench, the judge agent evaluates the testbench and regenerates it if deemed incorrect.

Step ❹ – If the RTL code is deemed correct, we will employ an **High-Temperature RTL Sampling and Scoring Process** (see Figure 1 (c) and detailed in Sec. III-B) to generate and rank RTL code Candidates.

Step ❺ – If the RTL code candidates still have function errors, we will employ a **RTL Debugging with State Checkpoint Mechanism** (see Figure 1 (d) and Sec. III-C) to make effective debug trials and debug the selected RTL code candidates.

B. High-Temperature RTL Sampling and Scoring

As depicted in Sec. II-B, high-temperature sampling has a better chance of exploring the correct code. However, in prior studies on Temperature Sampling for RTL code generation like [27], [28], it is commonly acknowledged that high-temperature sampling introduces significant randomness, often resulting in reduced correctness. To address this issue, we first explore the potential opportunity of using multiple RTL code generation candidates in high-temperature sampling to reduce simulated mismatch counts in one iteration. Fig. 2 shows the distribution of normalized mismatch counts for the best candidates at two different temperatures. We observe that the best candidate with high temperature has a lower mismatch count for most problems. This suggests that, despite increased randomness, high-temperature sampling yields better performance when the sampling is sufficiently extensive. The improvement in pass rates at each stage, as shown in Fig. 2, indicates the potential benefits of performing an effective sampling policy at high temperatures.

Based on the above observation, we propose a High-Temperature Sampling and Ranking Process as follows:

1. We sample c RTL code candidates for issue i:

$$\mathcal{R}_i = \{r_i^1, r_i^2, \ldots, r_i^c\}, r_i \sim P_T\left(r \mid p_{sys}, SP_i, TB_i\right) \quad (1)$$

Here, T is the temperature, p_{sys} denotes the system prompt of the RTL code generation, SP_i is the natural language specification of the issue i, and TB_i is the corresponding testbench. The term $P_T\left(r \mid p_{sys}, SP_i, TB_i\right)$ represents the probability distribution of the generated RTL code given the temperature, system prompt, specification, and testbench.

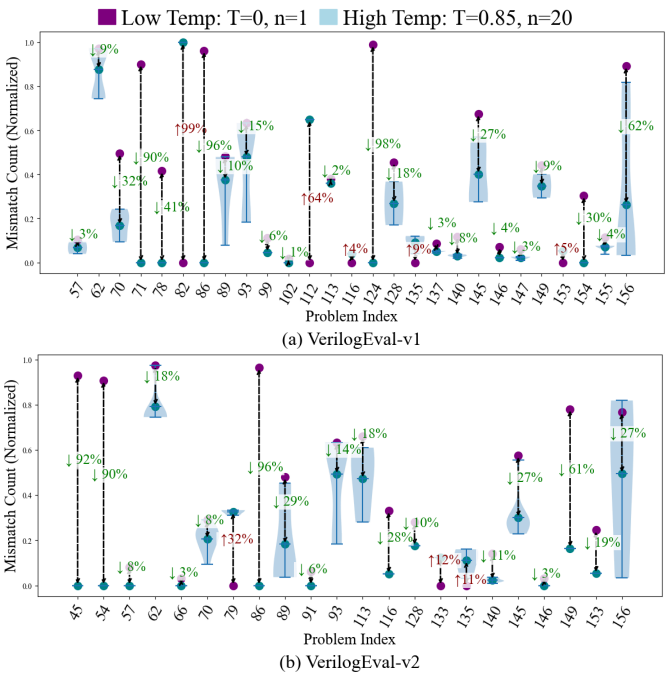


Fig. 2: Normalized mismatch count of generated testbenches at different stages under varying temperature configurations, using the Claude 3.5 Sonnet model (dated 2024-10-22) across two benchmarks: VerilogEval-v1-Human [20] and VerilogEval-v2 [21]. Problems that passed before Step ❹ and those with zero mean mismatches in both configurations are not shown in the figure. The blue violin plot shows that candidates (blue dots) generated with high-temperature sampling typically have lower mean mismatch counts across most problems compared to those generated with low temperatures (purple dots).

2. We select the Top-K candidates from $\mathcal{R}_i$ based on their scores. Specifically, each candidate in $\mathcal{R}_i$ is evaluated to obtain a normalized mismatch-based score as follows:

$$s(r) = 1 - \frac{m(r)}{tc(r)} \quad (2)$$

where $m(r)$ is the mismatch count and $tc(r)$ is the total number of checks of the generated RTL code r. We then select the K candidates with the highest scores, denoted as follows:

$$\mathcal{R}^{\star}_{i,0} = \underset{A \subseteq \mathcal{R}_i, |A|=K}{\arg\max} \sum_{r \in A} s(r) \quad (3)$$

3. We employ a debug agent to perform debugging trials, generating a new candidate $r^{\star}_{trial} = D(r^{\star})$ for each selected candidate $r^{\star}$, where $D(r)$ denotes the debug trial for RTL code r. The selected candidate set at iteration m is then updated as:

$$R^{\star}_{i,m} = \{ \underset{r \in \{D(r^{\star}), r^{\star}\}}{\arg\max} \; s(r) \mid r^{\star} \in \mathcal{R}^{\star}_{i,m-1} \} \quad (4)$$

And this process repeats until $\underset{r \in R^{\star}_{i,m}}{max}\; s(r) = 1$ or the iteration limit is reached.

C. RTL Debugging with State Checkpoint Mechanism

Our objective is to design a mechanism that is entirely LLM-based, independent of third-party closed-source tools,

and effective in improving RTL debugging efficiency. To achieve this, we propose the RTL Debugging with State Checkpoint Mechanism, which follows these steps:
1. Utilizing the optimized testbench generated in Step ❶ and ❷, we extract the input vector **I**, the DUT, and the output **O**, and subsequently identify the earliest mismatch point as:

$$t_m = \min\{t \mid \mathbf{O}_{DUT}(t) \neq \mathbf{O}_{exp}(t), t \geq 0\} \tag{5}$$

where $\mathbf{O}_{\text{DUT}}(t)$ is the output vector from the DUT at time t, and $\mathbf{O}_{\text{exp}}(t)$ is the expected output vector at time t.
2. We generate the State Checkpoint and collect the textual waveform window as:

$$W = \Big\{ \big(\mathbf{I}(t'), \mathbf{O}_{DUT}(t'), \mathbf{O}_{exp}(t')\big) \mid t' \in \big[\max(t_m - L_W, 0), t_m\big] \Big\} \tag{6}$$

where W represents the waveform window, L_W is the window length, and $\mathbf{I}(t')$ is the input vector at clock edge t'.
3. The debugging agent then takes the textual waveform window W and the original testbench as inputs, generates a new trial of debugged RTL code, and performs replacement actions to correct the identified faults in the RTL. The simulation is subsequently rerun to verify if the mismatch is resolved in the updated State Checkpoint.

IV. Experiments

A. Experimental Setup

Benchmarks. Two widely used benchmarking datasets: VerilogEval-v1-Human [20] and VerilogEval-v2 [25]. **Model.** Claude 3.5 Sonnet 2024-10-22 [9]. **Baselines.** i. Vanilla models: Generating RTL code in a single pass using language models, including both general-purpose LLMs (e.g. GPT-4o [30], Claude 3.5 Sonnet [9]) and RTL-specified models (e.g. ITERTL [22], CodeV [23]). ii. LLM Agent systems: Agent-based systems designed to enhance LLM performance for RTL code generation, including open-source solutions (OriGen [13]) and closed-source ones (e.g. VeriAssist [17], AutoVCoder [18], VerilogCoder [25], AIVRIL [24]). **Configurations.** Our implementation of MAGE integrates the open-source Verilog compiler and simulator Icarus Verilog [31] with an LLM-agnostic API interface offered by the open-source framework LlamaIndex [32]. Based on the superior performance [24], [33], [34] of Claude 3.5 Sonnet [9] in multiple coding tasks, we choose to run experiments with that language model. In accordance with the VerilogEval V1 [20] and V2 [21] benchmark, we conducted experiments to measure Pass@1 under 2 settings: *Low Temperature* (T=0, top_p=0.01, n=1) and *High Temperature* (T=0.85, top_p=0.95, n=20). Here, temperature T is described in Sec. II-A. Top_p limits the output pool to a cumulative probability threshold. n represents the number of evaluation runs. **Metrics.** Following prior works [17], [18], [21], the Pass@1 metric is computed as:

$$pass@k = \mathbb{E}_{\text{Problems}}\left[1 - \frac{\binom{n-c_p}{k}}{\binom{n}{k}}\right] \tag{7}$$

where $k = 1$, and c_p is the number of passing runs. This Pass@1 metric, which accounts for multiple runs, reflects the expected percentage of problems that the system solves correctly when executed once for each problem.

TABLE I: Pass rates of different temperature configurations in MAGE.

Config	VerilogEval-Human Pass@1	VerilogEval-V2 Pass@1
High Temp	**94.8**	**95.7**
Low Temp	89.1	93.6

As shown in Table I, the High Temperature setting achieves higher Pass@1, so it is adopted for subsequent experiments.

B. Key Results

Table II shows the comparison of MAGE and baselines. For a fair comparison, we select the highest pass rate among their experiment configuration. MAGE achieves the best performance on both benchmarks, delivering consistent improvements over specialized (e.g., VerilogCoder) and general-purpose systems(e.g., GPT-4, CodeQwen). Specifically, MAGE obtains a Pass@1 score of 94.8% on VerilogEval-Human and 95.7% on VerilogEval-V2, surpassing all baselines. These results underline the effectiveness of MAGE in advancing the capabilities of LLM-driven coding systems.

MAGE not only outperforms closed-source systems but also demonstrates substantial improvement over accessible solutions, including vanilla models and open-source coding systems. For instance, compared to the highest-performing vanilla LLMs (e.g., Claude 3.5 Sonnet), MAGE achieves a relative improvement of +19.8% on VerilogEval-Human and +23.3% on VerilogEval-V2. Similarly, against open-source systems like OriGen (54.4% Pass@1 on VerilogEval-Human), MAGE significantly improves the functional correctness, showcasing its superior performance in scenarios where accessible and reproducible solutions are prioritized.

C. Ablation Study

Multi-Agent System. We conducted an ablation study to assess the effectiveness of task distribution among multiple agents. The study compared three configurations: (a) Vanilla LLM, which involves one-pass RTL code generation using a single LLM; (b) Single-Agent, where different agent roles in the MAGE system were merged into a single agent; and (c) Multi-Agent(). As shown in Table III, the multi-agent configuration achieves the highest pass rate (93.6%), outperforming both the vanilla (72.4%) and single-agent (83.9%) setups. These results indicate that effective task partitioning can significantly enhance the performance of LLM-based systems, particularly for complex tasks like RTL code generation, which require handling both synthesizable RTL code and non-synthesizable testbenches.
RTL Code State Checkpoint Mechanism. We also conducted case studies to evaluate the effectiveness of the proposed RTL Code State Checkpoint mechanism. For example, Fig. 3 illustrates a debugging case study with and without state checkpoints. Without checkpoints, when only the log is provided

TABLE II: Pass rates of recent LLMs and Coding Systems. The highest Pass@1 among different temperature settings is reported.

System	System Type	LLM Model	VerilogEval-Human Pass@1	VerilogEval-V2 Pass@1
Generic LLM	N/A	GPT-4o	51.3	N/A
	N/A	Claude 3.5 Sonnet	60.3	N/A
	N/A	Claude 3.5 Sonnet 2024-10-22	75.0	72.4
RTL-specified LLM	N/A	ITERTL [22]	42.9	N/A
	N/A	CodeV [23]	53.2	N/A
OriGen [13]	**Open Source**	DeepSeek-Coder-7B + LoRA	54.4	N/A
VeriAssist [17]	Closed Source	GPT-4	50.5	N/A
AutoVCoder [18]	Closed Source	CodeQwen1.5-7B	48.5	N/A
VerilogCoder [25]	Closed Source	GPT-4 Turbo	N/A	94.2
AIVRIL [24]	Closed Source	Claude 3.5 Sonnet*	64.7	N/A
MAGE (ours)	**Open Source**	Claude 3.5 Sonnet 2024-10-22	**94.8**	**95.7**
		Improvement(Δ)†	+19.8	+23.3

† The improvement is directly compared to the performance of the same model without employing the MAGE system.



Fig. 3: The Case Study of RTL Code State Checkpoint on Prob093.

to the LLM Debug Agent, the agent can only approximate the problematic location in the buggy RTL code, making it unlikely to identify and fix the issue. In contrast, with state checkpoints included in the log, the LLM Debug Agent can reason more effectively, pinpointing mismatches in the output and identifying missing logic terms. This leads to a more accurate and efficient bug-fixing process.

Sampling and Debugging Mechanisms. Our experimental data demonstrates the effectiveness of the proposed sampling and debugging mechanisms in enhancing the quality of generated RTL. Fig. 4(a) illustrates that the RTL generated with the sampling strategy consistently outperforms the RTL without sampling across different problems. The score distribution

TABLE III: Multi-Agent task distribution ablation study: Pass Rates with Claude 3.5 Sonnet 2024-10-22, Low-Temperature Setting.

Config	Type	VerilogEval-V2 Pass@1
Vanila LLM	Pass%	72.4
Single-Agent	Pass%	83.9
	Improvement(Δ)	**+11.5**
Multi-Agent	Pass%	93.6
	Improvement(Δ)	**+21.2**

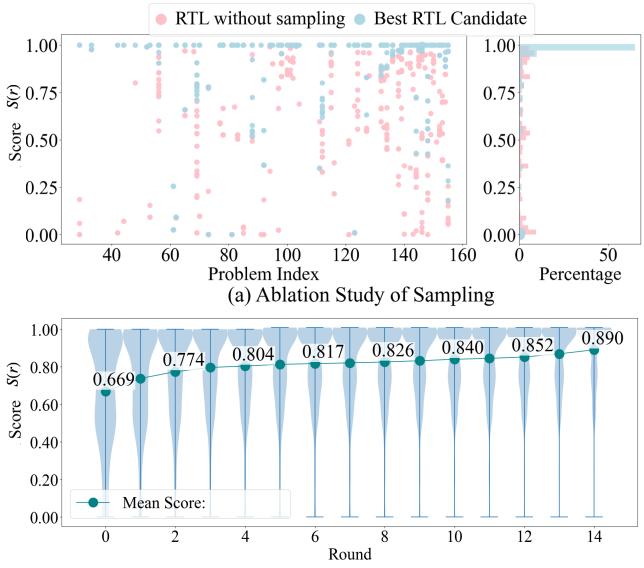


Fig. 4: Score $S(r)$ improvement of RTL by sampling and debugging. (a) Score distribution: generated RTL without sampling versus sampled and selected best RTL candidate; (b) Score distribution and the mean score of generated RTL in each debug round. Data of problems fixed before entering the debug stage are not included.

indicates that, without sampling, the RTL scores are nearly uniformly spread across the range $[0, 1]$. However, after applying the sampling method, the scores are concentrated near 1, reflecting a significant quality improvement. Fig. 4 (b) depicts the improvement in the mean score across multiple rounds of debugging, starting from an initial score of 0.669 and reaching 0.890 after sufficient refinement. This consistent increase in score demonstrates the cumulative benefit of iterative debugging, leading to more optimal RTL solutions.

V. Conclusion

In this paper, we propose MAGE, the first open-source LLM-based multi-agent system designed for automated and accurate Verilog RTL code generation. Integrated with a novel High-Temperature RTL Sampling and Scoring process, MAGE effectively explores more potentially correct candidates, leading to higher pass rates than prior studies. Augmented with RTL Debugging with State Checkpoint Mechanism, MAGE further optimizes the code with more precise feedback. Our system represents a critical step toward automating and optimizing hardware design workflows, offering a more robust methodology for AI-driven RTL design.

REFERENCES

[1] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, "HardFails: Insights into software-exploitable hardware bugs," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 213–230.

[2] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "On hardware security bug code fixes by prompting large language models," *IEEE Transactions on Information Forensics and Security*, 2024.

[3] K. Laeufer, B. Fajardo, A. Ahuja, V. Iyer, B. Nikolić, and K. Sen, "RTL-repair: Fast symbolic repair of hardware design code," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 867–881. [Online]. Available: https://doi.org/10.1145/3620666.3651346

[4] L. Lavagno, L. Scheffer, and G. Martin, *EDA for IC implementation, circuit design, and process technology*. CRC press, 2018.

[5] L. Scheffer, L. Lavagno, and G. Martin, *EDA for IC system design, verification, and testing*. CRC press, 2018.

[6] K. Wakabayashi and T. Okamoto, "C-based soc design flow and eda tools: An asic and system vendor perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1507–1522, 2000.

[7] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[8] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[9] Anthropic, "The claude 3 model family: Opus, sonnet, haiku," 2024. [Online]. Available: https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf

[10] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[11] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," *arXiv preprint arXiv:2402.03375*, 2024.

[12] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language model," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[13] F. Cui, C. Yin, K. Zhou, Y. Xiao, G. Sun, Q. Xu, Q. Guo, D. Song, D. Lin, X. Zhang *et al.*, "Origen: Enhancing RTL code generation with code-to-code augmentation and self-reflection," *arXiv preprint arXiv:2407.16237*, 2024.

[14] H. Pearce, B. Tan, and R. Karri, "Dave: Deriving automatically verilog from english," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 2020, pp. 27–32.

[15] S. Liu, W. Fang, Y. Lu, J. Wang, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[16] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu *et al.*, "ChipNemo: Domain-adapted LLMs for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[17] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards LLM-powered Verilog RTL assistant: Self-verification and self-correction," *arXiv preprint arXiv:2406.00115*, 2024.

[18] M. Gao, J. Zhao, Z. Lin, W. Ding, X. Hou, Y. Feng, C. Li, and M. Guo, "Autovcoder: A systematic framework for automated Verilog code generation using llms," *arXiv preprint arXiv:2407.18333*, 2024.

[19] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[20] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[21] N. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, "Revisiting verilogeval: Newer LLMs, in-context learning, and specification-to-RTL tasks," *arXiv preprint arXiv:2408.11053*, 2024.

[22] P. Wu, N. Guo, X. Xiao, W. Li, X. Ye, and D. Fan, "Itertl: An iterative framework for fine-tuning llms for rtl code generation," *arXiv preprint arXiv:2407.12022*, 2024.

[23] Y. Zhao, D. Huang, C. Li, P. Jin, Z. Nan, T. Ma, L. Qi, Y. Pan, Z. Zhang, R. Zhang *et al.*, "Codev: Empowering LLMs for verilog generation through multi-level summarization," *arXiv preprint arXiv:2407.10424*, 2024.

[24] H. Sami, P.-E. Gaillardon, V. Tenace *et al.*, "Aivril: Ai-driven rtl generation with verification in-the-loop," *arXiv preprint arXiv:2409.11411*, 2024.

[25] C.-T. Ho, H. Ren, and B. Khailany, "Verilogcoder: Autonomous Verilog coding agents with graph-based planning and abstract syntax tree (AST)-based waveform tracing tool," *arXiv preprint arXiv:2408.08927*, 2024.

[26] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[27] Q. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, Z. Wang, L. Shen, A. Wang, Y. Li *et al.*, "Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x," *arXiv preprint arXiv:2303.17568*, 2023.

[28] Y. Zhu, J. Li, G. Li, Y. Zhao, Z. Jin, and H. Mei, "Hot or cold? adaptive temperature sampling for code generation with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, 2024, pp. 437–445.

[29] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui, "Agentcoder: Multi-agent-based code generation with iterative testing and optimisation," *arXiv preprint arXiv:2312.13010*, 2023.

[30] OpenAI, "Hello GPT-4o," 2024, [Online; accessed 15-November-2024]. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[31] S. Williams, "The icarus verilog compilation system," 2008. [Online]. Available: https://github.com/steveicarus/iverilog

[32] J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github.com/jerryjliu/llama_index

[33] Anthropic, "Raising the bar on SWE-bench Verified with Claude 3.5 Sonnet," 2024, [Online; accessed 15-November-2024]. [Online]. Available: https://www.anthropic.com/research/swe-bench-sonnet

[34] S. Swaroopa, R. Mukherjee, A. Debnath, and R. S. Chakraborty, "Evaluating large language models for automatic register transfer logic generation via high-level synthesis," *arXiv preprint arXiv:2408.02793*, 2024.

Construction of DAG Models for Autonomous Systems

Jing Huang[1], Kuan Jiang[1], Weijie Wang[2], Wei Liang[1], and Wanli Chang[2]
[1]School of Computer Science and Engineering, Hunan University of Science and Technology, China
[2]Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, China
{huangjing@hnust.edu.cn, jkuan2022@163.com, weijiewang@hnu.edu.cn, wliang@hnust.edu.cn, wanli.chang.rts@gmail.com}

***Abstract*—Directed Acyclic Graphs (DAGs) are widely deployed as task models in autonomous systems, including vehicles and drones, to capture functional dependency. DAG scheduling has been extensively investigated by various communities to shorten makespan, under the common assumption that the model itself is given a priori. This work studies a rarely touched problem — construction of DAG models — and considers time-triggered blended task chains predominant in autonomous systems. We report representation semantics and a topology optimization method. Experiments show that the average end-to-end response time reduction is 4.8 times of the conventional Floyd algorithm. Our time complexity is $\mathcal{O}(n^2)$, making it suitable for handling dynamic tasks as well.**

I. Introduction

Autonomous systems are increasingly applied in fields such as vehicles and drones. These systems typically need to perform a series of tasks in real-time to interpret the environment and make behavioral decisions. For instance, autonomous vehicles utilize multiple sensors to collect environmental data, and then process these data through a sequence of tasks such as data preprocessing and data fusion, which enable the identification of pedestrians, obstacles, and road signs in real-time. Beside the timely requirements, autonomous systems are usually resource-constrained. To enhance the system's real-time performance and resource utilization, much research has focused on task scheduling. However, the construction of task models is often overlooked. Most studies assume that task models and their dependencies are predefined, but in reality, the way tasks are modeled directly influences the effectiveness of scheduling. The design of task models, including how tasks and their relationships are abstracted, is crucial to the resulting performance of task execution.

Tasks in autonomous systems can be either time-triggered or event-triggered, but the majority are time-triggered, with each task having its own job release period. For example, in Autonomous Driving Systems (ADS), data sampling tasks are periodically triggered by various sensors, such as LiDAR and cameras [1]. Different tasks may be organized into one or more task chains to accomplish complex system functions [2]. Therefore, task dependencies must be carefully designed to enable their effective collaboration execution. Modeling these tasks as periodic Directed Acyclic Graphs (DAGs) facilitates task scheduling and behavior analysis, especially when tasks are synchronized over time, as DAGs effectively capture execution order and inter-task dependencies [3]. However, constructing a DAG that accurately reflects task dependencies and optimizes system performance presents significant challenges due to the complex data dependencies and varying periods of individual tasks.

We note that several methods have been proposed to address DAG modeling and conversion in autonomous systems. For instance, Verucchi et al. [4] introduced a method for converting a multi-rate DAG task set with time constraints into a single-rate DAG, aiming to optimize schedulability and end-to-end latency. Zhao et al. [3] focused on DAG scheduling and analysis by modeling parallelism and dependencies. Similarly, Sun et al. [5] proposed an integrated framework to analyze the schedulability of individual tasks and the end-to-end delay of task chains in a DAG. While these works consider DAG construction, they do not sufficiently address the execution efficiency of the resulting DAGs. Moreover, they emphasize using scheduling algorithms to enhance system performance.

In fact, purely relying on scheduling algorithms has limited potential to further improve time performance and resource utilization, as many algorithms are already approaching their optimal performance. However, approaching the problem from the perspective of task modeling may offer a new breakthrough. This paper delves into the construction of DAG models that accurately represents all jobs released by each task modules within a hyper-period, along with their dependencies. Our main contributions are as follows:

- A novel DAG construct approach is proposed. The approach first represents the jobs released by each task module on the chain within the hyper-period as vectors, and then apply the Kronecker product to obtain a matrix capturing all jobs and their relationships based on module dependencies, thereby constructing the initial DAG.
- A DAG simplification method, DRM, is reported. DRM can enhance the execution efficiency of the constructed DAG by reducing its redundant edges, leading to high average waiting times for jobs. Compared to the Floyd algorithm, which has a time complexity of $O(n^3)$, DRM

This work was supported by the National Key Research and Development Program of China (2023YFB4503704), the National Natural Science Fund Joint Key Project of China (U2468205), and the Hunan Natural Science Foundation (2023JJ30263). Corresponding author: Wanli Chang.

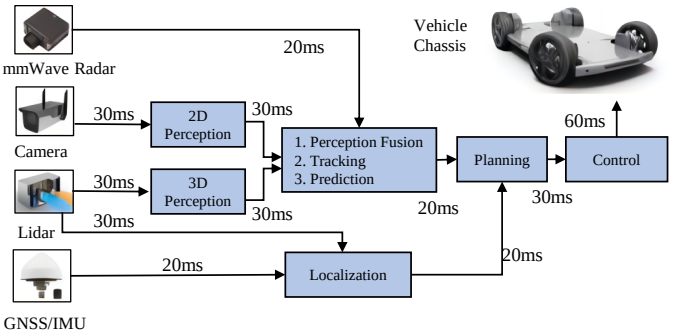


Fig. 1. An example of an ADAS (Advanced Driver Assistance System) in an intelligent vehicle

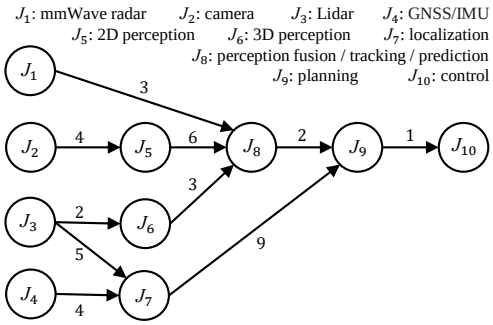


Fig. 2. Modeling the ADAS in Fig. 1, which includes 10 task modules and their dependencies.

achieves a lower time complexity of $O(n^2)$. Furthermore, to demonstrate that our method achieves a shorter average end-to-end response time, we used classic scheduling algorithms like HEFT and PEFT to schedule three types of DAGs: the unoptimized DAG, the DAG optimized by the Floyd algorithm, and our optimized DAG. Experimental results indicate that the Floyd algorithm-optimized DAG reduces the average end-to-end response time by about 2%, while our optimized DAG achieves a reduction of approximately 9.6%.

These contributions address existing limitations by providing a flexible, efficient task management framework that improves real-time responsiveness and resource allocation in autonomous systems, ultimately advancing their operational robustness.

II. Models and Problem Statement

A. Function Model

In autonomous systems, a function is often regarded as a task chain consisting of a series of task modules that periodically release jobs. These task modules can operate either sequentially or in parallel. Fig. 1 illustrates the ADAS (Advanced Driver Assistance System) in an intelligent vehicle, where modules such as Data Acquisition (DA), Data Perception (DP), Data Fusion (DF), Path Planning (PP), and Body Control (BC) form a task chain to collaboratively execute tasks. Each module has its own period; for example, the camera samples frames every 30ms, while the BC operates every 60ms. Clearly, this is an example of a directed acyclic graph of periodic tasks with data dependencies.

A function's major feature is reflected in the task modules it has and the relationships between them. Therefore, we model a function as: $F = \{J_1, J_2, \ldots, J_m, \boldsymbol{R}\}$. Here, $J_1, J_2, \ldots, J_m$ represent the task modules of the function, each defined as a tuple $J_i = (t_i, EC_i)$, where t_i denotes the time interval at which module J_i releases a job, and EC_i represents its execution cost. The term $\boldsymbol{R}$ represents the relationships between the m modules and is given by the following matrix:

$$\boldsymbol{R} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,m} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,m} \end{pmatrix}. \quad (1)$$

- $r_{i,j} = 0$ represents that there is no dependency between modules J_i and J_j. Modules J_i and J_j can be executed in parallel.
- $r_{i,j} = 1$ indicates that module J_i is a direct predecessor of module J_j. In this case, an additional communication cost $C_{i,j}$ is often required to send the processing results of J_i to J_j.

Fig. 2 illustrates an example of a function model with 10 modules abstracted from the ADAS shown in Fig. 1. The related parameters of modules are listed in Table I.

TABLE I
Parameter settings for each module in Fig. 2

Modules	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
Job period	20	30	30	20	30	30	20	20	30	60
Execution cost	10	9	6	5	8	7	10	6	5	2

B. End-to-End Response Time Model

The jobs activated by module J_i within a hyper-period T can be denoted as the set $\boldsymbol{J}_i = \{J_{i,1}, J_{i,2}, \ldots, J_{i,T/t_i}\}$. Similarly to existing studies, we set $T = lcm(t_1, t_2, \ldots, t_m)$. Correspondingly, the set of all jobs can be expressed as $\boldsymbol{J} = \{\boldsymbol{J}_1 \cup \boldsymbol{J}_2 \cup \cdots \cup \boldsymbol{J}_m\}$. If $r_{i,j} = 1$, indicating that module J_i is the predecessor of J_j, it is essential to ensure that no data is lost during the transmission from the predecessor to the successor within the hyper-period T [6]. To achieve this, every job $J_{j,k'} \in \boldsymbol{J}_j$ must have a corresponding precursor $J_{i,k} \in \boldsymbol{J}_i$, and conversely, each $J_{i,k} \in \boldsymbol{J}_i$ must have a corresponding successor $J_{j,k'} \in \boldsymbol{J}_j$. For convenience, we denote the sets of predecessors and successors of $J_{i,k}$ as $pre(J_{i,k})$ and $suc(J_{i,k})$, respectively. A job with no predecessors is referred to as an entry job J_{entry}. Similarly, a job with no successors is referred to as an exit job J_{exit}. A function can have multiple entry jobs, but it usually produces only one output. Therefore, we assume that the function has a unique exit job J_{exit}, and use $\boldsymbol{J}_{\text{entry}}$ to represent the set all of entry jobs, $\boldsymbol{J}_{\text{entry}} = \{J_{i,k} \mid J_{i,k} \in \boldsymbol{J},\ |pre(J_{i,k})| = 0\}$.

The end-to-end response time refers to the time from the arrival of the entry job released by module J_i to the completion of the unique exit job J_{exit}, and is given by

$$RT(J_{i,k}) = FT(J_{\text{exit}}) - AT(J_{i,k}),\ J_{i,k} \in \boldsymbol{J}_{\text{entry}} \cap \boldsymbol{J}_i. \quad (2)$$

Here, $AT(J_{i,k})$ represents the arrival time of $J_{i,k}$, which can be obtained by adding an offset of $k-1$ periods to $AT(J_{i,1})$.

$$AT(J_{i,k}) = AT(J_{i,1}) + (k-1) \times t_i. \tag{3}$$

$FT(J_{\text{exit}})$ is the finish time of the exit job J_{exit}. Since the focus of this paper is on the construction of the model rather than the scheduling strategy, we directly use classic algorithms, such as HEFT [7] and PEFT [8], to calculate $FT(J_{\text{exit}})$.

Since each task generates a potentially infinite sequence of jobs, a schedulability interval is needed to capture the overall system behavior [6]. For this interval, Goossens et al. [9] proposed a calculation method. Within the hyper-period T, multiple entry jobs can be released. We calculate the average end-to-end response time for all entry jobs in $\boldsymbol{J}_{\text{entry}}$ as follows:

$$ART(\boldsymbol{J}_{\text{entry}}) = \frac{1}{|\boldsymbol{J}_{\text{entry}}|} \times \sum_{J_{\text{entry}} \in \boldsymbol{J}_{\text{entry}}} RT(J_{\text{entry}}). \tag{4}$$

C. Problem Statement

With the above models, our problem can be formulated as follows: Given a function $F = \{J_1, J_2, \ldots, J_m, \boldsymbol{R}\}$, along with the job release period t_i and execution cost EC_i for each task module J_i ($1 \le i \le m$), the goal is to construct a DAG that can capture the dependencies of the jobs $\boldsymbol{J}$ released from the function within a hyper-period $T = lcm(t_1, t_2, \ldots, t_m)$. The resulting DAG should minimize the average end-to-end response time of jobs in the function as much as possible when scheduled by existing algorithms.

III. DAG Construction and Optimization

Directed Acyclic Graph (DAG) modeling not only clearly represents job dependencies but also facilitates parallelism analysis.

A. DAG Construction

Each task module J_i actives a job at intervals t_i time. Within a given time T, the number of jobs activated by module J_i is T/t_i. For the function with m task modules, the number of activated jobs within time T is

$$n = \sum_{i=1}^{m} T/t_i. \tag{5}$$

To represent the relationships among the n jobs, we construct a DAG $\boldsymbol{G}$ and represent it as an adjacency matrix, denoted as:

$$\boldsymbol{G} = \begin{pmatrix} \boldsymbol{G}_{1,1} & \boldsymbol{G}_{1,2} & \ldots & \boldsymbol{G}_{1,m} \\ \boldsymbol{G}_{2,1} & \boldsymbol{G}_{2,2} & \ldots & \boldsymbol{G}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{G}_{m,1} & \boldsymbol{G}_{m,2} & \ldots & \boldsymbol{G}_{m,m} \end{pmatrix}_{n \times n}. \tag{6}$$

Each element $\boldsymbol{G}_{i,j}$ is a sub-matrix of size $T/t_i \times T/t_j$, representing the dependency between the T/t_i jobs of module J_i and the T/t_j jobs of task module J_j.

It is evident that the value of $\boldsymbol{G}_{i,j}$ depends on the relationship between modules J_i and J_j. Generally, jobs within the same module are independent of each other, while the dependency between jobs from different modules is determined by the dependency between those modules. If module J_i is a direct predecessor of module J_j, i.e., $r_{i,j} = 1$, then any job in $\boldsymbol{J}_i$ may be a predecessor of each job in $\boldsymbol{J}_j$. Since the dependencies between modules are defined by the matrix $\boldsymbol{R}$, $\boldsymbol{G}_{i,j}$ can be calculated as follows:

$$\boldsymbol{G}_{i,j} = r_{i,j} \times \left(\mathbf{1}_{T/t_i \times 1} \otimes \mathbf{1}_{1 \times T/t_j}\right). \tag{7}$$

Here, $\mathbf{1}_{T/t_i \times 1}$ is a $T/t_i \times 1$ row vector with all elements equal to "1"; $\mathbf{1}_{1 \times T/t_j}$ is a $1 \times T/t_j$ column vector with all elements equal to "1"; the sign "$\otimes$" stands for the Kronecker product. Note that the sub-matrix $\boldsymbol{G}_{i,j}$ represents a temporarily constructed redundancy dependency between jobs, rather than an inherent characteristic of the jobs.

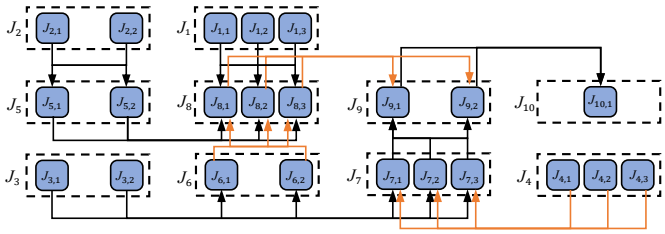


Fig. 3. The detailed structure of the DAG $\boldsymbol{G}$ with 23 jobs and 58 dependencies.

Take the ADAS in Fig. 1 as an example. Given time $T = 60$, we can construct a DAG $\boldsymbol{G}$ with 23 jobs and 58 dependencies, as shown in Fig. 3. To distinguish the different dependencies among jobs, we use different colors to label them in Fig. 3.

B. DAG Structure Optimization

Although we can use Eqs. (6) and (7) to construct a DAG representing the desired function, the resulting DAG is complex due to the intricate interconnections and hierarchical structure, complicating scheduling and parallel processing. Therefore, optimizing the DAG structure is crucial to enhance parallelism and execution efficiency.

1) Lower Bound Analysis of Dependencies: The DAG structure can be simplified by deleting redundant dependencies between jobs. According to ahead definition, the dependencies between jobs $\boldsymbol{J}_i$ and $\boldsymbol{J}_j$ are captured by $\boldsymbol{G}_{i,j}$. For convenience, we represent the element of $\boldsymbol{G}_{i,j}$ as $g_{k,k'}$, $g_{k,k'} \in \{0, 1\}$. In a real function, each task module transmits its processed results to the subsequent modules. To ensure the normal execution of the function, the following conditions for each job $J_{i,k} \in \boldsymbol{J}$ must be satisfied while optimizing the structure of the DAG $\boldsymbol{G}$.

- Satisfy the reachability of each job to the unique exit job;
- If $r_{i,j} = 1$, each job $J_{i,k} \in \boldsymbol{J}_i$ has at least one successor $J_{j,k'}$ released from module J_j, that is,

$$\sum_{k'=1}^{T/t_j} g_{k,k'} \ge 1; \tag{8}$$

- If $r_{i,j} = 1$, each job $J_{j,k'} \in \boldsymbol{J}_j$ has at least one predecessor $J_{i,k}$ released from module J_i, i.e,

$$\sum_{k=1}^{T/t_i} g_{k,k'} \ge 1. \tag{9}$$

Eqs. (8) and (9) ensure that each job can receive data from its predecessors and transmit its output to its successors.

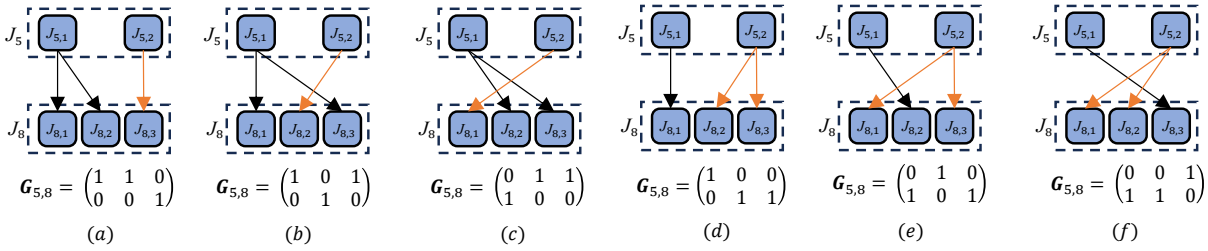


Fig. 4. 6 simple structures for sub-matrix $\boldsymbol{G}_{5,8}$ to improve parallelism and execution efficiency.

Theorem 1. *The minimum number of dependencies of DAG* $\boldsymbol{G}$ *is* $\sum_{i=1}^{m}\sum_{j=1}^{m} r_{i,j} \times max\{T/t_i, T/t_j\}$.

Proof: For each sub-matrix $\boldsymbol{G}_{i,j}$, when $t_i \le t_j$ (or $t_i > t_j$), setting $\sum_{k'=1}^{T/t_j} g_{k,k'} = 1$ (or $\sum_{k=1}^{T/t_i} g_{k,k'} = 1$) for each job $J_{i,k}$ (or $J_{j,k'}$), we can obtain that $\sum_{k=1}^{T/t_i}\sum_{k'=1}^{T/t_j} g_{k,k'} = \max\{T/t_i, T/t_j\}$. This setting ensures that both jobs $J_{i,k}$ and $J_{j,k'}$ satisfy the constraints of Eqs. (8) and (9), respectively. The same goes for other sub-matrices. ∎

Taking $\boldsymbol{G}_{5,8}$ in Eq. (6) as an example, $t_5 = 30$, $t_8 = 20$ and $T = 60$, so the minimum number of dependencies in sub-matrix $\boldsymbol{G}_{5,8}$ is $\max\{60/30, 60/20\} = 3$. Then, the minimum number of dependencies in matrix $\boldsymbol{G}$ in Fig. 3 is 27. Theorem 1 provides the minimum number of dependencies for the DAG, but identifying which redundant dependencies to delete requires further exploration. While there are various ways to remove redundant dependencies, our goal is to enhance the parallelism and execution efficiency of the DAG through their removal.

2) Removal of Dependencies: In DAG task scheduling, parallelism refers to the number of tasks that can be started simultaneously without violating dependencies. Therefore, a DAG with fewer dependencies has a relatively high parallelism. From Eq. (7), it follows that $\boldsymbol{G}_{i,j}$ is a $T/t_i \times T/t_j$ matrix, with all elements equal to "1" when $r_{i,j} = 1$. According to Theorem 1, the number of "1" in $\boldsymbol{G}_{i,j}$ can be reduced to $\max\{T/t_i, T/t_j\}$. In other words, $T/t_i \times T/t_j - \max\{T/t_i, T/t_j\}$ dependencies can be removed from $\boldsymbol{G}_{i,j}$ without compromising the correctness of the DAG.

To determine which dependencies can be removed, we assume the simplified $\boldsymbol{G}'_{i,j}$ as

$$\boldsymbol{G}'_{i,j} = \begin{pmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,T/t_j} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,T/t_j} \\ \vdots & \vdots & \ddots & \vdots \\ g_{T/t_i,1} & g_{T/t_i,2} & \cdots & g_{T/t_i,T/t_j} \end{pmatrix}. \tag{10}$$

Based on Eq. (10), the problem of removal of dependencies becomes to solve the following equation.

$$\begin{cases} \sum_{k=1}^{T/t_i}\sum_{k'=1}^{T/t_j} g_{k,k'} = \max\left\{\frac{T}{t_i}, \frac{T}{t_j}\right\}, g_{k,k'} \in \{0,1\}; & \text{(I)} \\ g_{k,1} + \cdots + g_{k,\frac{T}{t_j}} \ge 1,\ \forall k, 1 \le k \le \frac{T}{t_i}; & \text{(II)} \\ g_{1,k'} + \cdots + g_{\frac{T}{t_i},k'} \ge 1,\ \forall k', 1 \le k' \le \frac{T}{t_j}. & \text{(III)} \end{cases} \tag{11}$$

It is obvious that Eq. (11) has multiple solutions.

Theorem 2. *The number of solutions that satisfy Eq. (11) is*

$$N = \binom{ab}{c} - a\binom{(a-1)b}{c} - b\binom{a(b-1)}{c} + ab\binom{(a-1)(b-1)}{c}, \tag{12}$$

where $a = T/t_i$, $b = T/t_j$, *and* $c = max\{a, b\}$.

Proof: The total number of choices to place "1" in c positions in the matrix $\boldsymbol{G}'_{i,j}$ is $N_{(\mathrm{I})} = \binom{ab}{c}$, satisfying condition (I) in Eq. (11). Condition (II) indicates that each row at least has one "1", which is similarly to Condition (III) corresponding to each column. The number of options that fail to meet them are $N_{(\mathrm{II})} = a\binom{(a-1)b}{c}$ and $N_{(\mathrm{III})} = c\binom{a(b-1)}{c}$, respectively. The case that both a row and a column have no "1" has $N_{(\mathrm{II})\&(\mathrm{III})} = ab\binom{(a-1)(b-1)}{c}$ options. Therefore, the number of solutions satisfying Eq. (11) is given by $N = N_{(\mathrm{I})} - N_{(\mathrm{II})} - N_{(\mathrm{III})} + N_{(\mathrm{II})\&(\mathrm{III})}$. ∎

Each solution corresponds to a specific structure of the DAG $\boldsymbol{G}$. Although these structures have the same number of dependencies, they may have difference execution efficiency. For sub-matrix $\boldsymbol{G}_{i,j}$, to demonstrate the impact of different structures on execution efficiency, we consider the Average Waiting Time (*AWT*) of jobs $\boldsymbol{J}_j$, which is defined as

$$AWT(\boldsymbol{J}_j) = \frac{1}{|\boldsymbol{J}_j|} \times \sum_{J_{j,k'} \in \boldsymbol{J}_j} WT(J_{j,k'}), \tag{13}$$

where $WT(J_{j,k'})$ represent the waiting time of job $J_{j,k'}$, measured from their release time to their start time, and is defined as

$$WT(J_{j,k'}) = ST(J_{j,k'}) - (k'-1) \times t_j. \tag{14}$$

If $g_{k,k'} = 1$ in $\boldsymbol{G}_{i,j}$, the start time of job $J_{j,k'}$ is given by

$$ST(J_{j,k'}) = ST(J_{i,k}) + EC_i + C_{i,j}. \tag{15}$$

Note that the start time in Eq. (14) does not take the availability of processors into account, as specific scheduling strategies are not employed and processor contention is not considered. Therefore, the start time of job $J_{i,k} \in \boldsymbol{J}_i$ can be considered as its release time, $ST(J_{i,k}) = (k-1) \times t_i$.

Illustration. We use $\boldsymbol{G}_{5,8}$, which illustrates the dependencies between jobs $\boldsymbol{J}_5$ and $\boldsymbol{J}_8$, as an example. According to Theorem 2, the number of all solutions of $\boldsymbol{G}_{5,8}$ that satisfy Eq. (11) is calculated as follows, $N_{\boldsymbol{G}_{5,8}} = \binom{6}{3} - 2\binom{3}{3} - 3\binom{4}{3} + 6\binom{2}{3} = 6$. Fig. 4 shows 6 possible structures ($a \sim f$) for $\boldsymbol{G}_{5,8}$.

Algorithm 1 Dependencies Removal Method (DRM)

Input: $\boldsymbol{G}$, m, T
Output: The simplified DAG $\boldsymbol{G}$

1: **for** i from 1 to m **do**
2: **for** j from 1 to m **do**
3: **if** $T/t_i \geq T/t_j$ **then** $\boldsymbol{G}_{i,j} = \boldsymbol{G}_{i,j}^{\mathrm{T}}$;
4: $N_{\mathrm{r}} = \min\{T/t_i, T/t_j\}$; $N_{\mathrm{c}} = \max\{T/t_i, T/t_j\}$;
5: **if** $N_{\mathrm{r}} > 1$ **then**
6: **for** k from N_{r} to 2 **do**
7: *minWT* = $+\infty$; // The minimal waiting time.
8: **for** k' from N_{c} to 1 **do**
9: $wt = WT(J_{j,k'})$ // using Eq. (14)
10: $minWT = wt < minWT \ ? \ wt : minWT$;
11: **end for**
12: Remove the dependencies from the sub-matrix $\boldsymbol{G}_{i,j}$ with $WT(J_{j,k'}) > minWT$;
13: **end for**
14: **end if**
15: **if** $T/t_i \geq T/t_j$ **then** $\boldsymbol{G}_{i,j} = \boldsymbol{G}_{i,j}^{\mathrm{T}}$;
16: **end for**
17: **end for**

Some parameters are given in the functionality model: $t_5 = 30$, $t_8 = 20$, $ET_8 = 8$, and $C_{5,8} = 6$. For structure *(a)* in Fig. 4,

$$AWT(\boldsymbol{J}_8) = \big((8+6)+(20-20)+(30+8+6-40)\big)/3 = 6.$$

The average waiting times for the other five structures are 12.33, 14.67, 14.0, 16.0, and 22.67, respectively. Obviously, structure *(a)* in Fig. 4 accelerates the start of each job in $\boldsymbol{J}_8$, resulting in higher execution efficiency compared to the others.

Algorithm description. With the execution efficiency in mind, we design the DRM (Dependencies Removal Method) approach as shown in Algorithm 1. The DRM optimizes each sub-matrix $\boldsymbol{G}_{i,j}$, and their operations are concentrated in lines 5-13. Lines 7-12 remove redundant dependencies with large waiting time between jobs $J_{i,k}$ and $\boldsymbol{J}_j$. By calling DRM for each sub-matrix, we obtain the simplified DAG $\boldsymbol{G}$. Fig. 5 shows the simplified structure of the DAG $\boldsymbol{G}$ in Fig. 3, consisting of 23 jobs and 27 dependencies. Compared to the original DAG, 31 dependencies have been removed.

Time complexity analysis. The most computationally intensive operation occurs in Lines 9 and 10, iterating through all elements of the matrix $\boldsymbol{G}$. Lines 9 and 10 require a total execution count given by $N_{et} = \sum_{i=1}^{m}\sum_{k=1}^{T/t_i}\sum_{j=1}^{m}\sum_{k'=1}^{T/t_j} 1$. According to Eq. (5), the number of jobs is $n = \sum_{i=1}^{m} T/t_i$. Obviously, we have $N_{et} = n \times n$. Therefore, the time complexity of the DRM algorithm is equal to $\mathcal{O}(n^2)$. Since DRM only traverses the edges of the DAG, its time complexity is lower compared to other optimization algorithms. The time complexity of the Floyd-Warshall algorithm is $\mathcal{O}(n^3)$, the Dijkstra algorithm (multi-source) is $\mathcal{O}(n^2(\log n + n))$, and the Bellman-Ford algorithm (multi-source) is $\mathcal{O}(n^4)$.

IV. Experimental Results and Discussion

A. Experiment Setup

To validate the effectiveness of our task modeling and optimization approach, we conduct extensive simulation experiments and compare it with *Floyd* [10]. All experiments are simulated using Python on a computer with an 11th Gen Intel(R) Core(TM) i7-11800H CPU at 2.30 GHz and 32.0 GB of RAM.

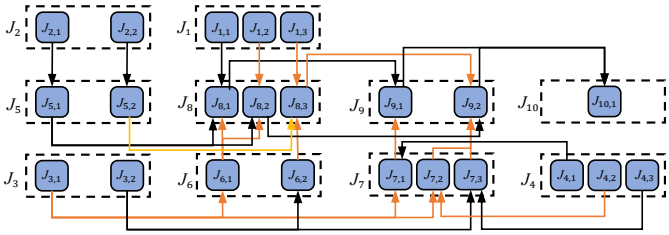


Fig. 5. A simplified structure of the DAG $\boldsymbol{G}$ in Fig. 3 with 23 jobs and 27 dependencies.

*1) **Workload Settings**:* The proposed DMR algorithm constructs the DAG based on the constraints between task modules, job periods, and loads. In our experiments, these parameters for task modeling are set as follows:

- *Number of modules.* $m \in \{10, 15, 20, 25\}$.
- *Module properties.* Each module J_i will active a job with execution cost EC_i at intervals t_i time, where $EC_i \in [1, 10]$ms and $t_i \in \{10, 25, 50\}$ms. The communication cost between modules J_i and J_j is set to $C_{i,j} \in [1, 4]$ms, $1 \leq i, j \leq m$.
- *Dependencies between modules.* We utilize a dependency generation tool sourced from the open-source project in [11] to simulate the relationship matrix $\boldsymbol{R}$ between modules. In this project, it is necessary to specify the number of modules, the in-degree and out-degree for each module (chosen from the set $\{1, 2, 3, 4, 5\}$), and the probability of each dependency being added (chosen from the set $\{0.3, 0.4, 0.5, 0.6\}$).

*2) **Platform Configuration**:* We consider computing platforms containing 8 identical processors.

*3) **Comparison Metrics**:* This section introduces two comparison metrics used in the experiments, as follow:

- *Average Waiting Time* (*AWT*). Refer to Eq. (13).
- *Average end-to-end Response Time* (*ART*). An important indicator of system execution efficiency.

B. Effectiveness of Task Modeling and Optimization

According to our workload settings, we will first construct numerous DAGs. Next, we shall simplify these DAGs using our approach DRM and the Floyd algorithm, respectively. Finally, we schedule both the simplified and unsimplified DAGs using the well-known HEFT and PEFT algorithms and compare their respective results. In our experiments, 1000 DAGs are synthesized with different numbers of modules.

*1) **Experiment 1**:* The results of scheduling using HEFT are shown in Fig. 6. We observe that the unsimplified DAGs have a higher *AWT* compared to the simplified cases using Floyd and DRM. Our proposed DRM achieves the smallest *AWT* compared with the unsimplified and Floyd. The specific *AWT* values for the unsimplified, Floyd, and DRM are 53.43ms, 50.18ms, and 42.52ms, respectively. Compared

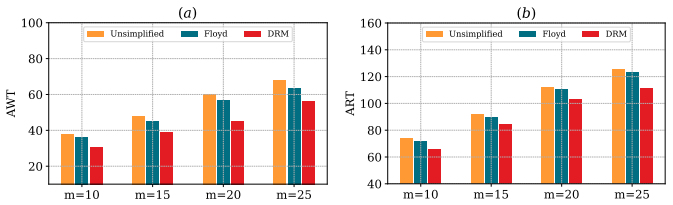


Fig. 6. Results of *AWT* and *ART* for the unsimplified, Floyd, and DRM, obtained by HEFT.

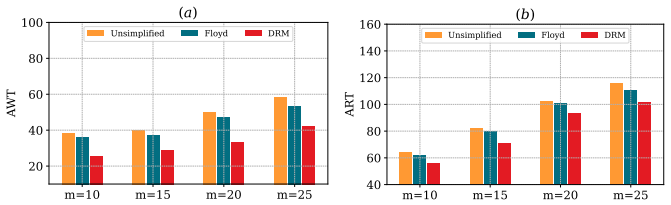


Fig. 7. Results of *AWT* and *ART* for the unsimplified, Floyd, and DRM, obtained by PEFT.

with unsimplified, Floyd and DRM reduce *AWT* by 6.08% and 20.42%, respectively. Clearly, Floyd has a smaller *AWT* compared to the unsimplified case, and our proposed DRM is smaller than Floyd. Through calculation, the *ART* values of the unsimplified, Floyd, and DRM are 101.04ms, 99.05ms, and 91.31ms, respectively. Compared with unsimplified, Floyd and DRM reduce *ART* by 1.97% and 9.63%, respectively. In summary, the relationship between the *AWT* and *ART* values for the unsimplified, Floyd, and DRM methods can be summarized as follows:

$$\begin{cases} AWT_{\text{Unsimplified}} > AWT_{\text{Floyd}} > AWT_{\text{DRM}}, \\ ART_{\text{Unsimplified}} > ART_{\text{Floyd}} > ART_{\text{DRM}}. \end{cases} \tag{16}$$

Therefore, we can be concluded that reducing the *AWT* of jobs can reduce the end-to-end response time to some extent. This also indicates that considering the *AWT* of jobs during the DAG structural optimization process is advisable.

*2) **Experiment 2**:* The results of scheduling using PEFT are shown in Fig. 7. The specific *AWT* values for the unsimplified, Floyd, and DRM are 46.43ms, 43.18ms, and 32.27ms, respectively. Compared with the *AWT* values in ***Experiment 1***, the *AWT* values obtained by PEFT are smaller than those of HEFT, which is attributed to the scheduling advantages of PEFT over HEFT. The *ART* values of the unsimplified, Floyd, and DRM are 90.03ms, 88.30ms, and 80.30ms, respectively. We can conclude that Eq. (16) holds true using PEFT. Compared to the unsimplified , Floyd and DRM reduce the *AWT* (*ART*) by 7.0% (1.92%) and 30.5% (10.81%), respectively.

The above two sets of experiments further validate the importance of optimizing the DAG structure and the rationale for considering the average waiting time of jobs during the structure optimization process.

V. Conclusion

In this paper, we have investigated the critical role of DAG construction in enhancing the real-time performance and resource utilization of autonomous systems, particularly in environments that have required intelligent decision-making across multiple task modules. We have introduced a novel approach to DAG abstraction and simplification that has considered task dependencies within a hyper-period, emphasizing both accurate representation and improved parallelism.

The proposed method has leveraged the Kronecker product to construct an initial DAG and has introduced an innovative edge simplification technique that has reduced average job waiting time while maintaining computational efficiency. Compared to traditional methods such as the Floyd algorithm, our approach has achieved a significant reduction in time complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ and has delivered superior performance in reducing the average response time of task chains. Experimental results have confirmed that the DAG optimized using our method has reduced the average response time by approximately 9.6%, significantly outperforming the 2% improvement achieved by the Floyd algorithm.

This work has highlighted the importance of DAG construction as a foundational step in performance optimization through scheduling strategy and has laid the groundwork for future research into real-time task management in autonomous systems. Potential future directions include extending the proposed methods to dynamic task environments and exploring integration with adaptive scheduling algorithms for enhanced robustness and scalability.

References

[1] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 159–169, IEEE, 2016.

[2] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–24, 2019.

[3] S. Zhao, X. Dai, and I. Bate, "Dag scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4019–4038, 2022.

[4] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 226–238, IEEE, 2020.

[5] J. Sun, K. Duan, X. Li, N. Guan, Z. Guo, Q. Deng, and G. Tan, "Real-time scheduling of autonomous driving system with guaranteed timing correctness," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 185–197, IEEE, 2023.

[6] J. Chen, C. Du, P. Han, and X. Du, "Work-in-progress: Non-preemptive scheduling of periodic tasks with data dependency upon heterogeneous multiprocessor platforms," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 540–543, IEEE, 2019.

[7] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[8] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[9] J. Goossens, E. Grolleau, and L. Cucu-Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms," *Real-time systems*, vol. 52, pp. 808–832, 2016.

[10] A. E. Mirino *et al.*, "Best routes selection using dijkstra and floyd-warshall algorithm," in *2017 11th International Conference on Information & Communication Technology and System (ICTS)*, pp. 155–158, IEEE, 2017.

[11] https://github.com/laser/random-dag-generator-go, 2023.

ACIM-QMM: Efficient Analog Computing-in-Memory Accelerator for QC-MDPC McEliece Cryptosystem

Pingdan Xiao[1], Zhengmiao Wei[1], Sichun Du[1], Wanli Chang[1*] and Qinghui Hong[1*]
[1]Hunan University, Changsha, Hunan, China
*Email: wanli.chang.rts@gmail.com, hongqinghui@hnu.edu.cn

***Abstract*—Quasi-cyclic moderate density parity-check McEliece (QMM) cryptosystem is designed to mitigate the security threat posed by quantum computers, and is considered to be a promising candidate for post-quantum cryptography (PQC). However, the growing requirement of data encryption pose severe challenges for QMM implementation in terms of latency and hardware overhead. In this work, we firstly propose ACIM-QMM, an analog computing-in-memory (CIM) accelerator design for QMM cryptosystem. The use of analog circuits and CIM enables the design to efficiently generate key and encrypt ciphertext while breaking the performance bottleneck constrained by digital computing paradigm in PQC. In the experiment, ACIM-QMM can work in low relative error, and it can achieve 31.4× ∼ 288.1× speedup compared with SOTA hardware of QMM cryptosystem. Furthermore, the results indicate that ACIM-QMM can achieve a maximum of 3.12× area efficiency and 20.32× energy efficiency compared to other PQC hardware for 256-bit security.**

***Index Terms*—ReRAM, Circuit design, Analog Computing-in-Memory, QC-MDPC McEliece**

I. Introduction

Different from other post-quantum cryptography (PQC), code-based McEliece cryptography has maintained its security to this day [1], [2]. In the fourth round of NIST PQC standardization process, it was selected as a key encapsulation mechanism finalist [4], [6]. However, code-based cryptosystems inherently grapple with the challenge due to the large key sizes, which impose a significant burden on limited hardware resources and make existing implementations of the McEliece cryptosystem inefficient and impractical [1], [3], [5].

To achieve a compact key while maintaining security, numerous alternatives have been proposed to reduce the size of public key [2]. QC-MDPC McEliece (QMM) cryptosystem stands out as a notable implementation, requiring only 1.2 kilobytes for a 128-bit security level (SL) [5]. Security analyses and research have validated its potential, positioning it as a promising variant within McEliece-type cryptosystems [5]. However, despite its compact nature, further efficient and rapid designs are required to meet the urgent demands of massive data encryption in the context of exponential data growth.

Unlike Lattice-base and Hash-based PQC, code-based QMM cryptosystem requires numerous matrices operation [7], [8]. Therefore, it naturally fits with the crossbar-based computing-in-memory (CIM) circuits for matrices operation, which can avoid the bottleneck of data transmission [9]. Within CIM, different from digital CIM (DCIM) , analog CIM (ACIM) circuits can perform matrices operation in one step due to the computation speed of electromagnetic waves [9]. Additionally, ACIM significantly reduces the use of hardware resources by avoiding signal conversion [10]. However, existing implementation of QMM cryptosystem mainly depends on von-Neumann architecture and digital computing paradigm, the use of which have difficulty in satisfying the aforementioned requirement, and there is no ACIM accelerator design for QMM cryptosystem currently.

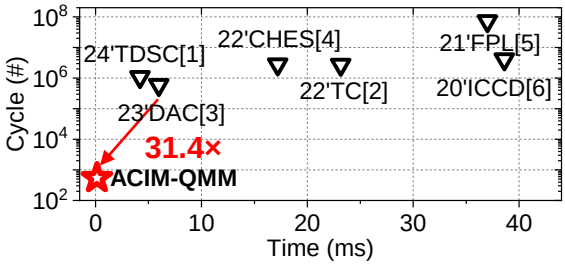


Fig. 1. Comparison with the state-of-the-arts.

Motivated by this, we introduce ACIM-QMM, an analog CIM accelerator leveraging resistive random-access memory (ReRAM) for rapid execution of QMM key generation and encryption. To address the limitation of analog circuits in performing matrix operations in Galois Fields (2) (which is abbreviated as GF (2)), we propose a data-flow deployment and accelerator design. This enables swift GF(2) matrix operations on analog circuits, achieving large-key computations within limited circuits, thereby drastically reducing hardware costs. ACIM-QMM stands out in high-speed, energy-efficient encryption hardware, efficiently tackling any SL task of QMM cryptosystem, outperforming traditional architecture and digital circuits. As illustrated in Fig. 1, ACIM-QMM significantly outpaces other hardware in 128-bit SL evaluations. We believe our design holds great potential for exploring analog circuits to empower PQC. The key contributions of this work are:

- **GF(2) Matrix Deployment:** Our data-flow scheme enables efficient GF(2) matrix operations in analog circuits, handling large-key computations for any SL QMM cryptosystem within limited resources.
- **ACIM-QMM Accelerator:** As the first ReRAM-based ACIM accelerator for QMM cryptosystem, ACIM-QMM significantly accelerates key generation and encryption while lowering hardware costs.
- **Performance Evaluations:** Comprehensive experiments showcase ACIM-QMM's superiority over other hardware, as illustrated in Fig. 1 and Section 4, with the proposed

strategies to mitigate non-ideal analog circuit factors, ensuring robust performance.

II. Background and Related Work

A. Key generation and encryption of QMM cryptosystem

According to the specification of QMM cryptosystem, its procedure of key generation and encryption is shown in Algorithm 1. Notably, the traditional approach to implementing QMM cryptosystems involves crafting gate arrays and digital circuits tailored for each operation. However, as encryption demands grow due to big data, digital circuits and gate arrays face increasing latency and hardware costs. While analog circuits excel in mitigating these issues and executing tasks swiftly and efficiently, they struggle with GF(2) matrix operations. Analog circuits are adept at handling continuous variable problems and can perform matrix calculations in one-step by constructing and mapping mathematical structures. However, unlike digital computing's sequential logical operations, analog circuits cannot establish the necessary mathematical framework for GF(2) matrices, which prevents them from directly executing modulo-2 operations and completing key generation and encryption in QMM cryptosystem within a single matrix operation step. Aiming at the above problem,

Algorithm 1 Key generation and Encryption algorithm in QMM cryptosystem

Require: n,t,r,w,n_0: Parameters; m: Message; e: Error vector; (n= $n_0 \times r$)

Ensure: H:Private Key; G: Public Key; x: Cipher Text.

1: Generate a (n, r, w)-QC-MDPC code $h \in F_2^n$ of weight $w = \sum_{i=0}^{1} w_i$ at random.
2: $r-1$ quasi-cyclic shifting h and obtaining parity-check matrix H, where $H = [H_0|H_1|...|H_{n_0-1}]$.
3: Generate public key $G = [H_{n_0-1}^{-1}H_0|...|H_{n_0-1}^{-1}H_{n_0-2}|I_{n\times n}]$ and $G \in F_2^{(n-r)\times n}$.
4: Generate $e \in F_2$ of $wt(e) \leq t$ at random.
5: Calculate Cipher text $x = mG^T + e$.

we propose a data-flow deployment for GF(2) matrices operation in analog circuits, and ACIM circuits design which can efficiently generate key and ciphertext.

B. Related work

McEliece-type cryptosystems hardware: Current implementations of McEliece cryptography focus on general computing platforms (CPUs, GPUs) and FPGAs, including both classic and QC-MDPC versions. Some relative researches have aimed to reduce ciphertext length for faster encryption [7], [11] and optimize memory usage for better performance [13]. However, the used von-Neumann architecture poses a challenge, limiting the achievement of highly efficient encryption, especially for the requirement of high security levels and large data processing. FPGA implementations have improved speed and power consumption [1], [3], [4], but still face difficulties in maintaining low latency and minimal hardware overhead with increasing data sizes due to the limitations of digital circuits.

CIM architecture for PQC: CIM architecture facilitates in-situ computing within the storage unit, significantly reducing data transmission and hardware overhead, making it ideal for high-speed, efficient hardware-based PQC solutions. Ding et al. proposed an MRAM-based CIM accelerator for LPN-based PQC protocols, achieving speeds 216.8 times faster than CPU and FPGA [14]. Li et al. introduced a CIM-based crossbar design that significantly accelerates polynomial multipliers using the Number Theoretic Transform [15]. These works clearly demonstrate substantial acceleration for PQC. However, they primarily concentrate on digital computing and overlook the benefits of non-volatile devices in analog computing [9]. As computing dimensions and SL escalate, these designs may encounter challenges due to frequent signal conversions and excessive peripheral circuit redundancy.

In summary, the rapid growth of data necessitates a novel hardware design for PQC that can fulfill the increased SL and dimensional data encryption demands.

III. ACIM-QMM Design

In this section, we propose a data-flow deployment and ACIM circuits, which address two issues: (1) enabling analog circuits to directly perform GF(2) matrix operations, including key generation and encryption for QMM cryptosystem; (2) improving efficiency of circuits in latency and performance.

A. GF(2) Matrix Deployment Scheme

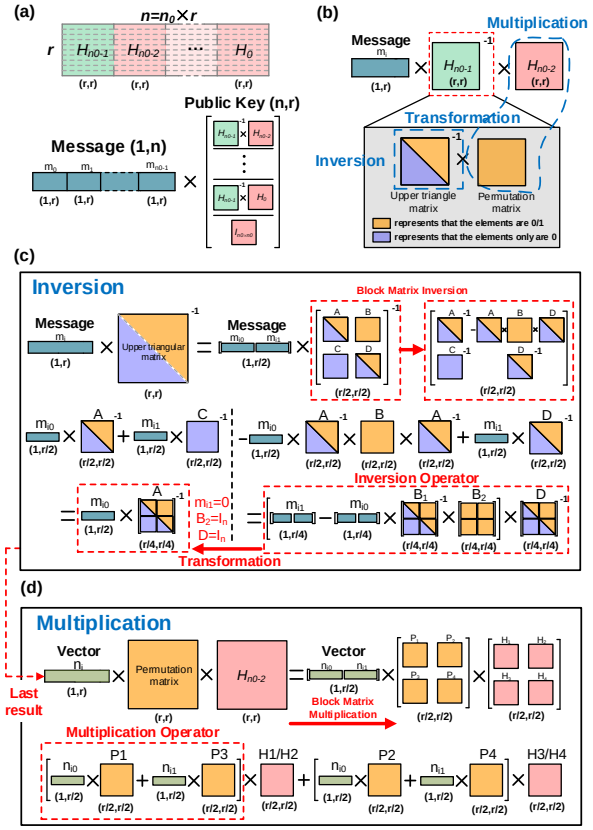


Fig. 2. (a) Parity-check matrix and transformation for H_{n_0-1}; Example of the matrices inversion (b) and multiplication (c) in key generation adopting the proposed scheme; (d) Procedure of cipher-text encryption.

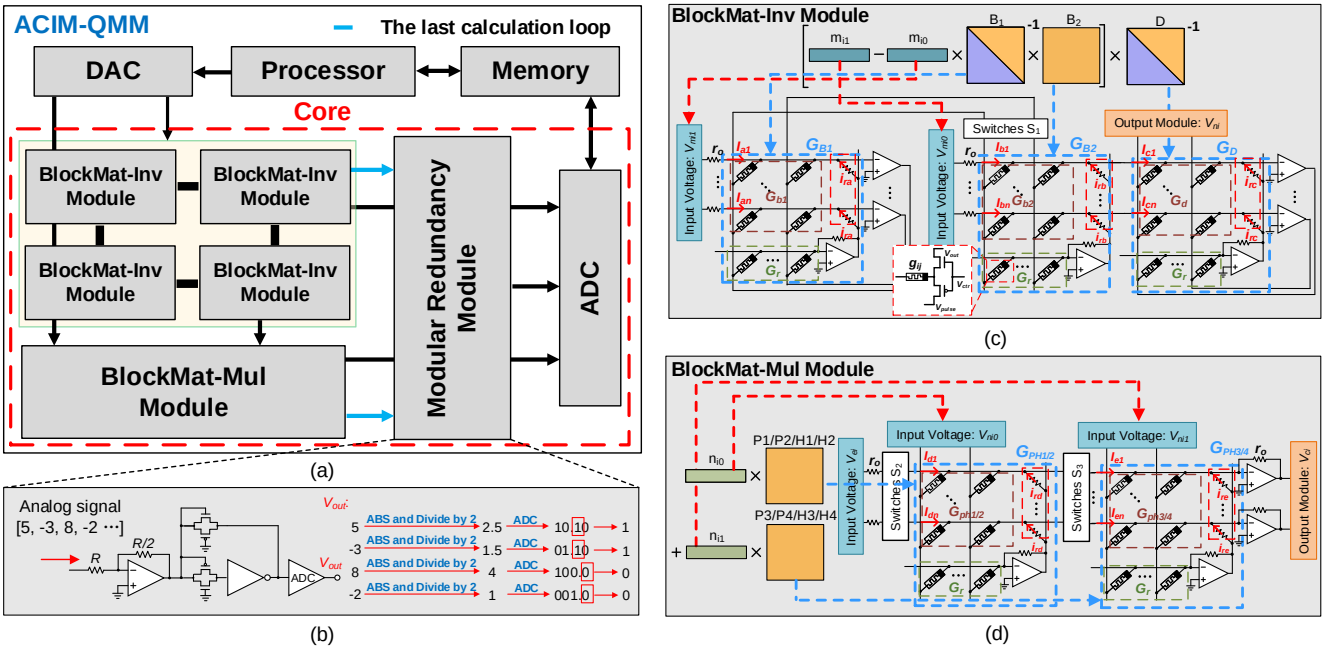


Fig. 3. (a) Overview of ACIM-QMM architecture; (b) Module for modular redundancy operation; (c) BlockMat-Inv circuit for IO in key generation; (d) BlockMat-Mul circuit for MO in key generation and ciphertext encryption.

The generated parity-check matrix H is divided into blocks, with H_{n_0-1} inverted as shown in Fig. 2(a). Algorithm 1 involves multiplying the message m by the public key matrix G built from H. In the process, m can be divided into n_0-1 vector m_i (where $i \in \{0, ..., n_0 - 1\}$), corresponding to the rows of the block matrices in G. Consequently, multiplying m by G can be broken down into a series of operations: multiplying n_0-2 vectors m_i by matrix $inv\,(H_{n_0-1})$ and H_i, and directly multiplying m_r by the identity matrix I_{n_0}, as shown in Fig. 2(a).

Given the inherent challenges of matrix inversion in analog circuits over GF(2), we propose a novel approach in the Fig. 2(b). We decompose H_{n_0-1} into an upper triangular matrix and a corresponding permutation matrix, both defined over GF(2). By inverting the upper triangular matrix and then multiplying it by the permutation matrix, we derive the inverse matrix modulo 2. By multiplying each message vector m_i with this inverse matrix and H_i, adding a randomly generated error e, and subsequently combining multiple block matrices, we can efficiently complete the entire key generation and encryption process.

For 80-bit to 256-bit SL encryption tasks, the dimension of H_{n_0-1} ranges from 3079 to 32771. However, limited ReRAM-based crossbars cannot directly handle such large matrix operations. Matrix multiplication can reduce complexity by segmenting matrices into blocks and sequentially performing block matrix multiplication and addition, allowing efficient execution on limited crossbars. However, matrix inversion does not lend itself to straightforward decomposition. To overcome this, we decompose the inversion matrix of the aforementioned upper triangular matrix and its corresponding permutation matrix. The inversion of the upper triangular matrix can be achieved by following these rules:

$$\begin{bmatrix} A & B \\ 0 & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}BD^{-1} \\ 0 & D^{-1} \end{bmatrix} \tag{1}$$

where block matrices A and B are the upper triangle matrices. Based on the rule, we can multiply the message vector m_i multiply with the inversion of upper triangular matrix, which is as demonstrated in Fig. 2(c) and has the following form:

$$m \begin{bmatrix} A & B \\ 0 & D \end{bmatrix}^{-1} = \begin{bmatrix} m_{i0}A^{-1} & (m_{i1} - m_{i0}A^{-1}B)D^{-1} \end{bmatrix} \tag{2}$$

where m_{i0} and m_{i1} are the sub-vector from vector m_i and the dimension relation is shown in Fig. 2(c). In the process of multiplying the inverse matrix of H_{n_0-1} with the message vector m_i the main operation involves $m_{i1} - m_{i0}A^{-1}BD^{-1}$, referred to as the Inversion Operator (IO) in this manuscript. By setting m_{i1} to 0 and matrices B and D to identity matrices, $m_{i0}A^{-1}$ can be calculated using this operator. According to block matrix rules, we can further decompose the operator and continuously reduce the scale of the upper triangular matrix to be compatible with limited-scale crossbars.

The above results following need to multiply permutation matrix P and matrix H_i. This process is illustrated in Fig. 2(c) and it follows:

$$\begin{aligned} nP &= \begin{bmatrix} n_0 & n_1 \end{bmatrix} \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \\ &= \begin{bmatrix} n_0P_1 + n_1P_3 & n_0P_2 + n_1P_4 \end{bmatrix} \end{aligned} \tag{3}$$

Through analysis, it is evident that there is a basic operator: $n_0P_1 + n_1P_3$, which can be adapted by replacing the permutation matrix P with matrix H_i to perform operations with H_i, and it is called as Multiplication Operator (MO) in this manuscript. Similarly, the block matrices used can be further decomposed to perform vector-matrices multiplication within them. Consequently, we can implement matrix inversion op-

erations in GF(2) on analog circuits. Based on the proposed operators, we can design domain-specific analog circuits that pave the way for executing key generation and encryption tasks of any SL QMM with limited hardware resources.

B. ACIM-QMM Circuits and Architecture

Using the aforementioned GF(2) matrix deployment scheme, we design analog circuits based on the operator and accelerator architecture shown in Fig. 3. The crossbar employed in our design adopts a 2T1R structure, and the programming method is based on the model described in [10].

The BlockMat-Inv circuit is proposed for mapping with IO, which is primarily composed of three 256×256 crossbar. Specifically, the messages m_{i1} and m_{i0} in IO are mapped to the input voltage source within the module, while the computed matrices B_1, B_2 and D are mapped to the crossbars, which is illustrated in Fig. 3(c). Since ReRAM cannot directly represent 0/1 elements in GF(2), we propose an equivalent matrix expression circuit comprising sub-crossbars and a common ReRAM row. For instance, $G_{B1}$is composed of a sub-crossbar G_{b1} and common ReRAM row G_r. Under the same voltage V, the following relation holds:

$$I = VG_{b1} - VG_r vec(1)^T \tag{4}$$

where I represents the current obtained by applying voltage V into the sub-crossbar G_{b1} in conjunction with the common ReRAM row G_r; $vec(1)$ denotes a vector of all ones. Utilizing operational amplifiers (OAs), the current resulting from voltage V applied to G_r can be redirected to flow back into the sub-crossbar G_{b1}. Consequently, we can derive the equivalent matrix $G_{B1} = G_{b1} - G_r vec(1)^T$. This equivalent matrix G transforms the high and low states of ReRAM into a 0/1 expression. Initially, when voltage V_{mi1} is applied into crossbar G_{B1}, the following principle applies:

$$\begin{cases} V_o = -\left(V_{mi1}/r_o\right) \cdot \left(G_{b1} - G_r\right)^{-1} \\ V_{mi1}/r_o = I_a + i_{ra} \\ I_a = V_o/G_{b1}, i_{ra} = -V_o/G_r \end{cases} \tag{5}$$

where G_{B1} is the equivalent matrix of the city constructed by G_r and G_{b1}. In there, the use of OAs and negative-feedback principle, the circuit converts the current into the voltage, and it is formally equivalent to taking the reciprocal of conductance G according to the principle of trans-impedance amplifiers (TIA). Based on the equivalent matrix, the sub-circuit achieve matrix inversion of upper triangle matrix B_1. After obtaining results V_o, under the control of switches, the remainder sub-circuits have the following principles:

$$\begin{cases} V_{ni} = \left[V_{mi1}/r_o + V_o\left(G_{b2} - G_r\right)\right] \cdot \left(G_d - G_r\right)^{-1} \\ V_{mi1}/r_o = I_b + i_{rb} + I_c + i_{rc} \\ I_b = V_o/G_{b2}, I_c = V_{ni}/G_d, i_{rb} = -V_o/G_r, i_{rc} = -V_{ni}/G_r \end{cases} \tag{6}$$

where $G_{b2} - G_r = G_{B2}$ and $G_d - G_r = G_{D2}$. The current flowing into matrix inversion circuit is obtained from voltage V_{mi0}, V_o and crossbar G_{B2}, and then the right sub-circuit converts the input current into voltage. Based on the above all analysis, the BlockMat-Inv circuit has the following relation:

$$V_{ni} = -\left[V_{mi1}/r_o - S \cdot \left[V_{mi0}/r_o \cdot G_{B1}^{-1}\right] \cdot G_{B2}\right] \cdot G_D^{-1} \tag{7}$$

where V_{ni} is the output, r_o is the resistor which can convert voltage into current, S is the control signal from Switches. The BlockMat-Inv circuit perform the IO operation when $S = 0$, otherwise it can map with the term $m_{i0}A^{-1}$.

The BlockMat-Mul circuit of Fig. 3(d) can achieve the MO operator, and the BlockMat-Mul circuit is composed of 512×512 ReRAM-based crossbar. The aforementioned result voltage V_{ni} is transferred to this circuit, it also divided into two vector n_{i0} and n_{i1}, which are respectively mapped with input voltage source. The block matrices of P and H in Fig. 2 can respectively map with the crossbar $G_{PH1/2}$ and $G_{PH3/4}$. Therefore, the principle is as follow:

$$V_{ci} = -r_o \cdot \left[S_2 \cdot \left(V_{ei}/r_o\right) + I_{ni}\right] \tag{8}$$

$$\begin{aligned} I_{ni} &= S_3 \cdot \left(V_{ni0}G_{PH1/2}\right) + V_{ni1}G_{PH3/4} \\ &= S_3 \cdot \left(I_d + i_{rd}\right) + I_e + i_{re} \end{aligned} \tag{9}$$

where S_2 and S_3 respectively represent the Switches. To be specific the above principle , it has the following form:

$$I_{ni} = S_3 \cdot \left(V_{ni0}G_{ph1/2} - V_{ni0}G_r\right) + V_{ni1}G_{ph3/4} - V_{ni1}G_r \tag{10}$$

The BlockMat-Mul circuit identically adopts equivalent matrices, it's kernel is to perform MVM operation. It will perform the operation that the multiplication of calculating results and permutation matrix P/block matrix H_i when $S_3 = 1$, the error vector also will be added if $S_2 = 1$ (which achieve map with the Process 9 of Algorithm 1), otherwise it will only perform multiplication of single block matrix.

The modular reduction operation can be the final step after completing all matrix operations, as noted in [16]. Moreover, we also propose a circuit design to perform modular reduction aiming at analog circuits, as shown in Fig. 3(b), which should be used after the other operation. The design consists of switches, OAs, inverters and analog to digital converter (ADC). The module has the following discipline:

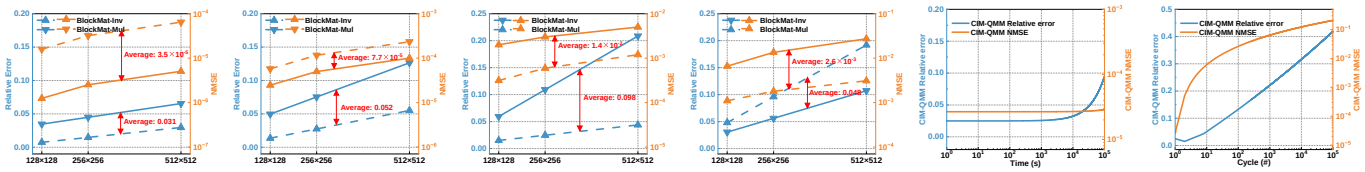
$$C\left(x, y\right) = \begin{cases} 1, x > y \\ 0, x \leqslant y \end{cases} \tag{11}$$

$$f\left(i\right) = C\left(\left|\frac{V_c}{2}\right|, \frac{V_r}{2^n}\left(2^{i+1} - 2\right)\right) + C\left(\left|\frac{V_c}{2}\right|, \frac{V_r}{2^n}\left(2^{i+1} - 1\right)\right) \tag{12}$$

$$V_{final} = \operatorname{sgn}\left(\sum_{i=1}^{n} f\left(i\right)\right) \tag{13}$$

where sgn represents sign function, V_r is the reference voltage of ADC, n is the bits number of ADC. When signal input, it becomes absolute value and divides into 2 by using switches and inverter. After 4-bit ADC conversion, the analog signal converts as digital signal. If 1 exists on the decimal place of the the modular reminder, the result should be 1, otherwise 0.

The above circuits together form the ACIM-QMM architecture, which is shown in Fig. 3(a). In this architecture, we use four BlockMat-Inv Modules for performing IO operation



(a) (b) (c) (d) (e) (f)

Fig. 4. Relative error and NMSE in the BlockMat-Inv and BlockMat-Mul circuits (a), under wire resistance impact (b), under device deviation impact (c), under noise impact (d), under device retention impact (e) and under device endurance impact (f).

and one BlockMat-Mul Module for performing MO operation. For any SL key generation and encryption task, we can continuously decompose the matrix until it fits the circuit scale of ACIM-QMM based on the above data-flow deployment method, and programming the decomposed block matrices into the crossbar sequentially. The results calculated by the modules of Fig. 3(c) and Fig. 3(d) are transferred into the Modular Redundancy Module, which performs the modulo-2 operation. The obtained results are the generated ciphertext.

IV. Results and Analysis

ACIM-QMM architecture is simulated for its performance in this section. The circuits are designed and tested in TSMC 180nm and 65nm CMOS library in the Cadence tool. All parameters of the ReRAM cells are set to the values in [17].

A. Evaluation for Calculation

Since ReRAM is easy to be influenced by some non-ideal factors, we test the proposed circuits under various non-ideal factors, as shown in Fig. 4. Fig. 4(a) shows the relative error and normalized mean square root (NMSE) of it, which do not consider non-ideal factor impact, Fig. 4 respectively considers 2.5Ω wire resistance, 2% device deviation and 5% signal noise interference. The above data demonstrates it has good accuracy and minor difference comparing with ideal results.

However, it's worth noting that the relative error from analog circuits may adversely affect ACIM-QMM's performance in key generation and encryption. Accumulated errors can lead to incorrect modular operation results. While small errors in QMM cryptosystem can be tolerated and corrected by the decoding algorithm using the parity-check matrix, excessive errors can severely compromise ciphertext generation. In evaluation, the small NMSE indicates minimal deviation from ideal results. To mitigate error impact, we propose a compensation scheme tailored for QMM cryptosystem. This scheme consists of two parts: adjusting the resistor R in the negative terminal of the OA in Fig. 3(b), and adding a compensating voltage in the Modular Redundancy module. Using Fig. 4(a) as an example, we set R as following:

$$R_{comp} = \frac{R}{1 - RE} \tag{14}$$

where RE is the average value of BlockMat-Inv module's and BlockMat-Mul module's the relative error. Moreover, it can be performed that add complementary voltage in the BlockMat-Mul module, the complementary voltage set as follow:

$$V_{comp} = \frac{\lfloor \lfloor n/\omega \rfloor / r \rfloor}{a} \tag{15}$$

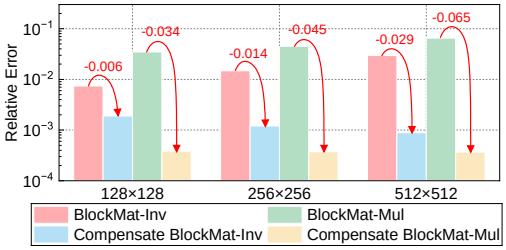
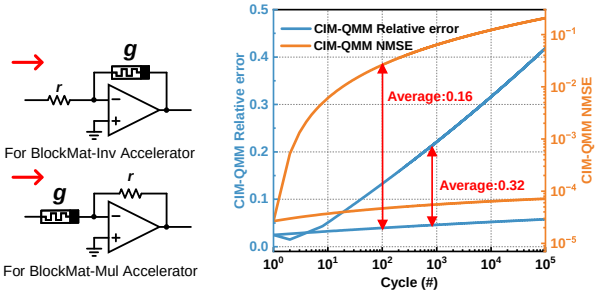


Fig. 5. Compensated accuracy for the proposed modules

where a is the seting parameter; n, r and w are the same as Algorithm 1. Therefore, as shown in Fig.5, the relative error has been reduced to below 10^{-3} and do not impact the results.

In the evaluation of endurance and retention, it can be found that the accelerator under retention has less impact in Fig. 4(e) and it does not influence the final results, but the accelerator under endurance can not be utilized due to large relative error and NMSE in Fig. 4(f). Aiming at the problem, we propose a circuit-level scheme to alleviate the impact, the compensating circuits of Fig. 6(a) is used in the input ports, it can make the input voltage having the same change relation with the conductance of ReRAM under different cycles. Therefore, the input voltage and conductance of ReRAM all obey the identical attenuation curve, which can minimise the relative error and NMSE, as shown in Fig. 6(b).



(a) (b)

Fig. 6. (a) Compensating circuits for ACIM-QMM; (b) Results.

B. Evaluation for Performance

In this section, we evaluate ACIM-QMM in the area and power consumption, and compare it with other PQC accelerator. Firstly, the area and power consumption are evaluated in Fig. 7, it shows that OAs has the most occupation in power, and DAC and OAs need more area overhead in the design.

Moreover, we compare the performance of ACIM-QMM, CPU and GPU platforms over the standard QMM cryptosystem. The CPU experiments were carried out on an Intel

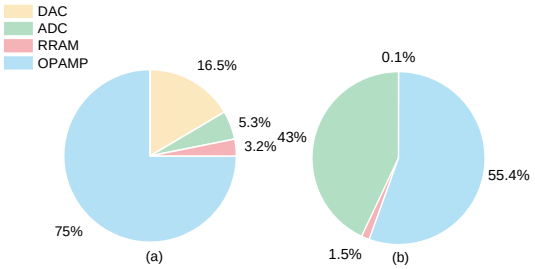


Fig. 7. (a) Power consumption and (b) area consumption of the ACIM-QMM.

TABLE I
PERFORMANCE COMPARISON OF ACIM-QMM WITH OTHER PQC HARDWARE.

Scheme	SL[bit]	Time(ms)		Area (mm^2)	Energy (μJ)
		Gen	Enc./Sig.		
ACIM-QMM	80	0.036		0.42	18.4
	128	0.134			68.6
	256	1.290			660.5
Kyber [18]	128	3.32	4.27	0.67	140.9
	256	7.73	8.96		307.7
Frodo-KEM [19]	128	159	161	0.28	2177.6
	256	994	99		13425.3
Dilithium [20]	128	3.84		1.31	26.6
	256	10.2			82.5

Xeon(R) Platinum 8352V processor. The GPU evaluation is based on the NVIDIA A800-80GB and V100-32GB. All parameters of the tasks are instantiated referencing [7]. Fig. 8 shows that it has great improvement in speed, especially it can achieve maximum 3E6× improvement in 256-bit SL comparing with GPU A800.

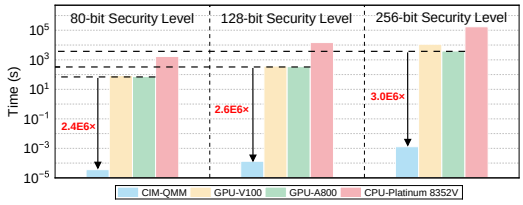


Fig. 8. Comparison with GPU and CPU in executing time.

Table I compares ACIM-QMM with other PQC cryptography hardware, including Kyber, Frodo-KEM, and Dilithium, which offer encryption and signature capabilities. Under a 256-bit SL evaluation, ACIM-QMM achieves a minimum of 7.9× and a maximum of 847.3× improvement in key generation and encryption/signature time. In terms of energy, it offers up to 31.7× and 20.3× improvement for 128-bit and 256-bit SLs, respectively. This significant improvement is attributed to ACIM-QMM's use of ACIM, which performs matrix operations in one-step, drastically lowering power and area overhead. Therefore, we believe that ACIM-QMM can offer a solution for advancing PQC hardware design.

V. CONCLUSION

In summary, we propose an efficient and robust ReRAM-based CIM circuit design for accelerating the QMM cryptosystem, which can rapidly perform key generation and encryption. To address the challenge of analog circuits' inability to perform GF(2) matrix operations, we propose a dataflow deployment method. This method not only resolves the aforementioned issues, but also enables the efficient execution of key generation and encryption for any SL on analog circuits with limited scale. Our experimental results demonstrate that the innovative design can achieve maximum 288.1× speedup compared to SOTA, and maximum 31.7× energy improvement compared to other PQC hardware in 128-bit of security.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No. 62234008, No. 62371186, No.62471177), Key R&D Program of Hunan Province (No. 2024JK2009), Natural Science Foundation of Hunan Province (No. 2022JJ30160, No.2023JJ30168), Special Funds for the Construction of Innovative Provinces in Hunan Province (No. 2023RC3103, No. 2022SK2007, No. 2023GK2062, No. 2023GK2071), Scientific Research Project of Education Department of Hunan Province (No. 21A0012), Changsha Science and Technology Project (No. KH2202001), Huxiang Young Talents (No. 2023RC3103), Hunan Postgraduate Scientific Research Innovation Project (No. CX20240421).

REFERENCES

[1] E. Cantó-Navarro et al. AXI Hardware Accelerator for McEliece on FPGA Embedded Systems. *IEEE TDSC*, 2024.
[2] J. Richter-Brockmann et al. Folding BIKE: Scalable Hardware Implementation for Reconfigurable Devices. *IEEE TC*, 2022.
[3] Y. Zhu et al. Mckeycutter: A high-throughput Key Generator of Classic McEliece on Hardware. in *60th ACM/IEEE DAC*, 2023.
[4] P. Chen et al. Complete and Improved FPGA Implementation of Classic McEliece. *IACR TCHES*, 2022.
[5] I. von Maurich et al. Lightweight Code-based Cryptography: QC-MDPC McEliece Encryption on Reconfigurable Devices. In *DATE*, 2014.
[6] V. Kostalabros et al. HLS-Based HW/SW Co-Design of the Post-Quantum Classic McEliece Cryptosystem In *FPL*, 2021.
[7] R. Misoczki et al. MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes. In *IEEE ISIT*, 2013.
[8] R. Agrawal et al. Quantum-Proof Lightweight McEliece Cryptosystem Co-processor Design. in *38th IEEE ICCD*, 2020.
[9] S. Wang et al. Smaller, Faster, Lower-power Analog RRAM Matrix Computing Circuits without Performance Compromise. *SCIS*, 2024.
[10] Q. Hong et al. Memristive Circuit Implementation of Caenorhabditis Elegans Mechanism for Neuromorphic Computing. *IEEE TNNLS*, 2023.
[11] R. Hooshmand et al. Reducing the Key Length of McEliece Cryptosystem using Polar Codes. in *11th ISCISC*, 2014.
[12] Y. Li et al. Security Analysis of the Classic McEliece, HQC and BIKE Schemes in Low Memory. *JISA*, 2023.
[13] B. Colombier et al. Profiled Side-channel Attack on Cryptosystems based on the Binary Syndrome Decoding Problem. *IEEE TIFS*, 2023.
[14] L. Ding et al. PIMA-LPN: Processing-in-memory Acceleration for Efficient LPN-based Post-Quantum Cryptography. in *60th DAC*, 2023.
[15] H. Nejatollahi et al. CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware. in *57th DAC*, 2020.
[16] William Easttom. *Modern Cryptography: Applied Mathematics for Encryption and Information Security*. Springer, 2022.
[17] Y. Lu et al. An Electronic Silicon-based Memristor with a High Switching Uniformity. *Nature Electronics*, 2019.
[18] F. Tim et al. RISQ-V: Tightly Coupled RISQ-V Accelerators for Post-Quantum Cryptography. *IACR TCHES*, 2020.
[19] B. Utsav et al. Sapphire: A configurable Crypto-processor for Post-quantum Lattice-based Protocols. *IACR TCHES*, 2019.
[20] P. Karl et al. Post-Quantum Signatures on RISC-V with Hardware Acceleration. *ACM TECS*, 2024.

ReMaP: Macro Placement by Recursively Prototyping and Periphery-Guided Relocating

Yunqi Shi[1,2,3*], Xi Lin[1,2*], Siyuan Xu[3], Shixiong Kai[3], Ke Xue[1,2], Mingxuan Yuan[3], Chao Qian[1,2], Zhi-Hua Zhou[1,2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
[3]Huawei Noah's Ark Lab, China

***Abstract*—We introduce the ReMaP framework, which generates expert-quality macro placements through recursively prototyping and periphery-guided relocating. A key innovation is ABPlace, an angle-based analytical method that arranges macros along an ellipse to facilitate a rough distribution near the periphery, while optimizing dataflow, minimizing overlap, and ensuring convergence. Based on the results of ABPlace, an efficient heuristic is proposed to position macros along the chip's periphery, mirroring practices often employed by experts. Our framework outperforms three leading macro placers in both WNS and TNS across eight test cases, achieving improvements up to 34.15% in WNS and 65.39% in TNS, as tested on the popular OpenROAD-flow-scripts infrastructure. Additionally, our parameter auto-tuning method further improves timing by 8.75%.**

I. Introduction

The macro placement problem has been a significant focus in both academia and industry for decades. Ineffective macro placement can obstruct essential core areas of a chip, leading to overcrowded and elongated paths that delay timing and complicate the positioning of standard cells. This issue has become increasingly important as chips grow in size and complexity, requiring the integration of more macros, which makes achieving efficient placement more challenging.

Due to its critical role, macro placement has traditionally required experts who use specialized knowledge for each specific case. This includes optimizing dataflow to enhance timing, placing macros along chip periphery to reduce routing congestion, tiling macros to improve power management, and positioning macros away from pin areas to facilitate pin access [9], [10]. Experts also use feedback from back-end evaluations and front-end engineers to refine the results iteratively, which can take weeks or months to complete [19].

To achieve expert-quality placement results and reduce turn-around time (TAT), automated placement algorithms emulate expert behaviors and incorporate heuristics into their optimization processes. Our work focuses on positioning macros along the chip periphery, and incorporating specific strategies to enhance placement regularity. It's important to distinguish between periphery placement and regularity, as they serve different purposes. Some approaches address them separately [9], [14], while others do not [22], [26]. Periphery placement helps preserve central regions for standard cell placement, reducing routing congestion and detours, which enhances overall timing performance. Improving regularity involves tiling macros of the same size and minimizing notch and dead areas [9], [14], which not only benefits power planning but also helps placement and routing for large-scale, high-utilization cases.

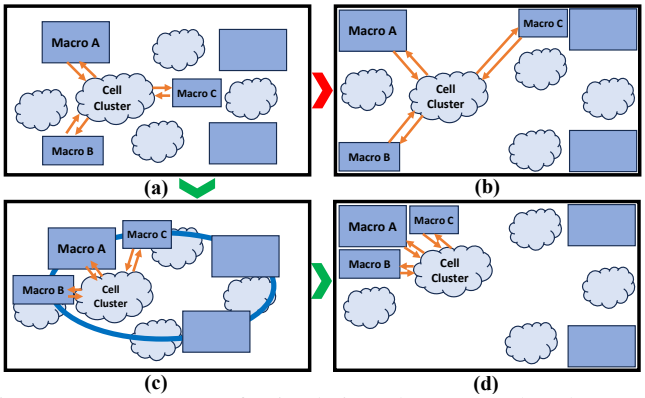


Fig. 1: (a) Prototype of mixed-size placement. (b) Placement result following the direct application of the periphery-pushing heuristic. (c) Optimization of macros on an ellipse, called ABPlace. (d) Periphery-guided relocating of macros based on their placement along the ellipse.

The general motivation and strategy of our flow is shown in Fig. 1. We begin with a mixed-size prototyping as shown in Fig. 1(a). An intuitive yet less effective method, shown in Fig. 1(b), involves moving each macro towards the nearest corner or periphery of the chip layout. No matter where we place the cell cluster, the total communication cost remains high due to the considerable distances between macros A, B, and C. Alternatively, to relocate macros to the periphery, we fix all cell clusters and optimize macro positions along an ellipse, considering dataflow cost and macro overlap, as shown in Fig. 1(c). The macros are then strategically moved to optimal locations considering dataflow and periphery cost, as illustrated in Fig. 1(d). This sequence of steps (a), (c), and (d) is repeated recursively. The number of macros placed at each iteration is tuned to balance efficiency and accuracy.

Our contributions are summarized as follows:

- We develop a macro placement framework, **ReMaP**, that delivers expert-quality results through a recursive process of prototyping and periphery-guided relocating. Our algorithm naturally handles pre-placed macro scenarios by iteratively placing and fixing macros in stages. We have open-sourced the code at https://github.com/lamda-bbo/DAC25-ReMaP and provided detailed evaluation scripts and metadata to facilitate replication.
- We introduce ABPlace, an innovative angle-based ana-

* Equal contribution.

lytical method that arranges and optimizes macros along an ellipse. This approach considers the dataflow strength between macros and macro-cell clusters, and helps distribute macros uniformly near the chip periphery. With a single angle variable to optimize for each macro, ABPlace achieves easy convergence across all test cases.
- We design an efficient heuristic to relocate macros to the chip periphery, guided by the results from ABPlace. We propose two criteria to determine the placement order: one emphasizes dataflow, while the other enhances layout regularity. By dividing the feasible placement region into grids, we then optimize dataflow through greedy selection.
- Our extensive comparisons with three leading macro placement tools—RTL-MP [9], Hier-RTLMP [10], and DREAMPlace 4.1.0 with macro placement enhancements [2], [18]—demonstrate that the proposed ReMaP achieves superior post-route WNS and TNS across all eight chip cases from the OpenROAD-flow-scripts [1]. We also involve Bayesian optimization for parameter autotuning and performance improvement.

II. Related Works

Generally, macro placement techniques can be classified into three categories: zeroth-order methods, analytical methods, and learning-based methods.

Zeroth-order methods are extensively utilized in the field of design automation, as most of the power, performance and area (PPA) metrics are black-box and non-differentiable. There are two fundamental components for a zeroth-order method for macro placement: solution representation and evaluation. Initial research efforts mainly focus on proposing effective data structures to represent a placement solution, including corner block list [7], sequence pair [20], B*-tree [25], slicing tree [27], and so on. The evaluation criteria mainly include wirelength and total area. Then zeroth-order optimization algorithms like simulated annealing or evolutionary algorithms are applied to optimize the solution. Recently, dataflow has emerged as a prominent optimization target [24], offering insights into RTL-level information. It aims to reveal the underlying connection relationships between macros [28]. Specifically, RTL-MP [9] and Hier-RTLMP [10] incorporate multilevel clustering through dataflow extraction to enhance placement regularity and performance. These methods employ a weighted sum of seven distinct targets as the optimization objective, effectively integrating engineering know-how. Another representative work by Lin et al. [15] uses corner stitching [21] to represent the solution, effectively addressing the challenges associated with pre-placed blocks and enhancing overall efficiency.

Analytical methods [2], [3], [17], [18] try to distribute macros and cells simultaneously by optimizing the wirelength and placement density. Post-processing steps such as macro adjustment or legalization are then applied to further enhance the placement. These methods have the merit of efficiency and the global view of wirelength. Recently, DG-RePlAce [11] and DAPA [16] try to involve dataflow as a constraint into the global placement, trying to capture the critical datapaths and optimize them accordingly. Another recent work IncreMacro [22] identifies macros placed away from the periphery and incorporates smoothed cost as a nonlinear optimization metric.

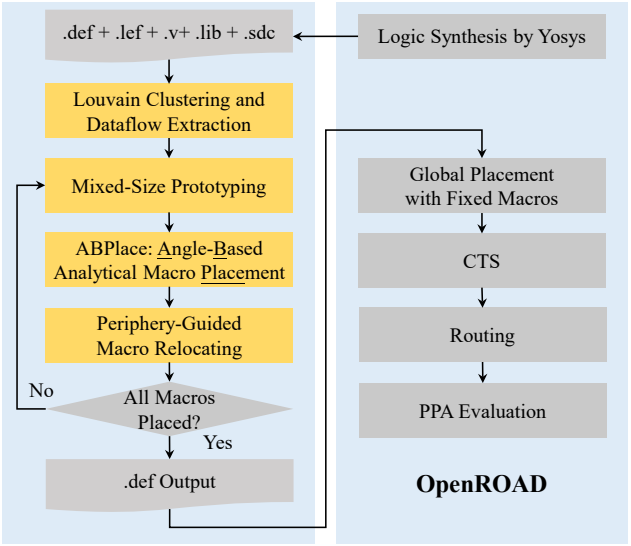


Fig. 2: Overflow of our proposed ReMaP method. Components in yellow are key steps of ReMaP.

Learning-based methods have recently gained attention, starting with Google's Graph Placement [19]. These methods employ reinforcement learning for macro placement, yet face challenges with reward shaping. Providing rewards only after all macros are placed [4], [19] leads to sparse feedback, while using incremental wirelength as a dense reward [6], [12], [13], [23] often misaligns with PPA metrics.

Despite advancements, the macro placement challenge remains largely unresolved. Traditional zeroth-order methods may fail to capture the locations of standard cells after global placement. Recent approaches like RTL-MP [9] and Hier-RTLMP [10] attempt to address this by involving dataflow and using a sequence pair to simultaneously represent macros and cell clusters. However, the positions of cell clusters determined by simulated annealing may not align with those from an actual global placement. Analytical methods are also widely adopted, but they face convergence issues for mixed-size designs. Moreover, further refining and legalizing macros typically involve heuristics [2], [22], which can lead to inevitable performance degradation when all standard cells are treated as fixed [22].

III. Proposed ReMap Framework

In this section, we introduce ReMaP, a novel macro placement framework that generates expert-quality macro placement layouts. As shown in Fig. 2, our process begins with a synthesized netlist, from which we perform clustering and extract dataflow connections among macros and cell clusters. We employ DREAMPlace 4.1.0 [2], [18] to establish a mixed-size placement prototype, setting initial positions for each macro and cell cluster. It is followed by ABPlace, an angle-based analytical method that optimizes macro positions on a predefined ellipse (or a circle for square chip canvas). We then relocate macros to optimize the dataflow and periphery cost on a discretized canvas. In each iteration, we process a subset

of macros, fix their positions, and cycle the remaining macros back into the loop for next iteration. The final PPA evaluation is performed using the full OpenROAD-flow-scripts [1].

A. Clustering and Dataflow Extraction

To reduce complexity and uncover connections among macros and cells, we begin our placement process with clustering. We use star decomposition to break down nets into individual edges, forming an undirected graph. The modularity-based Louvain algorithm [5] then identifies communities within this graph, while also considering pre-placed I/O pins. For precise macro placement, each macro is extracted and treated as its own cluster. After forming all clusters, we traverse the initial graph to calculate the **direct** dataflow strength between each cluster pair, acquiring the dataflow matrix A. The process above is only performed once during the entire placement process.

Although several hierarchical clustering and dataflow extraction methods have been proposed [8]–[10], we opt for the straightforward Louvain algorithm and focus on direct connections for dataflow analysis, which have already yielded superior results. Our ReMaP framework can be integrated with any advanced clustering method for enhanced performance.

B. Mixed-Size Prototyping

Incorporating cell locations into macro placement is crucial, as these combinational components occupy significant space and connect macros. Traditional macro placers often model cell clusters as soft blocks and optimize them alongside macros. However, determining the area and shape of these clusters is challenging, and the final cell positions may differ after analytical placement. Direct mixed-size global placement can also face divergence issues, leading to suboptimal results.

To accurately estimate cell cluster positions and avoid divergence, we adopt a self-adaptive analytical mixed-size placement approach. Initially, a high target density facilitates convergence when many macros are movable. As more macros become fixed, the target density decreases to align with the low-density design's final cell placement[1]. Using this mixed-size prototype, we can determine the locations of macros and cell clusters by averaging the positions of all cells within each cluster.

C. ABPlace: Angle-Based Analytical Macro Placement

After acquiring the positions of macros and cell clusters, we fix the cell clusters and map each macro to the boundary of an ellipse. Let the width and height of the chip core be W and H. The center of the ellipse aligns with the chip's center, both at the origin. The ellipse's semi-major a and semi-minor b axes are scaled by β and shrink with each iteration by γ. Thus, the ellipse's equation at the k-th iteration is:

$$\begin{cases} a = \beta\gamma^k W/2,\ b = \beta\gamma^k H/2 \\ x^2/a^2 + y^2/b^2 = 1 \end{cases}$$

where x and y describe the coordinates of points on the ellipse. When $W = H$, the ellipse becomes a circle. To map macros onto the ellipse, we draw a ray from the ellipse center to the macro center. The intersection point with the ellipse is the macro's new position. Given the original coordinates (x_i^0, y_i^0) of macro i, the mapped position (x_i, y_i) is calculated by:

$$\theta_i = \arctan\frac{y_i^0}{x_i^0},\ x_i = a\cos\theta_i,\ y_i = b\sin\theta_i,$$

where $\theta_i \in [0, 2\pi)$ represents the angle of macro i.

After mapping, we fix cell clusters and optimize macro positions using angle-based analytical optimization. With macros restricted to the ellipse, we only optimize the angle vector $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \left(\sum_{i\in M}\sum_{j\in N} \text{dist}(i,j)A_{i,j} + \lambda \sum_{i,j\in M, i\neq j} \text{overlap}(i,j) \right), \tag{1}$$

where A is the dataflow matrix indicating connection strength, M is the set of all unplaced macros, and N includes all macros and cell clusters. We use Euclidean distance for dist function. The overlap penalty between macros i and j is defined as:

$$\begin{aligned} \text{overlap}(i,j) = &\max\{(w_i + w_j)/2 - |x_i - x_j|, 0\} \\ &* \max\{(h_i + h_j)/2 - |y_i - y_j|, 0\}, \end{aligned}$$

where w_i and h_i are the width and height of macro i, respectively. The $\max$ function has discontinuities at zero, which can complicate gradient calculations. We use `torch.clamp` to handle the boundary conditions. Additionally, we approximate the absolute value using a smoothing function $\sqrt{x^2 + \epsilon^2} - \epsilon$ to maintain differentiability throughout the optimization process.

Considering our ultimate goal of placing macros regularly along the chip's periphery while minimizing data flow loss, ABPlace offers two key advantages by incorporating an ellipse: it positions macros close to their final periphery locations in a uniform manner while considering dataflow cost, and it allows for easy convergence without extensive parameter tuning.

D. Periphery-Guided Macro Relocating

After optimizing macro positions on the ellipse, we relocate certain macros to the chip's periphery, aiming to minimize dataflow loss. The problem is formulated as follows: given a chip layout with several placed macros, we relocate a set of macros previously positioned on the ellipse boundary, one by one. At each placement step, we solve two key problems: selecting which macro to place and determining its location.

We propose two criteria for selecting macros to place:

- **Descending order of macro's overlapping area**: Larger macros in crowded regions on the ellipse are prioritized for placement. This approach enhances regularity in two ways. First, placing larger macros near the periphery and smaller ones towards the core improves the utilization of outer space. Second, macros with similar shapes and sizes tend to be sorted and placed together.
- **Ascending order of dataflow strength**: Macros with weaker dataflow connections are placed first, allowing macros with stronger connections to be positioned closer to the core. This strategy reduces the negative impact on overall dataflow from periphery placement. Additionally,

[1]This dynamic approach is designed for OpenROAD-flow-scripts benchmarks [1] with low utilization and target density, as shown in Table I. We will extend ReMaP to larger, high-density designs in the future.

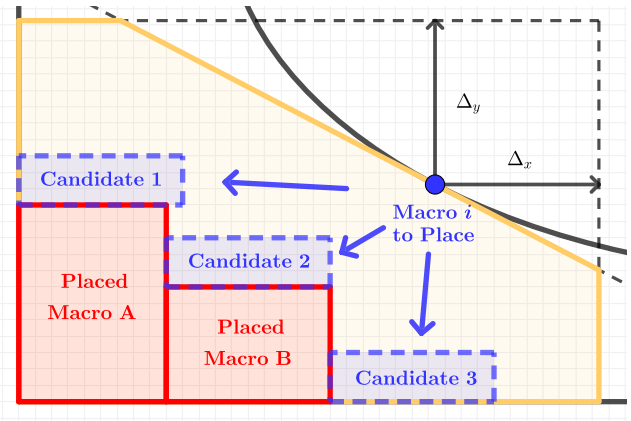


Fig. 3: Illustration of periphery-guided macro relocating: With macros A and B already placed in previous iterations, macro i is optimized on the ellipse by ABPlace in the current iteration. It will be placed in one of three candidate positions.

macros of the same size and group often share similar connection strengths, so they tend to be placed together, further enhancing placement regularity.

We choose the second criterion as our default setting for potentially better timing performance. The first criterion is also tested in ablation studies IV-B, producing layouts with higher regularity and occasionally better timing in specific cases. Overall, the second criterion generally performs better.

Next, we determine macro positions along the chip periphery inside a discretized feasible region. As illustrated in Fig. 3, we draw a tangent to the ellipse at macro i. Then we draw lines parallel to the x and y axes at distances Δ_x and Δ_y from the macro center. This forms a pentagon, shaded in yellow in Fig. 3, representing the placable region for macro i. We discretize this region into grids, and traverse feasible grids (x, y) that avoid overlaps, minimizing two cost functions c_{peri} and c_{df}:

$$c_{\text{peri}} = \min\{W - x, x\} + \min\{H - y, y\},$$
$$c_{\text{df}} = \sum\nolimits_{j \in P} \text{dist}(i, j) A_{i,j},$$

where c_{peri} and c_{df} represent the periphery cost and dataflow cost, respectively, and P includes placed macros and all cell clusters. We first consider c_{peri} using a Pareto dominance approach. Grid (x, y) dominates (x', y') if and only if:

$$\begin{cases} \min\{W - x, x\} \leq \min\{W - x', x'\} \\ \min\{H - y, y\} \leq \min\{H - y', y'\} \end{cases}$$

We select grids that are not dominated by others as candidates. Such selection prefers locations that are close to the chip corner. As shown in Fig. 3, only three candidate positions remain. We then compute c_{df} for each candidate grid, and choose the grid with the optimal (minimal) c_{df} value.

E. Summary and Implementation Details

The proposed ReMaP is presented in Algorithm 1. Lines 1–3 prepare the initial data for placement optimization. We extract only direct connections as the dataflow matrix, as discussed in Sec. III-A. While we attempted to incorporate indirect connections, like one-hop and two-hop links [10], [28], the improvement was negligible and it required further parameter tuning for these weaker connections. With accurate standard cell positions integrated into our optimization, direct connections suffice in our practice.

Algorithm 1 ReMaP Framework

Input: .def, .lef, .v, .lib, .sdc, decay factor γ, scale factor β, number k of macros to place per iteration, number n of macros.
Output: .def with all macros placed.
Process:

1: Cluster all macros and cells using the Louvain algorithm.
2: Extract dataflow matrix A.
3: Separate macros from clusters, treating each as an individual cluster.
4: $iter \leftarrow 0$
5: **while** there remain unplaced macros **do**
6: Perform a preliminary mixed-size placement and compute the centroid of each cell cluster.
7: $a \leftarrow \beta\gamma^{iter}W/2$, $b \leftarrow \beta\gamma^{iter}H/2$
8: Map each unplaced macro onto the ellipse defined by a and b.
9: Execute ABPlace for angle-based analytical placement.
10: Rank macros by specified criteria and select the top k.
11: **for** each macro in the top k **do**
12: Select the optimal grid considering dataflow and periphery cost.
13: Place and fix the macro.
14: **end for**
15: $iter \leftarrow iter + 1$
16: **end while**

Lines 5–16 iteratively optimize macro placement. In each iteration, k macros are placed and fixed. The parameter k balances efficiency and accuracy; setting k to 1 maximizes accuracy but increases runtime, especially for circuit with numerous macros. We typically set k to $\lceil n/10 \rceil$ to ensure that the algorithm completes within 10 iterations. The ellipse size is updated in each iteration. Initially, it is large to reach the periphery of the layout. As more macros are placed, the ellipse size reduces to fit the available space. We set the scale factor β to 0.9 and keep it fixed. The decay factor γ, however, is adjusted based on specific cases. During ABPlace in line 9, the Lagrange multiplier λ in Eq. (1) is crucial; high values ensure uniform distribution, while low values emphasize dataflow. We manually choose and fix λ throughout placement.

Our algorithm primarily requires tuning γ and λ. Our main results in Table I are slightly tuned manually. We further employ Bayesian optimization for automatic tuning and promote the performance, as detailed in Sec. IV-D.

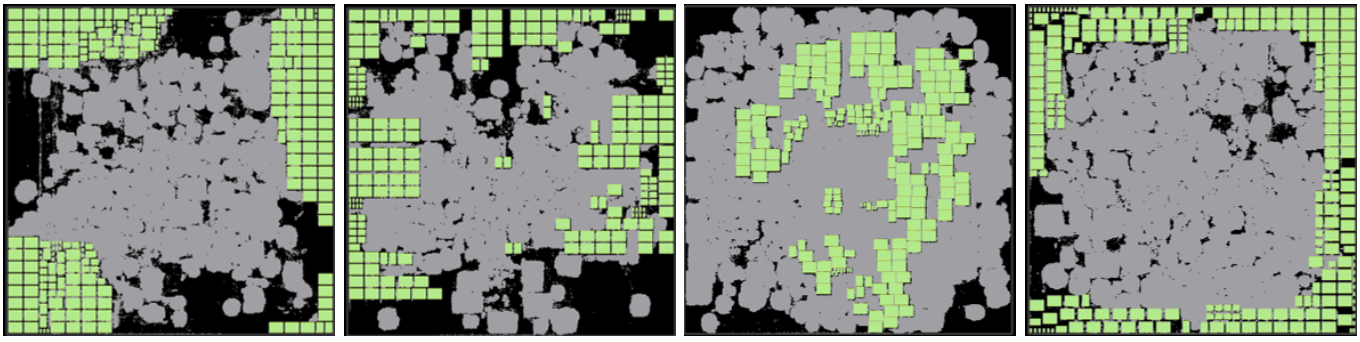
IV. Experimental Results

We implement ReMaP in Python and evaluate it on a Linux server with a 52-core Intel Xeon CPU (2.60 GHz), an NVIDIA RTX 2080S GPU, and 128GB RAM. We use eight RTL designs from the OpenROAD-flow-scripts[2] (ORFS) repository, synthesized with the Nangate45 library. For timing

[2]https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts

TABLE I: Post-route PPA results on eight designs. Best results of WNS and TNS are in **bold**. Freq. and Util. represent the frequency and area utilization ratio of the design, respectively.

Benchmark	Macros (#)	Cells (#)	Nets (#)	Freq. (MHz)	Util.	Methods	rWL (m)	WNS (ns)	TNS (ns)	Power (W)	Overflow (#)	Runtime (s)
ariane133	132	168K	204K	250.0	0.35	*RTL-MP*	6.76	-1.11	-2839.00	0.314	2055	1753
						Hier-RTLMP	7.07	-1.17	-2967.29	0.316	0	572
						DREAMPlace	7.10	-1.33	-3628.90	0.325	23	36
						ReMaP (ours)	7.48	**-1.07**	**-2615.74**	0.318	24	478
ariane136	136	171K	209K	333.3	0.36	*RTL-MP*	8.16	-2.07	-6315.44	0.479	332	4859
						Hier-RTLMP	7.91	-2.23	-6830.66	0.479	100	153
						DREAMPlace	7.88	-2.21	-6700.25	0.483	78	47
						ReMaP (ours)	7.81	**-2.04**	**-6085.84**	0.464	7	589
black_parrot	24	307K	359K	500.0	0.45	*RTL-MP*	9.72	-4.98	-1102.72	0.475	6281	18
						Hier-RTLMP	10.84	-4.75	-2111.50	0.463	75323	124
						DREAMPlace	10.52	-5.15	-1216.80	0.457	6922	45
						ReMaP (ours)	9.34	**-4.67**	**-1090.18**	0.454	4548	322
bp_be	10	51K	66K	1250.0	0.45	*RTL-MP*	3.36	-0.98	-125.01	0.134	25978	2
						Hier-RTLMP	3.35	-0.96	-105.89	0.135	80552	17
						DREAMPlace	3.80	-0.92	-111.73	0.139	23007	24
						ReMaP (ours)	3.89	**-0.90**	**-98.00**	0.136	31288	389
bp_fe	11	33K	43K	555.6	0.47	*RTL-MP*	2.16	-0.41	-19.88	0.156	8841	2
						Hier-RTLMP	2.17	-0.45	-23.68	0.155	34839	17
						DREAMPlace	2.58	-0.65	-43.28	0.162	28885	18
						ReMaP (ours)	2.65	**-0.27**	**-6.88**	0.160	15936	401
bp_multi	26	152K	179K	1250.0	0.47	*RTL-MP*	5.17	-5.84	-6981.22	0.609	237	20
						Hier-RTLMP	5.24	-6.18	-6677.56	0.615	2539	72
						DREAMPlace	5.99	-5.95	-6735.01	0.625	29477	26
						ReMaP (ours)	4.98	**-5.82**	**-6190.78**	0.614	160	491
bp_quad	220	1135K	1448K	384.6	0.39	*RTL-MP*	55.35	-1.40	-22767.70	1.803	161655	10918
						Hier-RTLMP	44.68	-0.99	-16445.80	1.805	11330	1270
						DREAMPlace	51.92	-1.31	-21669.80	1.824	28822	120
						ReMaP (ours)	54.30	**-0.93**	**-14139.20**	1.812	4629	1785
swerv_wrapper	28	98K	117K	500.0	0.61	*RTL-MP*	4.46	-1.36	-1318.75	0.256	15624	17
						Hier-RTLMP	5.25	-1.79	-1705.52	0.261	62726	38
						DREAMPlace	5.33	-1.41	-1364.19	0.262	47836	22
						ReMaP (ours)	4.44	**-1.12**	**-979.03**	0.256	20072	459
Average Rank of *ReMaP* (ours)							2.375	1	1	2.125	1.75	3.5



(a) *RTL-MP* (b) *Hier-RTLMP* (c) *DREAMPlace* (d) *ReMaP* (ours)

Fig. 4: Placement visualization of design bp_quad.

analysis, we increase the frequency of cases without violated timing paths. Then we place macros using different placers, followed by global placement, clock tree synthesis, and routing using OpenROAD [1]. The released version[3] of DREAMPlace 4.1.0 [2] is used as our mixed-size placer for prototyping.

A. Main Results

Table I compares post-route PPA metrics of three baselines and our ReMaP method. We use RTL-MP [9] and Hier-RTLMP [10] with default settings from ORFS, and DREAMPlace 4.1.0 with macro placement enhancements [2]. ReMaP achieves the best WNS and TNS in all eight cases, with average improvements of 8.39% and 16.57%, and maximum improvements of 34.15% and 65.39% for WNS and TNS compared to the strongest baseline on each case. ReMaP also performs above average in routing wirelength, power, and routing overflow, excelling in overflow due to the regularly placed macros.

Fig. 4 visualizes the placement results. RTL-MP shows regular placement but poor timing, while Hier-RTLMP improves timing but lacks regularity with large notch regions.

[3]https://github.com/limbo018/DREAMPlace/releases/tag/4.1.0

TABLE II: Ablation results.

Benchmark	Methods	WNS (ns)	TNS (ns)	Overflow (#)
ariane133	*Fix_Density_0.5*	-1.20	-3051.71	121
	Fix_Density_0.91	**-1.06**	-2636.54	9
	Overlap_Criteria	**-1.06**	-2658.09	18
	Default	-1.07	**-2615.74**	24
ariane136	*Fix_Density_0.5*	-2.14	-6385.50	6
	Fix_Density_0.91	-2.09	-6260.21	2
	Overlap_Criteria	-2.23	-6726.42	4
	Default	**-2.04**	**-6085.84**	7
black_parrot	*Fix_Density_0.5*	-4.82	-1278.52	5669
	Fix_Density_0.91	**-4.66**	-1078.88	2535
	Overlap_Criteria	-5.09	**-880.59**	977
	Default	-4.67	-1090.18	4548
bp_be	*Fix_Density_0.5*	**-0.89**	**-87.29**	30745
	Fix_Density_0.91	-1.02	-118.58	36056
	Overlap_Criteria	-0.91	-100.52	31225
	Default	-0.90	-98.00	31288
bp_fe	*Fix_Density_0.5*	-0.55	-36.08	8843
	Fix_Density_0.91	-0.37	-19.68	2627
	Overlap_Criteria	-0.65	-43.50	7684
	Default	**-0.27**	**-6.88**	15936
bp_multi	*Fix_Density_0.5*	-5.85	-7021.07	493
	Fix_Density_0.91	-6.21	-6376.50	471
	Overlap_Criteria	**-5.67**	**-5937.93**	26
	Default	-5.82	-6190.78	160
bp_quad	*Fix_Density_0.5*	-0.95	-16328.90	4298
	Fix_Density_0.91	-1.40	-17404.60	8417
	Overlap_Criteria	-1.01	-16952.10	5784
	Default	**-0.93**	**-14139.20**	4629
swerv_wrapper	*Fix_Density_0.5*	**-1.12**	-1035.99	31050
	Fix_Density_0.91	-1.37	-1349.50	20305
	Overlap_Criteria	-1.13	-1061.89	21970
	Default	**-1.12**	**-979.03**	20072

DREAMPlace struggles with timing and core congestion. Our ReMaP achieves both superior timing and regularity.

B. Ablation Study

Section III-B discusses our choice of dynamic target density for mixed-size prototyping, which decreases from 0.91 to 0.5 across all eight cases. We also explore fixed densities of 0.5 and 0.91. Additionally, we propose two criteria for macro relocating order in Section III-D. By default, we prioritize weaker dataflow strength for better timing performance, but we also evaluate an overlap criterion for more regular placements. As shown in Table II, while the ablation methods occasionally excel, the default settings generally perform best. Given the variability in macro placement challenges, it's clear that no single fixed setting can address every scenario effectively.

C. Runtime Analysis

In Table I, we compare the runtime of our ReMaP method with baselines. For smaller cases, as shown in Fig. 5(a), a significant portion of our runtime is due to gradient calculations during prototyping. This is an inherent limitation of our approach owing to recursive prototyping. However, for larger cases like bp_quad, as shown in Fig. 5(b), our method's runtime does not escalate steeply, thanks to DREAMPlace's efficient gradient propagation. Despite the case scale increasing by over ten times, the absolute time spent on gradients only increases by about 12.5%. The majority of the runtime, however, is consumed by benchmark I/O, primarily due to DREAMPlace's inefficiency with recursive calls. We are working on improving the I/O efficiency of our framework, and these engineering challenges will be addressed soon. Our method shows potential for greater efficiency in modern, large-scale designs.

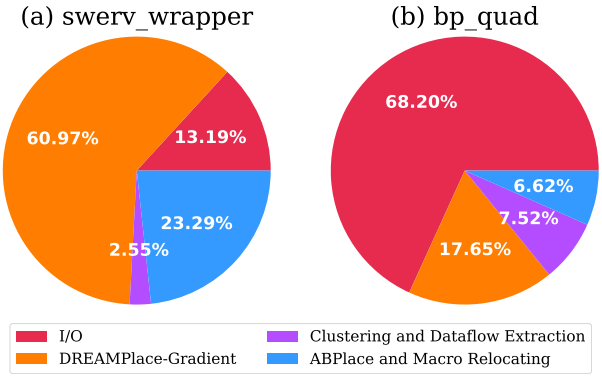


Fig. 5: Runtime breakdown of ReMaP on two representative designs swerv_wrapper and bp_quad.

TABLE III: Results of automatic parameter tuning.

Benchmark	Methods	Params		Metrics	
		γ	λ	WNS	TNS
ariane133	*hand-crafted*	0.943	0.006	-1.07	-2615.74
	50-round BO	0.956	0.001	**-1.02**	**-2484.17**
bp_be	*hand-crafted*	0.943	0.006	-0.90	-98.00
	50-round BO	0.990	0.440	**-0.78**	**-76.26**

D. Parameter Auto-tuning

As highlighted in Sec. III-E, our ReMaP method requires tuning of only two parameters: the decay factor γ and the Lagrange multiplier λ. While we manually adjust these parameters for our main results, we also explore their full potential using Bayesian optimization (BO) for automatic tuning. Table III shows that after 50 rounds of BO, there is an average improvement of 8.75% on WNS and TNS metrics in the ariane133 and bp_be cases compared to our manually tuned parameters.

V. Conclusion

In this work, we introduce and open-source the ReMaP framework, which optimizes macro placement through recursively prototyping and relocating. Our ABPlace method uses angle-based distribution to place macros on an ellipse, followed by periphery-guided relocating, yielding regular layouts with superior timing performance, as confirmed by extensive experiments. While recursive prototyping may reduce efficiency in smaller cases, ReMaP shows promise for scaling to larger cases after addressing engineering challenges.

VI. Acknowledgement

This work was supported by the National Science and Technology Major Project (2022ZD0116600), Jiangsu Science Foundation Leading-edge Technology Program (BK20232003), National Science Foundation of China (62276124, 624B2069), Fundamental Research Funds for the Central Universities (14380020), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Chao Qian is the corresponding author.

REFERENCES

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 2019, pp. 1–4.

[2] Y. Chen, Z. Wen, Y. Liang, and Y. Lin, "Stronger mixed-size placement backbone considering second-order information," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Francisco, CA, 2023, pp. 1–9.

[3] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2018.

[4] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," in *Advances in Neural Information Processing Systems*, Virtual, 2021.

[5] M. Fogaca, A. B. Kahng, E. Monteiro, R. Reis, L. Wang, and M. Woo, "On the superiority of modularity-based clustering for determining placement-relevant clusters," *Integration*, vol. 74, pp. 32–44, 2020.

[6] Z. Geng, J. Wang, Z. Liu, S. Xu, Z. Tang, M. Yuan, J. Hao, Y. Zhang, and F. Wu, "Reinforcement learning within tree search for fast macro placement," in *Proceedings of the International Conference on Machine Learning*, Vienna, Austria, 2024.

[7] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, "Corner block list: An effective and efficient topological representation of non-slicing floorplan," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, 2000, pp. 8–12.

[8] A. B. Kahng, S. Kang, S. Kundu, K. Min, S. Park, and B. Pramanik, "PPA-relevant clustering-driven placement for large-scale VLSI designs," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, 2024.

[9] A. B. Kahng, R. Varadarajan, and Z. Wang, "RTL-MP: toward practical, human-quality chip planning and macro placement," in *Proceedings of the International Symposium on Physical Design*, New York, NY, 2022, pp. 3–11.

[10] A. B. Kahng, R. Varadarajan, and Z. Wang, "Hier-RTLMP: A hierarchical automatic macro placer for large-scale complex ip blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 5, pp. 1552–1565, 2023.

[11] A. B. Kahng and Z. Wang, "DG-RePlAce: A dataflow-driven gpu-accelerated analytical global placement framework for machine learning accelerators," *arXiv:2404.13049*, 2024.

[12] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao, and P. Luo, "ChiPFormer: Transferable chip placement via offline decision transformer," in *Proceedings of the International Conference on Machine Learning*, Honolulu, HI, 2023.

[13] Y. Lai, Y. Mu, and P. Luo, "MaskPlace: Fast chip placement via reinforced visual representation learning," in *Advances in Neural Information Processing Systems*, Virtual, 2022.

[14] J.-M. Lin, Y.-L. Deng, S.-T. Li, B.-H. Yu, L.-Y. Chang, and T.-W. Peng, "Regularity-aware routability-driven macro placement methodology for mixed-size circuits with obstacles," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 1, pp. 57–68, 2019.

[15] J.-M. Lin, Y.-L. Deng, Y.-C. Yang, J.-J. Chen, and P.-C. Lu, "Dataflow-aware macro placement based on simulated evolution algorithm for mixed-size designs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 5, pp. 973–984, 2021.

[16] J.-M. Lin, W.-F. Huang, Y.-C. Chen, Y.-T. Wang, and P.-W. Wang, "DAPA: A dataflow-aware analytical placement algorithm for modern mixed-size circuit designs," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, Munich, Germany, 2021, pp. 1–8.

[17] J.-M. Lin, S.-T. Li, and Y.-T. Wang, "Routability-driven mixed-size placement prototyping approach considering design hierarchy and indirect connectivity between macros," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 2019, pp. 1–6.

[18] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern VLSI placement," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 2019, pp. 1–6.

[19] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[20] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518–1524, 1996.

[21] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 1, pp. 87–100, 1984.

[22] Y. Pu, T. Chen, Z. He, C. Bai, H. Zheng, Y. Lin, and B. Yu, "IncreMacro: Incremental macro placement refinement," in *Proceedings of the International Symposium on Physical Design*, New York, NY, 2024, pp. 169–176.

[23] Y. Shi, K. Xue, S. Lei, and C. Qian, "Macro placement by wire-mask-guided black-box optimization," in *Advances in Neural Information Processing Systems*, New Orleans, LA, 2023.

[24] A. Vidal-Obiols, J. Cortadella, J. Petit, M. Galceran-Oms, and F. Martorell, "RTL-aware dataflow-driven macro placement," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition*, Florence, Italy, 2019, pp. 186–191.

[25] M.-C. Wu and Y.-W. Chang, "Placement with alignment and performance constraints using the B*-tree representation," in *Proceedings of IEEE International Conference on Computer Design*, San Jose, CA, 2004, pp. 568–571.

[26] K. Xue, R.-T. Chen, X. Lin, Y. Shi, S. Kai, S. Xu, and C. Qian, "Reinforcement learning policy as macro regulator rather than macro placer," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2024.

[27] J. Z. Yan, N. Viswanathan, and C. Chu, "Handling complexities in modern large-scale mixed-size placement," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, 2009, pp. 436–441.

[28] X. Zhao, T. Wang, R. Jiao, and X. Guo, "Standard cells do matter: Uncovering hidden connections for high-quality macro placement," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition*, Valencia, Spain, 2024, pp. 1–6.

Dual-Issue Execution of Mixed Integer and Floating-Point Workloads on Energy-Efficient In-Order RISC-V Cores

Luca Colagrande
Integrated Systems Laboratory (IIS)
ETH Zurich
Zurich, Switzerland
colluca@iis.ee.ethz.ch

Luca Benini
Integrated Systems Laboratory (IIS)
ETH Zurich
Zurich, Switzerland
lbenini@iis.ee.ethz.ch

***Abstract*—To meet the computational requirements of modern workloads under tight energy constraints, general-purpose accelerator architectures have to integrate an ever-increasing number of extremely area- and energy-efficient processing elements (PEs). In this context, single-issue in-order cores are commonplace, but lean dual-issue cores could boost PE IPC, especially for the common case of mixed integer and floating-point workloads. We develop the COPIFT methodology and RISC-V ISA extensions to enable low-cost and flexible dual-issue execution of mixed integer and floating-point instruction sequences. On such kernels, our methodology achieves speedups of 1.47x, reaching a peak 1.75 instructions per cycle, and 1.37x energy improvements on average, over optimized RV32G baselines.**

***Index Terms*—RISC-V, dual-issue, energy efficiency, general purpose**

I. Introduction

With the end of Dennard scaling, energy-efficiency has become a primary constraint in the design of high-performance computer architectures [1]. Simultaneously, the rapid growth of Machine Learning (ML) models, requiring immense computational resources for training and inference, are dramatically escalating the demand for high-performance computing power [2].

To meet this challenge, modern high-performance architectures integrate an ever-increasing number of cores, designed to meet extreme area- and energy-efficiency targets to enable large-scale replication [3]. In-order single-issue cores with shallow pipelines are favoured over power- and resource-hungry deeply-pipelined out-of-order processors.

Within the context of massively-parallel general-purpose accelerators, few architectures propose cores with limited multiple-issue capabilities. Nvidia's Turing architecture, for instance, implements concurrent execution of FP32 and INT32 operations [4], allowing the Floating-Point (FP) math datapath to remain active during the execution of non-FP-math instructions. According to the authors, this results in a 36% average increase in throughput across several gaming workloads. Unfortunately, due to the proprietary nature of the architecture, implementation details and their area and power impact remain undisclosed.

A similar form of dual-issue execution was implemented in the Snitch processor [5], coupling a tiny streamlined single-issue in-order RV32I control core and a 64-bit SIMD-capable Floating Point Unit (FPU). Instructions are fetched and issued by the control core, and offloaded to the FPU as required. To support dual-issue execution, the Floating Point Subsystem (FPSS) integrates a small instruction buffer to cache small FP loop bodies, the Floating Point Repetition (FREP) buffer. FP instructions are first issued by the integer core, and cached in the FREP buffer. In the remaining loop iterations, the FREP sequencer can independently issue instructions, while the integer core is free to fetch, and potentially execute, successive instructions. Its authors refer to this scheme as *pseudo dual-issue execution*.

This approach implies the following constraints:

- the FP loop body must be small enough to fit entirely in the FREP buffer;
- the programmer or compiler need to explicitly mark the FP loop for FREP execution in software;
- there can be no dependencies between the FP and integer threads.

The latter constraint follows from the asynchronous execution of the two threads, between which no ordering guarantees are enforced. While this enables Snitch's dual-issue capabilities at a very low power and area cost, it limits flexibility. In particular, it hinders the execution of computations featuring a mix of interdependent integer and FP instructions. We refer to these as mixed integer and FP instruction sequences.

In this paper, we present *COPIFT*, a generic method to develop **C**o-**O**perative **P**arallel **I**nteger and **F**loating-point **T**hreads in Snitch, overcoming the limitation imposed by pseudo dual-issue, and enabling low-cost dual-issue execution of mixed integer and FP instruction sequences.

To summarize our contributions, we:

- Present COPIFT, a generic methodology to enable sustained dual-issue execution of mixed integer and

FP instruction sequences in Snitch, a state-of-the-art open-source RISC-V processor.
- Develop a set of Instruction Set Architecture (ISA) extensions to broaden the range of applications suited for pseudo dual-issue execution, in conjunction with the COPIFT method.
- Implement COPIFT-accelerated codes for selected mixed integer and FP instruction sequences, achieving average speedups of 1.47× over optimized RV32G baselines, and a peak IPC of 1.75.
- Present a first account of the energy-efficiency of pseudo multiple-issue execution schemes, demonstrating average energy reductions of 1.37× with COPIFT, over optimized RV32G baselines.

Our implementation is fully open source and performance experiments are reproducible using free software. [1]

II. Implementation

A. COPIFT Methodology

Akin to other ISAs, RISC-V defines distinct Register Files (RFs) for the RV32I base integer ISA and the "D" standard extension [6]. As a result, integer and FP instructions operate mostly on independent sets of registers. This fundamental characteristic allows us to define two threads with independent state, comprising respectively the integer RF and RV32I instructions and the FP RF and "D" extension instructions. Several notable FP instructions, however, also require access to the integer RF, consuming operands from integer source registers (FP load, store, conversion and move instructions) or producing results into integer destination registers (FP comparisons). Such instructions conflict with the independent thread abstraction, precluding pseudo-dual issue execution.

Partly relaxing this constraint, Snitch implements Stream Semantic Registers (SSRs) [7], a custom ISA extension to stream data from memory directly into the RF, eliminating explicit load and store instructions. SSRs support any *affine stream* or memory access pattern which can be described as an affine function of the induction variables of a four-dimensional loop nest. Extending the SSR mechanism, Indirection Stream Semantic Registers (ISSRs) [8], further provide support for *indirect streams* or arbitrary memory access patterns, provided a list of relative addresses encoding the pattern. SSRs enable memory accesses without introducing dependencies on the integer RF, solving only a particular instance of the problem. Conversely, our methodology tackles any form of interdependency between integer and FP threads.

Consider the code in Figure 1a, computing the exponential of an array of single-precision FP numbers. Compiling this code for the RV32G architecture results in the mix of integer and FP instructions shown in Figure 1b. Dependencies between these instructions are marked with red and blue boxes in the consuming and producing instructions, respectively. These can be classified into:

Type 1 Dynamic memory dependencies, from FP load-stores with dynamically-computed addresses.

Type 2 Static memory dependencies, from FP load-stores with statically-determined addresses.

In the most general case, a third type may be present:

Type 3 Register dependencies, from FP conversion, move and comparison instructions.

Due to these dependencies, we cannot trivially partition integer and FP instructions for pseudo dual-issue execution. The COPIFT method addresses the problem by eliminating or increasing the distance between dependent instructions. It involves the following steps:

Step 1 Construct a Data Flow Graph (DFG) from the RISC-V assembly and identify all dependencies between integer and FP instructions.

Step 2 Partition the DFG on the identified edges, such that there exists an acyclic precedence-relation among the subgraphs, and the number of edges between them is minimized.

Step 3 Reorder the instructions according to the subgraph ordering established in Step 2.

Step 4 Apply loop tiling to block the data, and loop fission to reschedule the subgraph computations on each block.

Step 5 Apply software pipelining and multiple buffering to overlap subgraph computations.

Step 6 Map all FP load-store instructions to SSRs.

Step 7 Map all FP loops to FREP loops, and reorder them to preceed the integer loop.

To clarify each step, we walk through how to apply COPIFT to the exponential kernel in Figure 1a.

Figure 1c shows the DFG of the basic block in Figure 1b. For brevity, we omitted instructions 24 and 25 and associated dependencies, as we will see that these can be simplified in Step 6. Following Step 2, we cut edges $4 \rightarrow 5$, $12 \rightarrow 18$ and $14 \rightarrow 18$. This alone results in two subgraphs for which a precedence-relation cannot be established. We therefore further partition the graph, cutting edge $21 \rightarrow 22$. We obtain three subgraphs, each defining a phase of the computation with clear ordering requirements w.r.t. the others: FP Phase 0 must precede integer Phase 1, which in turn must precede FP Phase 2. This allows us to reorder the instructions in consecutive groups of integer-only or FP-only instructions, while respecting the dependencies within each loop iteration (Step 3). The resulting code would resemble Figure 1d, where each function would be replaced by the respective inline assembly code.

In Step 4, we then apply loop tiling [9] to obtain the code in Figure 1e. Each phase is now applied to an entire block of inputs, before passing on a block of intermediate results to the next phase. Implicitly, this step requires allocating block-sized buffers for each variable associated to an edge between subgraphs. Additionally,

[1] https://github.com/colluca/snitch_cluster/tree/copift

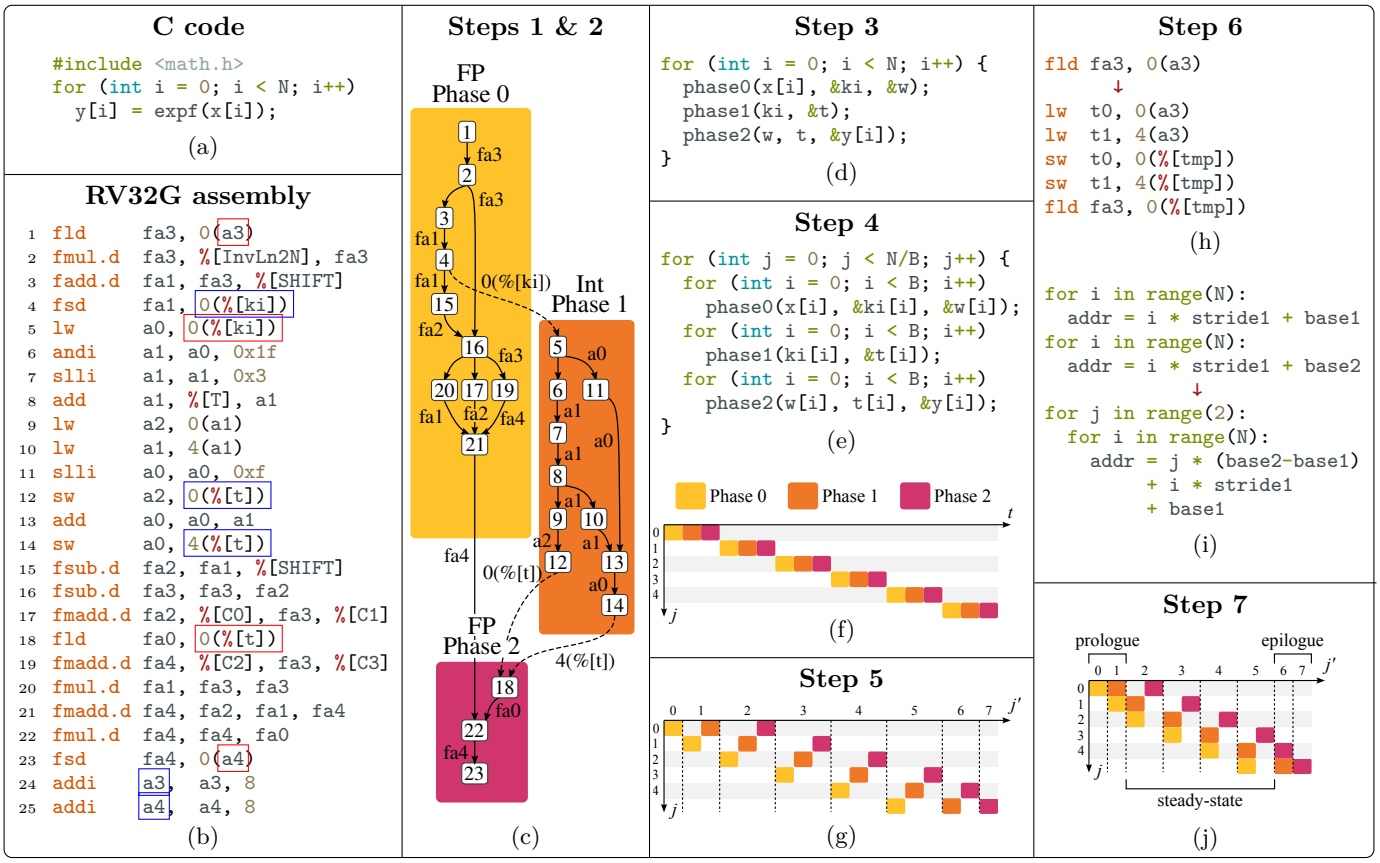


Fig. 1: Illustration of the COPIFT methodology applied to a kernel calculating the exponential of a vector.

all inter-phase communication originally through the RF, must be preventively spilled to memory. To reduce spill communication overheads and memory requirements, it is important to minimize the number of edges between subgraphs in Step 2.

Step 5 prepares for the parallel execution of the FP and integer phases, by applying software pipelining [10] to increase the distance between dependent phases. Figures 1f and 1g illustrate the original and modified block schedules, respectively. Within each j' iteration in the new schedule, every phase operates on different blocks of data j, allowing them to be parallelized in Step 7. Implicitly, this requires replicating the buffers, as e.g. Phase 0 produces different blocks of w before the first is consumed by Phase 2. The exact number of replicas for each buffer equals the distance between the subgraphs connected by the respective edge in the total ordering established in Step 2, plus one. In our example, the w buffer, associated to the edge between Phase 0 and 2, must be replicated three times.

Steps 6 to 7 enable the integer and FP phases within each block iteration to be executed in parallel. In Step 4, we spilled all inter-phase communication through the RFs to memory, introducing additional load-store instructions. In Step 6, we eliminate all FP load-stores by mapping the respective memory accesses to SSRs. The accesses performed by the two FP phases in our example can be described as a combination of 6 one-dimensional streams: reads from blocks of x, w and t, and writes to blocks of w, ki and y. All streams originate from tiling in Step 4, and can thus be naturally represented as regular accesses into contiguous arrays. Special treatment is required to convert streams of addresses originating from Type 1 dependencies, into regular streams of data for the FP thread. This can be done in software by converting Type 1 to Type 2 dependencies as shown in Figure 1h. This involves prefetching the data in the integer thread and storing it in a regular layout for the SSRs. Alternatively, Type 1 dependencies can be directly mapped to ISSRs, performing the indirection on the regular stream of addresses in hardware. Finally, we note that in cases where the Type 1 address stream can be statically generated by an affine function of up to four variables, the entire address calculation can be eliminated and performed by an SSR, which is the case for the x and y streams in our example. The associated dependency and all artifacts produced by our methodology can be optimized away, which allowed us to ignore instructions 24 and 25 from the start.

To map the 6 streams to the 3 SSRs available in every Snitch core, we apply a *stream fusion* technique, which involves mapping multiple lower-dimensional affine streams

Kernel	Baseline			Step 4		Steps 5 to 6		Max Block	COPIFT		I'	S''	▼S'
	#Int	#FP	TI	Int Ld/St	#Buff.	FP Ld/St	#Buff.		#Int	#FP			
`expf`	43	52	0.83	0	5	-4	13	157	43	36	1.84	1.83	2.21
`logf`*	39	52	0.75	+4	6	-4	12	273	57	36	1.63	1.75	1.6
`poly_lcg`*†	44	80	0.55	+3	3	0	6	341	72	80	1.9	1.55	1.55
`pi_lcg`*†	44	56	0.79	+3	3	0	6	341	72	56	1.78	1.79	1.39
`poly_xoshiro128p`*†	172	80	0.47	+3	3	0	6	341	200	80	1.4	1.47	1.26
`pi_xoshiro128p`*†	172	56	0.33	+3	3	0	6	341	200	56	1.28	1.33	1.14

Require support for *`fcvt.d.w` and †`flt.d`.

TABLE I: Characteristics of the evaluated kernels, ordered by expected speedup S'.

to a single higher-dimensional affine stream. Figure 1i illustrates the process of merging two one-dimensional affine streams. Using this technique, we merge the x and t read streams and the w, ki and y write streams.

Finally, FP block computations are mapped to FREP loops (Step 7). Since instructions for the first iteration of an FREP loop are issued by the integer core, the FREP loops must preceed the integer loop in program order, such that remaining iterations overlap with the integer thread. Similarly, for the integer thread to overlap with both FP Phase 0 and 2, these should be fused to a single FREP loop in block iterations where both phases are performed. The resulting schedule is shown in Figure 1j.

The steps in this methodology can be followed by developers to obtain an optimized mixed C and assembly representation.

B. COPIFT ISA Extensions

FP conversion and comparison instructions are critical for mixed integer and FP workloads. To support these, we can alter their semantics under FREP operation, to operate entirely on the FP RF. Communication between the integer and FP RFs is spilled to memory, through the use of additional load-store instructions, decoupling it from the computation, so the target instructions can find all operands directly in the FP RF. This technique can be combined with SSRs, to implicitly encode the FP load-store operations.

Albeit not strictly necessary, we chose to develop custom instructions to carry the altered semantics, for each of the relevant "D" extension instructions, i.e. `fcvt.w[u].d`, `fcvt.d.w[u]`, `feq.d`, `flt.d`, `fle.d` and `fclass.d`. We copy the original encodings, allocating the new instructions in the `custom-1` opcode space [6].

III. Results

We implement a Snitch cluster hosting one compute core in GlobalFoundries' 12LP+ FinFET technology using Fusion Compiler, with a target clock frequency of 1 GHz. Our extensions introduce negligible area and timing overheads, within the margin of synthesis process variability.

All experiments are conducted at the target frequency in cycle-accurate RTL simulations using QuestaSim 2023.4. Switching activities are extracted from post-layout simulations, and used for power estimation in PrimeTime, assuming typical operating conditions of 25 °C and 0.8 V supply voltage.

A. Performance Evaluation

To evaluate the benefits of the COPIFT method and ISA extensions, we implement a set of mixed integer and FP workloads with diverse characteristics, extracted from two different domains:

- Hit and miss Monte Carlo integration methods [11], evaluating two different integration problems and pseudo-random number generators. Integer instructions are typically employed in pseudo-random number generation, while the integration is performed in double-precision FP arithmetic.
- Transcendental function evaluation on a vector of single-precision FP numbers. Reference implementations are taken from the GNU C Library [12] v2.40, and involve integer instructions for bit-manipulation and double-precision FP arithmetic to evaluate approximating polynomials.

The latter in particular, applied to the exponential function, is the main component of softmax operations, which consume a considerable fraction of cycles in modern Large Language Models (LLMs) [13].

We implement Snitch-optimized RV32G-compliant baseline codes and COPIFT-accelerated variants for each kernel, compiled using a custom toolchain for Snitch based on LLVM 18.1.4 with −O3 optimization.

Table I lists all of the evaluated kernels, with their unique characteristics. As the table shows, the number of integer and FP instructions can vary as a result of applying the COPIFT method, as e.g. integer load-stores are introduced in Step 4 and FP load-stores are eliminated in Step 6. These easily measurable kernel characteristics can be used to estimate the speedup potential of the COPIFT method. In fact, assuming similar IPCs in the integer and FP threads, the speedup S can be approximated as a function of the number of integer and FP instructions:

$$S \coloneqq \frac{t_{\text{int}}^{\text{base}} + t_{\text{fp}}^{\text{base}}}{\max(t_{\text{int}}^{\text{copift}}, t_{\text{fp}}^{\text{copift}})} \approx \frac{n_{\text{int}}^{\text{base}} + n_{\text{fp}}^{\text{base}}}{\max(n_{\text{int}}^{\text{copift}}, n_{\text{fp}}^{\text{copift}})} = S' \quad (1)$$

Similarly, we can approximate the IPC improvement I:

$$I \coloneqq S * \frac{n_{\text{int}}^{\text{copift}} + n_{\text{fp}}^{\text{copift}}}{n_{\text{int}}^{\text{base}} + n_{\text{fp}}^{\text{base}}} \approx \frac{n_{\text{int}}^{\text{copift}} + n_{\text{fp}}^{\text{copift}}}{\max(n_{\text{int}}^{\text{copift}}, n_{\text{fp}}^{\text{copift}})} = I' \quad (2)$$

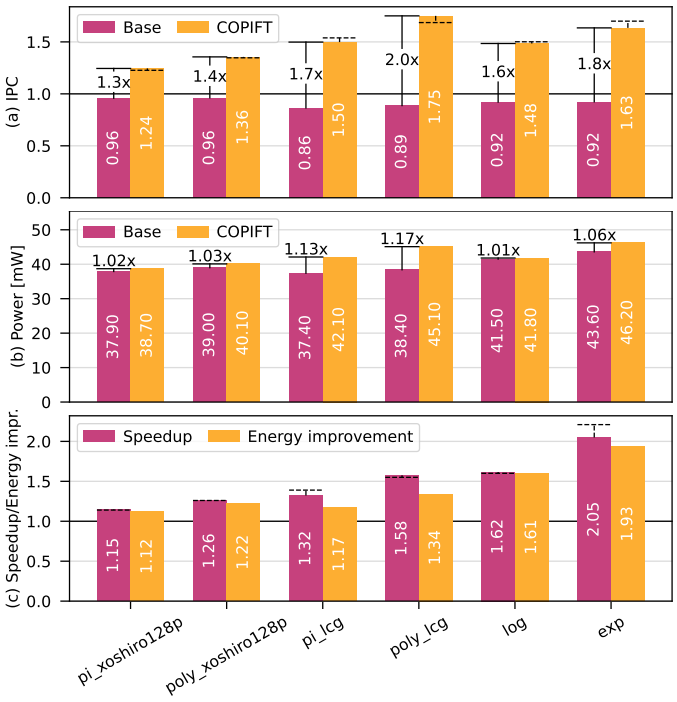


Fig. 2: Comparison of various metrics between COPIFT and baseline codes. Dashed lines indicate expected values: (a) the IPC derived from I' and (c) the speedup S'.

If the number of instructions does not differ significantly between the two code variants, we can further approximate the speedup as a function of the number of instructions in the original implementation alone:

$$S' \approx \frac{n_{\text{int}}^{\text{base}} + n_{\text{fp}}^{\text{base}}}{\max(n_{\text{int}}^{\text{base}}, n_{\text{fp}}^{\text{base}})} \overset{2}{=} 1 + \frac{\min(n_{\text{int}}^{\text{base}}, n_{\text{fp}}^{\text{base}})}{\max(n_{\text{int}}^{\text{base}}, n_{\text{fp}}^{\text{base}})} = \\ = 1 + \text{TI} = S'' = I'' \quad (3)$$

Intuitively, this shows how the speedup approximately depends on the thread imbalance (TI), as defined above.

Figure 2a compares the IPC of a steady-state iteration in the two implementations, for all kernels ordered by increasing expected speedup S'. We observe a significant geomean IPC improvement of 1.62×, and a peak IPC of 1.75, as a demonstration of dual-issue performance. As hypothesized, the IPC improvement correlates with I'. The deviations observed in the `poly_lcg` and `pi_lcg` kernels are due to the hypothesis made in Equation (1) being invalid. In fact, both benchmarks present stalls in the Pseudo-Random Number (PRN) generation with the Linear Congruential Generator (LCG), which are due to structural hazards on the register file's writeback port, and could not be eliminated by unrolling. These stalls effectively balance out the execution times of the integer and FP threads in the `poly_lcg` kernel, leading to a near-ideal IPC improvement of 1.97×, while unbalancing the

[2]Using equality $a + b = \max(a, b) + \min(a, b)$

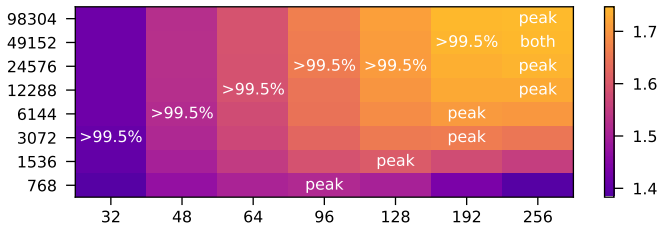


Fig. 3: IPC of the `poly_lcg` kernel for various problem and block sizes.

`pi_lcg` kernel. Finally, the `exp` kernel also shows a small deviation. This is due to the instruction overhead required to program the SSRs and switch buffers in every block iteration, which is not amortized as effectively on smaller blocks, as this kernel features.

To understand the impact of block size and problem size on performance, in Figure 3 we plot the IPC for various configurations of the `poly_lcg` kernel. As expected, IPC increases with problem size, as fixed prologue and epilogue overheads are amortized over a larger number of block iterations. Smaller blocks result in smaller prologue and epilogue overheads and a larger number of block iterations, for equal problem sizes, and can thus amortize these overheads more effectively. This trend is displayed in the ">99.5%" annotations, marking the smallest problem size which achieves >99.5% of the maximum IPC attainable with each block size. On the other hand, smaller blocks are less able to amortize overheads within the block, e.g. from SSR programming and switching buffers. As such, smaller block sizes feature lower maximum IPCs, when all other overheads are fully amortized. In this tradeoff, there exists an optimal block size per problem size, which we annotate with the "peak" labels. As the problem size increases, the peak shifts towards larger block sizes, up to the maximum size we can fit in L1 memory, reported in Table I. Finally, we note that, as we tend to amortize all overheads, the IPC converges to the steady-state IPC presented in Figure 2a. In the following discussions we will thus restrict our analysis to steady-state measurements.

Figure 2c shows how the IPC increase translates to improved kernel performance. Again, we observe a significant geomean speedup of 1.47×, with a peak 2.05× speedup on the `exp` kernel. The deviations from the expected speedup S' are a consequence of the same overheads affecting the IPC. Finally, we note that speedups greater than two are possible, as a result of additional optimizations, such as load-store elision with the SSRs, on top of dual-issue execution.

B. Energy Evaluation

Figure 2b compares the power consumption of the two implementations. The Monte Carlo benchmarks exhibit a lower base power consumption than their counterparts, due to the inactivity of the Direct Memory Access (DMA) engine and reduced L1 memory accesses. The increase in

power is consistently correlated with the IPC increase, with exception of the `log` and `exp` kernels. In fact, after separating the FP instructions, the integer loop bodies of these two kernels comprise less than 64 instructions, which entirely fit in Snitch's L0 I$. As a result, the I$ consumes significantly less power in the COPIFT implementations, which do not experience thrashing at the L0 level.

Ultimately, as the power consumption is dominated by constant components such as the clock network activity, we observe a maximum power increase of only 1.17×, and a geomean power increase of 1.07× over all kernels. As a result, COPIFT is still largely beneficial in overall energy terms, as shown in Figure 2c. The reduced execution time, resulting from dual-issue execution, outweighs the increase in power consumption, resulting in geomean energy savings of 1.37× on the measured benchmarks, and a peak 1.93× saving on the `exp` kernel.

IV. Related Work

To issue multiple instructions in parallel while preserving program correctness, multiple-issue processors must be able to identify dependencies within an instruction stream. Multiple-issue capabilities are often built into out-of-order processors, leveraging the facilities which enable out-of-order execution, such as reservation stations, register renaming and reorder buffers, which lead to high area and power overheads. On the other hand, in-order multiple-issue processors can be implemented in two flavours: statically-scheduled superscalar processors and VLIW processors [14]. In-order superscalar processors integrate logic to identify hazards between instructions in the issue packet and those in execution, while VLIW processors offload part of the job to the compiler, in charge of packing independent instructions into bundles of a predetermined size, at the cost of increased code size.

Few recent works in the literature look into the area- and energy-efficient design of dual-issue in-order RISC processors. All of these target dual-issue execution of arbitrary instruction pairs, with minor restrictions; as a consequence, they require a minimum of four read and two write (4R2W) ports into the RF. Lozano et al. [15] design a novel latency-optimized 4R2W-port SRAM-based RF implementation for soft FPGA cores, yet their implementation incurs a 4× cost in physical RAM blocks compared to the single-issue baseline, which also accounts for the majority of the +18% power increase. Similarly, Kra et al. [16] implement a dual-issue in-order RISC-V core based on CV32E40P, duplicating instruction decode, issue and execution stages and integrating a 5R3W-port RF with cross-forwarding support from all write to all read ports, for a total 60% area increase. Patsidis et al. [17] claim to further push the area and energy efficiency limits of dual-issue RISC-V cores, by restricting dual-issue to 16-bit compressed instructions, fitting within a single 32-bit instruction fetch port. They extend the RF with extra read ports, and a partitioned design to harvest the energy benefits from locality of reference to eight commonly-addressed registers by compressed instructions.

Wygrzvwalski et al. [18] present a pseudo dual-issue mechanism for small loop bodies, which resembles the Snitch implementation. A separate instruction buffer is allocated as a source of instructions for the second issue slot. Differently from Snitch, they rely on a banked RF design to prevent structural hazards on the RF ports, restricting parallel execution to pairs of instructions accessing different register banks. Conversely, Snitch targets pseudo dual-issue execution of integer and FP instructions, leveraging the separate RFs available by ISA design. Unfortunately, Wygrzvwalski's design is targeted to a specific application, limiting their investigation to a rather simple code featuring two independent threads operating in lockstep, without diving into the intricacies required to support more complex general-purpose scenarios.

In contrast to these works, we presented a novel methodology enabling dual-issue execution of mixed integer and FP instruction sequences in an area- and energy-efficient processor, with virtually no area overhead due to our extensions. This is possible by leveraging Snitch's SSR and FREP extensions, which extend beyond the scope of dual-issue execution, enabling the core's high FPU utilization.

V. Conclusion

In this work, we presented COPIFT, a generic methodology to enable sustained dual-issue execution of mixed integer and floating-point instruction sequences in Snitch, a state-of-the-art open-source RISC-V processor. We further developed a set of ISA extensions to broaden the range of applications which can leverage our methodology. To evaluate the benefits of COPIFT-enabled dual-issue execution, we implemented optimized RV32G baseline codes for a variety of mixed integer and floating-point workloads on Snitch, and compared these to COPIFT-optimized variants leveraging our ISA extensions. We measured speedups of 1.47× on average and a peak IPC of 1.75, proving COPIFT's effectiveness in enabling sustained dual-issue execution. We further compared the energy-efficiency of baseline and COPIFT-accelerated codes. Despite a small 1.07× average increase in power consumption, COPIFT proves beneficial in overall energy terms, resulting in 1.37× average energy savings on the measured benchmarks, and a peak 1.93× saving on the `exp` kernel, proving that effective dual-issue execution is possible on area- and energy-efficient in-order cores.

Acknowledgment

This work has been supported in part by 'The European Pilot' project under grant agreement No 101034126 that receives funding from EuroHPC-JU as part of the EU Horizon 2020 research and innovation programme.

We would like to thank Lannan Jiang, Fabio Cappellini and Hakim Filali for their early experiments with Monte Carlo methods on Snitch.

References

[1] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[2] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[3] J. Choquette, "Nvidia hopper h100 gpu: Scaling performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.

[4] J. Burgess, "Rtx on—the nvidia turing gpu," *IEEE Micro*, vol. 40, no. 2, pp. 36–44, 2020.

[5] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845–1860, 2021.

[6] The risc-v instruction set manual volume i, unprivileged architecture, version 20240411. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/tag/20240411

[7] F. Schuiki, F. Zaruba, T. Hoefler, and L. Benini, "Stream semantic registers: A lightweight risc-v isa extension achieving full compute utilization in single-issue cores," *IEEE Transactions on Computers*, vol. 70, no. 2, p. 212–227, Feb. 2021.

[8] P. Scheffler, F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Sparse stream semantic registers: A lightweight isa extension accelerating general sparse linear algebra," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 12, p. 3147–3161, Oct. 2023.

[9] E. Hammami and Y. Slama, "An overview on loop tiling techniques for code generation," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, 2017, pp. 280–287.

[10] A. Douillet and G. R. Gao, "Software-pipelining on multi-core architectures," in *2007 16th International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2007, pp. 39–48.

[11] *Monte Carlo Integration*. John Wiley & Sons, Ltd, 2014, ch. 2, pp. 31–64. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527683147.ch2

[12] Gnu c library. [Online]. Available: https://sourceware.org/git/?p=glibc.git

[13] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, p. 469–474.

[14] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[15] H. B. Lozano and M. Ito, "Energy efficient dual-issue processor for embedded applications," in *2015 IEEE 12th International Conference on Networking, Sensing and Control (ICNSC)*, 2015, pp. 544–549.

[16] Y. Kra, Y. Shoshan, Y. Rudin, and A. Teman, "Hamsa-di: A low-power dual-issue risc-v core targeting energy-efficient embedded systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 1, pp. 223–236, 2024.

[17] K. Patsidis, D. Konstantinou, C. Nicopoulos, and G. Dimitrakopoulos, "A low-cost synthesizable risc-v dual-issue processor core leveraging the compressed instruction set extension," *Microprocessors and Microsystems*, vol. 61, pp. 1–10, 2018.

[18] M. Wygrzvwalski, P. Skrzypiec, and R. Szczygiel, "Hardware acceleration method using risc-v core with no isa extensions," in *2024 31st International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*, 2024, pp. 265–269.

A Cutting-Edge Parallel Solver for Scalable Power Grid Analysis Using Nested Domain Decomposition

Jianfei Song[1], Xiaoyu Yang[2], Zhou Jin[1], Cheng Zhuo[1*]
[1]Zhejiang University, Hangzhou, China
[2]China University of Petroleum-Beijing, Beijing, China
[*]Corresponding author: czhuo@zju.edu.cn

***Abstract*—As transistor scaling approaches sub-5nm technologies, power distribution networks (PDNs) in integrated circuits have grown increasingly complex, with billions to trillions of nodes. Simultaneously, reduced noise margins and increased power density necessitate more accurate and efficient power grid analysis. Traditional methods for solving large-scale PDNs, especially those requiring the solution of sparse linear systems, face significant challenges due to high computational costs. Although domain decomposition methods (DDM) allow for efficient parallel computation, the size of the dense global Schur complement grows excessively large as the number of partitions increases, limiting scalability and imposing substantial computational burdens. This paper introduces an efficient parallel nested domain decomposition solver that incorporates a parallel Schur complement computation strategy and intermediate Schur complement to address these challenges. Experimental results demonstrate that by introducing an intermediate Schur complement, the size of the global Schur complement is significantly reduced, achieving an average 1.70× speedup in computation, which results in a 1.30× speedup for the entire solver compared to the conventional DDM parallel solver.**

I. Introduction

As Moore's law drives transistor scaling to its limits, power distribution networks (PDNs) in integrated circuits have grown increasingly complex, encompassing billions to even trillions of nodes in sub-5nm technologies [1], [2]. Meanwhile, reduced noise margins, now as low as tens of millivolts due to lower supply voltages, and rising power density [3], [4] have made accurate and efficient power grid analysis more critical than ever. However, the computational demands of analyzing large-scale PDNs significantly hinder design efficiency and iteration speed. Consequently, enhancing the efficiency of power grid analysis methods has become a pivotal research focus.

Power grid analysis typically involves solving large-scale symmetric positive definite (SPD) linear systems, which can be addressed using direct or iterative solvers. Direct methods, such as Cholesky and LU decomposition [5], [6], provide exact solutions but are limited by high memory requirements, making them unsuitable for scalability on a single machine. Iterative methods, including Krylov subspace techniques [7], algebraic multigrid approaches [8], and graph spectral sparsification approaches [9], [10] offer better memory efficiency but often face stability and convergence challenges if preconditioners are not selected appropriately.

The rise of parallel computing technology has brought domain decomposition methods (DDM) to the forefront as a powerful approach for large-scale power grid analysis [11], [12]. These methods, which rely on graph partitioning, enable the division of PDN into smaller subdomains that can be processed in parallel, significantly improving computational efficiency. However, DDM relies on the Schur complement matrix. **Forming and solving the dense global Schur complement matrix can be even more demanding than directly solving the original system** [13], [14]**.** This challenge involves two main aspects:

- Calculating the local Schur complement for each subdomain is inherently complex and resource-demanding due to intricate interdependencies.
- While current computational resources are capable of handling many subdomains, excessive partitioning increases the size of the dense global Schur complement, making this large matrix challenging to parallelize and ultimately diminishing computational efficiency.

To overcome the above challenges, this paper introduces **an efficient parallel nested domain decomposition solver** specifically for power grid analysis, focusing on accelerating the formation of the local Schur complement, reducing the size of the global Schur complement matrix and addressing the limited parallelizability at the top level to ensure better scalability and resource utilization. This solver integrates multiple parallelization strategies to tackle the challenges associated with solving large-scale sparse linear systems, optimizing Schur complement matrix computation, and providing a comprehensive solution to enhance computational efficiency. The main contributions are as follows:

- **A parallel method for forming the Schur complement.** In this method, the nodes to be eliminated are divided into segments and processed in parallel, utilizing DDM to partition and solve in parallel blocks.
- **An intermediate Schur complement matrix to reduce the size of the dense global Schur complement.** This intermediate matrix can be naturally solved in parallel, further optimizing the overall process and improving computation and resource utilization.
- **A new solver that exploits the above optimizations to achieve higher scalability.** This solver overcomes the scalability limitations of conventional DDM solver, which struggles to achieve significant speedup as the number of partitions increases beyond a certain threshold.

Experimental results demonstrate that by introducing an intermediate Schur complement, the overall size of the global Schur complement is significantly reduced, effectively overcoming the limitations of traditional DDM. The intermediate Schur complement is well-suited for parallel computation, enabling task distribution to the intermediate layer and achieving a 1.70× speedup, which leads to our solver outperforming the conventional DDM solver with a 1.30× speedup. Furthermore, it achieves an 8.44× speedup over the CKTSO parallel solver and an 8.36× speedup over the CHOLMOD-based serial direct solver when utilizing 32 threads.

II. Background and Motivation

A. Introduction to DDM for Power Grid Analysis

DDM is a parallel method for solving large linear systems of the form $Gv = b$, particularly effective in PDN analysis [11], [12]. It partitions the PDN into subdomains by linking the coefficient matrix to an undirected graph and applying the Schur complement to enhance computational efficiency. Each subdomain contains internal nodes (fully within the subdomain) and interface nodes (shared between subdomains), forming a block sparse matrix that supports efficient parallel processing. The structure of the block sparse matrix is as follows:

$$\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{B}_1 & O & O & \cdots & O & O \\ \boldsymbol{B}_1^T & \boldsymbol{D}_1 & O & E_{1,2} & \cdots & O & E_{1,m} \\ O & O & \boldsymbol{A}_2 & \boldsymbol{B}_2 & \cdots & O & O \\ O & E_{1,2}^T & \boldsymbol{B}_2^T & \boldsymbol{D}_2 & \cdots & O & E_{2,m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ O & O & O & O & \cdots & \boldsymbol{A}_m & \boldsymbol{B}_m \\ O & E_{1,m}^T & O & E_{2,m}^T & \cdots & \boldsymbol{B}_m^T & \boldsymbol{D}_m \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_m \\ y_m \end{bmatrix} = \begin{bmatrix} f_1 \\ g_1 \\ f_2 \\ g_2 \\ \vdots \\ f_m \\ g_m \end{bmatrix} \quad (1)$$

where A_i, B_i, and D_i represent the connections within the i-th subdomain, and $E_{i,j}$ represents the interactions between the i-th and j-th subdomains.

DDM works by first eliminating internal nodes using the Schur complement, as in Eq. (2)-(4), reducing the problem to a smaller system involving only interface nodes, which is given by Eq. (5). This reduction makes the problem more suitable for parallel computation by decreasing both the size and complexity of the system. Once the interface nodes are solved, the internal nodes can then be calculated using Eq. (6).

$$\begin{bmatrix} P_i A_i P_i^T & P_i B_i \\ B_i^T P_i^T & D_i \end{bmatrix} = \begin{bmatrix} L_{i,11} & O \\ L_{i,21} & L_{i,22} \end{bmatrix} \begin{bmatrix} L_{i,11}^T & L_{i,21}^T \\ O & L_{i,22}^T \end{bmatrix} \quad (2)$$

$$\widetilde{D_i} = D_i - B_i^T A_i^{-1} B_i = L_{i,22} L_{i,22}^T \quad (3)$$

$$\widetilde{g_i} = g_i - B_i^T A_i^{-1} f_i = g_i - L_{i,21} L_{i,11}^{-1} P_i f_i \quad (4)$$

$$\begin{bmatrix} \widetilde{D_1} & E_{1,2} & \cdots & E_{1,m} \\ E_{1,2}^T & \widetilde{D_2} & \cdots & E_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ E_{1,m}^T & E_{2,m}^T & \cdots & \widetilde{D_m} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \widetilde{g_1} \\ \widetilde{g_2} \\ \vdots \\ \widetilde{g_m} \end{bmatrix} \quad (5)$$

$$A_i x_i = f_i - B_i y_i \quad (6)$$

B. Motivation

For conventional DDM, challenges remain in managing the cost of constructing and solving the dense global Schur complement and handling subdomain dependencies. In large-scale parallel computing, circuits are divided into small blocks, with the effects varying by partition count:

- **Fewer partitions:** Fewer partitions lead to longer local Schur complement calculations. *Larger sub-blocks require more time to compute*, limiting the advantages of parallel computing.
- **Too many partitions:** Numerous partitions can make top-level computation time significant, *potentially dominating the entire process*. The increased overhead for managing the global Schur complement reduces the benefits of parallel computing.

TABLE I: Breakdown time comparison for DDM solver.

Thread	2		4		8		16		32	
Case	T_{loc}	T_{top}	T_{loc}	T_{top}	T_{loc}	T_{top}	T_{loc}	T_{top}	T_{loc}	T_{top}
ibmpg8	10.5	0.01	4.03	0.40	2.53	1.65	1.88	3.62	1.38	2.95
thupg1	40.1	0.48	24.1	3.60	8.16	7.96	5.32	13.9	3.89	14.6
thupg2	82.6	0.87	53.1	6.90	25.3	15.7	12.5	22.3	7.95	25.6
thupg3	101	1.16	63.2	9.60	41.8	21.1	17.7	30.0	11.2	29.6
thupg4	154	2.11	91.2	11.3	51.2	25.9	23.5	46.1	16.5	55.7
thupg5	216	3.11	129	31.0	72.8	42.1	45.3	73.2	23.4	100

Table I shows the relationship between top-level computation time and the number of partitions. **With fewer partitions**, the time for local Schur complement calculations, T_{loc}, becomes too long. **With more partitions**, the top-level computation time, T_{top}, becomes dominant. This occurs because the global Schur complement size becomes too large and cannot be efficiently parallelized. Thus, traditional DDM has faced scalability challenges, often struggling to achieve speedup when the number of partitions increases beyond a certain point, as solving the dense global Schur complement tends to dominate the computation time. However, **modern hardware supports a significantly higher number of processes and threads, offering opportunities for enhanced parallelism.**

III. Proposed Method

This section presents our parallel computing framework for scalable power grid analysis. We first introduce a parallel method for computing the Schur complement, which is then integrated to form a nested DDM approach. This nested structure enhances scalability and computational efficiency, making it ideal for analyzing large-scale power grid systems.

A. Parallel Schur Complement Computation

Forming the Schur complement is closely related to Gaussian elimination. This process simplifies the network by eliminating certain nodes and reducing conductances to equivalent values at the selected ports. By applying Gaussian elimination to the conductance matrix, electrical connections between

nodes are combined into effective conductances at key ports, resulting in a simplified version of the system that preserves its essential electrical behavior.

The DDM also facilitates forming the Schur complement. In this approach, we first identify selected ports and non-port nodes, then apply a graph partitioning tool like METIS [15] to classify nodes. To achieve balanced computation, higher weights are assigned to selected ports, allowing an even distribution across blocks. Afterward, non-port nodes are further divided into internal and interface nodes within each subdomain. This categorization aids in the elimination process and organizes the system into a block-structured format, which is optimized for parallel computation.

Within each subdomain, internal nodes, along with their associated interface nodes and selected ports, are initially extracted to facilitate efficient parallel processing. Subsequently, the internal nodes are eliminated parallelly, producing a streamlined structure for further computations. Finally, interface nodes are removed, yielding the Schur complement corresponding to the selected ports. To clarify this process, Fig. 1 uses edge-based partitioning to depict the steps involved.

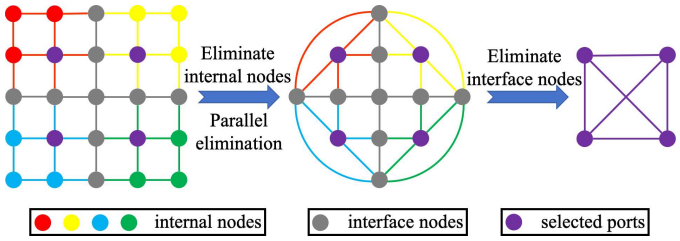


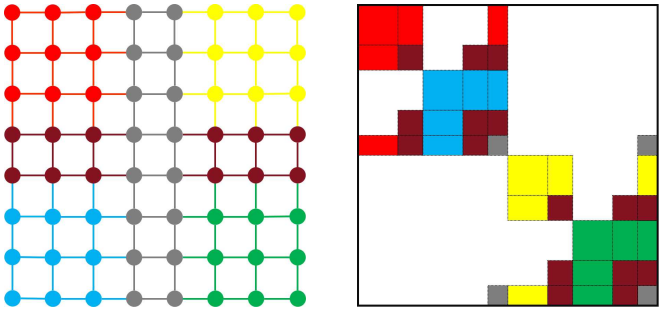
Fig. 1: Flow of parallel Schur complement computation.

This parallel Schur complement approach is not only effective for power grid analysis but can also be applied in scenarios where full-stack PDN simulation is unnecessary. For instance, in machine learning-based methods that predict supply noise [16]–[18]), selected ports can represent locations of worst-case noise. Our proposed parallel Schur complement method enables efficient computation of the noise at these critical locations.

B. Nested DDM for Power Grid Analysis

In conventional DDM for power grid analysis, the system is initially divided into m subdomains. As the number of subdomains increases (i.e., m becomes very large), the size of the global Schur complement matrix grows significantly, resulting in a dense matrix that is difficult to solve in parallel. To mitigate this, we apply a nested decomposition approach to the PDN, including the following steps.

Partitioning to construct nested DDM form matrix. The original matrix G can be transformed into the nested DDM form through a two-step graph partitioning process. First, the nodes corresponding to the entire matrix are partitioned to form the outer subdomains. Then the outer internal nodes within each outer subdomain are subdivided into inner interface nodes and inner internal nodes. This process classifies nodes into three types: *outer interface nodes* (e.g., gray nodes in Fig. 2), *inner interface nodes* (e.g., brown nodes in Fig. 2), and *inner internal nodes* (e.g., other colors in Fig. 2). After applying the corresponding permutation matrix, the transformed matrix takes the final form presented in Eq. (7).



(a) Graph (b) Matrix

Fig. 2: Graph partitioning to nested DDM: node colors in the graph (left) correspond to the sub-matrix block colors in the matrix (right).

For inner nodes, the matrices $A_{11}, \ldots, A_{pq}$ represent the inner internal nodes within the $p \times q$ inner subdomains, while $D_{11}, \ldots, D_{pq}$ correspond to the inner interface nodes. The matrix B_{ij} captures the interactions between the inner internal nodes and the inner interface nodes in the ij-th inner subdomain, while $E_{ij,st}$ describes the couplings between the inner interface nodes of the ij-th and st-th inner subdomains. The vectors f_{ij} and g_{ij} denote the right-hand side values associated with the inner internal nodes and inner interface nodes of the ij-th inner subdomain, respectively.

For outer interface nodes and their couplings, the matrix R_i represents the interactions among the outer interface nodes within the i-th outer subdomain. The matrix P_{ij} captures the coupling between the outer interface nodes of the i-th outer subdomain and the inner internal nodes of the ij-th inner subdomain, while Q_{ij} represents the coupling between the outer interface nodes of the i-th outer subdomain and the inner interface nodes of the ij-th inner subdomain. Additionally, $F_{i,j}$ characterizes the couplings between the outer interface nodes of the i-th and j-th outer subdomains. Finally, h_i refers to the right-hand side values associated with the outer interface nodes of the i-th outer subdomain.

Next step is to eliminate inner internal nodes.

1. Extract submatrices A_{ij}, B_{ij}, D_{ij}, P_{ij}, Q_{ij}, R_i, along with the right-hand sides f_{ij}, g_{ij}, and h_i parallelly for inner internal nodes, inner interface nodes, and related outer interface nodes, as shown in Eq. (7) within each inner subdomain.

2. Form matrices parallelly for inner interface nodes:

$$\widetilde{D_{ij}} = D_{ij} - B_{ij}^T A_{ij}^{-1} B_{ij} \tag{8}$$

3. Form matrices parallelly for outer interface nodes:

$$\widetilde{R_i} = R_i - \sum_{j=1}^{q} P_{ij}^T A_{ij}^{-1} P_{ij} \tag{9}$$

4. Form coupling matrices parallelly between inner and outer interface nodes:

$$\widetilde{Q_{ij}} = Q_{ij} - B_{ij}^T A_{ij}^{-1} P_{ij} \tag{10}$$

$$\begin{bmatrix} \boldsymbol{A_{11}} & \boldsymbol{B_{11}} & \cdots & O & O & P_{11} & \cdots & O & O & \cdots & O & O & O \\ \boldsymbol{B_{11}^T} & \boldsymbol{D_{11}} & \cdots & O & E_{11,1q} & Q_{11} & \cdots & O & O & \cdots & O & O & O \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ O & O & \cdots & \boldsymbol{A_{1q}} & \boldsymbol{B_{1q}} & P_{1q} & \cdots & O & O & \cdots & O & O & O \\ O & E_{11,1q}^T & \cdots & \boldsymbol{B_{1q}^T} & \boldsymbol{D_{1q}} & Q_{1q} & \cdots & O & O & \cdots & O & O & O \\ P_{11}^T & Q_{11}^T & \cdots & P_{1q}^T & Q_{1q}^T & R_1 & \cdots & O & O & \cdots & O & O & F_{1,p} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ O & O & \cdots & O & O & O & \cdots & \boldsymbol{A_{p1}} & \boldsymbol{B_{p1}} & \cdots & O & O & P_{p1} \\ O & O & \cdots & O & O & O & \cdots & \boldsymbol{B_{p1}^T} & \boldsymbol{D_{p1}} & \cdots & O & E_{p1,pq} & Q_{p1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ O & O & \cdots & O & O & O & \cdots & O & O & \cdots & \boldsymbol{A_{pq}} & \boldsymbol{B_{pq}} & P_{pq} \\ O & O & \cdots & O & O & O & \cdots & O & E_{p1,pq}^T & \cdots & \boldsymbol{B_{pq}^T} & \boldsymbol{D_{pq}} & Q_{pq} \\ O & O & \cdots & O & O & F_{1,p}^T & \cdots & P_{p1}^T & Q_{p1}^T & \cdots & P_{pq}^T & Q_{pq}^T & R_p \end{bmatrix} \begin{bmatrix} x_{11} \\ y_{11} \\ \vdots \\ x_{1q} \\ y_{1q} \\ z_1 \\ \vdots \\ x_{p1} \\ y_{p1} \\ \vdots \\ x_{pq} \\ y_{pq} \\ z_p \end{bmatrix} = \begin{bmatrix} f_{11} \\ g_{11} \\ \vdots \\ f_{1q} \\ g_{1q} \\ h_1 \\ \vdots \\ f_{p1} \\ g_{p1} \\ \vdots \\ f_{pq} \\ g_{pq} \\ h_p \end{bmatrix} \tag{7}$$

5. Form right-hand sides parallelly for inner interface nodes:

$$\widetilde{g_{ij}} = g_{ij} - B_{ij}^T A_{ij}^{-1} f_{ij} \tag{11}$$

6. Form right-hand sides parallelly for outer interface nodes:

$$\widetilde{h_i} = h_i - \sum_{j=1}^{q} P_{ij}^T A_{ij}^{-1} f_{ij} \tag{12}$$

Therefore the intermidiate Schur complement is formed. Eq. (7) is condensed by removing the inner internal nodes through Gaussian elimination into a compact form, resulting in the intermediate Schur complement matrix, ready for further computations:

$$\begin{bmatrix} \widetilde{D_{11}} & \cdots & E_{11,1q} & \widetilde{Q_{11}} & \cdots & O & \cdots & O & O \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ E_{11,1q}^T & \cdots & \widetilde{D_{1q}} & \widetilde{Q_{1q}} & \cdots & O & \cdots & O & O \\ \widetilde{Q_{11}}^T & \cdots & \widetilde{Q_{1q}}^T & \widetilde{R_1} & \cdots & O & \cdots & O & F_{1,p} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ O & \cdots & O & O & \cdots & \widetilde{D_{p1}} & \cdots & E_{p1,pq} & \widetilde{Q_{p1}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ O & \cdots & O & O & \cdots & E_{p1,pq}^T & \cdots & \widetilde{D_{pq}} & \widetilde{Q_{pq}} \\ O & \cdots & O & F_{1,p}^T & \cdots & \widetilde{Q_{p1}}^T & \cdots & \widetilde{Q_{pq}}^T & \widetilde{R_p} \end{bmatrix} \begin{bmatrix} y_{11} \\ \vdots \\ y_{1q} \\ z_1 \\ \vdots \\ y_{p1} \\ \vdots \\ y_{pq} \\ z_p \end{bmatrix} = \begin{bmatrix} \widetilde{g_{11}} \\ \vdots \\ \widetilde{g_{1q}} \\ \widetilde{h_1} \\ \vdots \\ \widetilde{g_{p1}} \\ \vdots \\ \widetilde{g_{pq}} \\ \widetilde{h_p} \end{bmatrix} \tag{13}$$

Then, eliminate inner inerface nodes.

1. Extract submatrices $\widetilde{U_i}$, $\widetilde{V_i}$ and right-hand sides $\widetilde{g_i}$, $\widetilde{h_i}$ parallelly for inner and outer interface nodes in Eq. (13) within each outer subdomain:

$$\widetilde{U_i} = \begin{bmatrix} \widetilde{D_{i1}} & E_{i1,i2} & \cdots & E_{i1,iq} \\ E_{i1,i2}^T & \widetilde{D_{i2}} & \cdots & E_{i2,iq} \\ \vdots & \vdots & \ddots & \vdots \\ E_{i1,iq}^T & E_{i2,iq}^T & \cdots & \widetilde{D_{iq}} \end{bmatrix} \tag{14}$$

$$\widetilde{V_i} = \begin{bmatrix} \widetilde{Q_{i1}^T} & \widetilde{Q_{i2}^T} & \cdots & \widetilde{Q_{iq}^T} \end{bmatrix}^T, \quad \widetilde{g_i} = \begin{bmatrix} \widetilde{g_{i1}^T} & \widetilde{g_{i2}^T} & \cdots & \widetilde{g_{iq}^T} \end{bmatrix}^T \tag{15}$$

2. Form matrices parallelly for outer interface nodes:

$$\widetilde{\widetilde{R_i}} = \widetilde{R_i} - \widetilde{V_i}^T \widetilde{U_i}^{-1} \widetilde{V_i} \tag{16}$$

3. Form right-hand sides parallelly for outer interface nodes:

$$\widetilde{\widetilde{h_i}} = \widetilde{h_i} - \widetilde{V_i}^T \widetilde{U_i}^{-1} \widetilde{g_i} \tag{17}$$

And form the global Schur complement. The process of eliminating the inner internal and interface nodes employs the proposed parallel Schur complement computation method, ultimately resulting in the Schur complement for the selected ports (i.e., the outer interface nodes):

$$\begin{bmatrix} \widetilde{\widetilde{R_1}} & F_{1,2} & \cdots & F_{1,p} \\ F_{1,2}^T & \widetilde{\widetilde{R_2}} & \cdots & F_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ F_{1,p}^T & F_{2,p}^T & \cdots & \widetilde{\widetilde{R_p}} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_p \end{bmatrix} = \begin{bmatrix} \widetilde{\widetilde{h_1}} \\ \widetilde{\widetilde{h_2}} \\ \vdots \\ \widetilde{\widetilde{h_p}} \end{bmatrix} \tag{18}$$

Last step is solution process. The solution to Eq. (7) begins with Eq. (18) to determine the outer interface unknowns z_i. Next, the subsequent equations are used to find the inner interface unknowns y_{ij}, and finally, solving the second equation below yields the inner internal unknowns x_{ij}:

$$\begin{bmatrix} \widetilde{D_{i1}} & E_{i1,i2} & \cdots & E_{i1,iq} \\ E_{i1,i2}^T & \widetilde{D_{i2}} & \cdots & E_{i2,iq} \\ \vdots & \vdots & \ddots & \vdots \\ E_{i1,iq}^T & E_{i2,iq}^T & \cdots & \widetilde{D_{iq}} \end{bmatrix} \begin{bmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{iq} \end{bmatrix} = \begin{bmatrix} \widetilde{g_{i1}} - \widetilde{Q_{i1}} z_i \\ \widetilde{g_{i2}} - \widetilde{Q_{i2}} z_i \\ \vdots \\ \widetilde{g_{iq}} - \widetilde{Q_{iq}} z_i \end{bmatrix} \tag{19}$$

$$A_{ij} x_{ij} = f_{ij} - B_{ij} y_{ij} - P_{ij} z_i \tag{20}$$

Combining the above derivations, the Laplacian matrix $Gv = b$ is solved with nested DDM as in Algorithm 1.

C. The Overall Framework

Building on the optimizations outlined above, the overall framework of our proposed solver is presented in Algorithm 1, which consists of three stages:

- First, the PDN is divided into p outer subdomains, and then each outer subdomain is further subdivided into q inner subdomains, creating local grids labeled as $11 \sim pq$, where $m = p \times q$. This hierarchical strategy significantly reduces the size of the top-level problem by introducing an intermediate Schur complement, which, while still dense, is more suitable for parallel computation.

- Next, the local grids $i1 \sim iq$ within each outer subdomain i are used to construct the middle grid i through the computation of the Schur complement, which consolidates the conductance of most nodes onto specific nodes. The p middle grids are then combined through further node elimination to form the top grid. The solution process starts with solving the top grid, and its solution is then used to solve the p middle grids.
- Finally, the solutions of the middle grids are back-substituted into their respective local grid, as illustrated in Fig. 3.

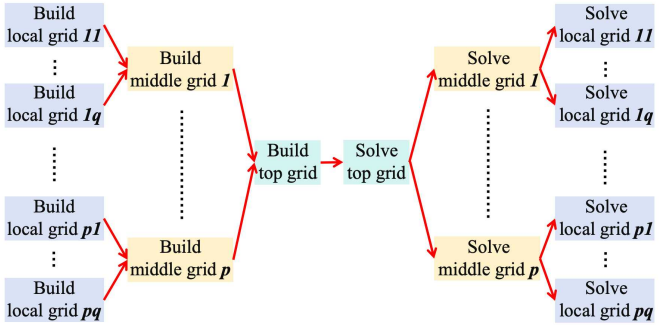


Fig. 3: Flow of the proposed nested DDM approach.

Efficient Elimination. In the local layer, instead of solving Eq. (8)-(12) individually, we concatenate A_{ij}, B_{ij}, D_{ij}, P_{ij}, Q_{ij}, and R_i into a unified matrix. This approach enables only a single factorization to yield the result, as shown in Eq. (2). To eliminate submatrix A_{ij}, techniques such as matrix reordering can be employed to reorganize the A_{ij}, thereby reducing fill-ins during factorization. Matrix reordering techniques can also be used in the middle layer when solving Eq. (16)-(17).

Maximizing Multithreading Efficiency. For the inner subdomains, each thread processes an individual submatrix. In the intermediate layer, only p tasks can be evidently parallelized, as described in Eq. (16)–(17) and (19). Therefore, q threads are used to solve each task. For the top layer, however, there are no obvious parallelization opportunities, as there is only one task, as indicated in Eq. (18). In this case, $p \times q$ threads are used to solve the single task.

IV. Experimental Results

Our proposed framework, implemented in C++ and compiled with GCC 9.4.0 using the -O3 optimization flag, utilizes METIS [15] for graph partitioning and CHOLMOD direct solver [5] with Intel MKL [19] for matrix factorization. We test on the IBMPG [20] and THUPG [21] benchmarks and compare our method with three state-of-the-art solvers, the CHOLMOD-based serial solver, the CKTSO parallel solver [6], and the conventional DDM parallel solver. Although CHOLMOD supports multithreading, our experiments show no significant performance improvement in the parallel version, so we compare the serial version's execution time. All experiments are conducted on a Linux server with two Intel Xeon Silver 4210R CPUs and 128GB of RAM.

Algorithm 1 Proposed parallel solver with nested DDM

Input: Initial matrix G, number of outer subdomains p, number of inner subdomains q.
Output: Solution x_{ij}, y_{ij}, and z_i of Eq. (7).

```
Perform initial graph partitioning on G to generate p outer subdomains, resulting in outer interface nodes and outer internal nodes.
for each of the p outer subdomains in parallel do
    Partition the outer internal nodes into q inner subdomains, resulting in inner interface nodes and inner internal nodes.
end for
for each of the p × q inner subdomains in parallel do
    Build the local grid using inner internal nodes, inner interface nodes, and corresponding outer interface nodes.
    Eliminate inner internal nodes via Eq. (8)-(12).
end for
for each of the p outer subdomains in parallel do
    Build the middle grid, resulting in Eq. (13).
    Eliminate inner interface nodes using Eq. (16)-(17).
end for
Build the global grid, resulting in Eq. (18).
Solve the global grid for z_i using Eq. (18).
for each of the p outer subdomains in parallel do
    Solve the middle grid for y_ij using Eq. (19).
end for
for each of the p × q inner subdomains in parallel do
    Solve the local grid for x_ij using Eq. (20).
end for
```

A. Results on DC Analysis of Power Grids

Table II provides the comparison results between the proposed nested DDM parallel solver and three other solvers. For DDM, the circuit matrix is divided into $p \times q$ subdomains, while nested DDM divides it into p outer subdomains, each with q inner subdomains. Optimal performance is observed with the divisions 2×2, 2×4, 2×8, and 4×8, which are used in Table II. Here, T_{loc}, T_{mid}, T_{top} and T represent the computation time for inner subdomains, intermediate Schur complement, global Schur complement and overall execution, respectively. T_{mt} is the total time for the intermediate and global Schur complements. The speedup ratios Sp_1, Sp_2, and Sp_3 indicate performance improvements over the CHOLMOD, CKTSO, and conventional DDM parallel solver, respectively.

It can be seen from Table II that our framework achieves an average 8.36× speedup compared to the CHOLMOD-

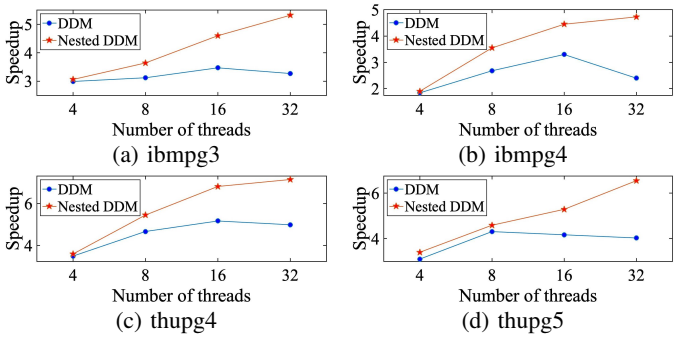


Fig. 4: Speedup comparison of DDM and nested DDM parallel solver over CHOLMOD-based serial solver.

TABLE II: Runtime Comparison of the Proposed Nested DDM Solver with CHOLMOD-based Serial Solver [5], CKTSO Parallel Solver [6], and Conventional DDM Solver.

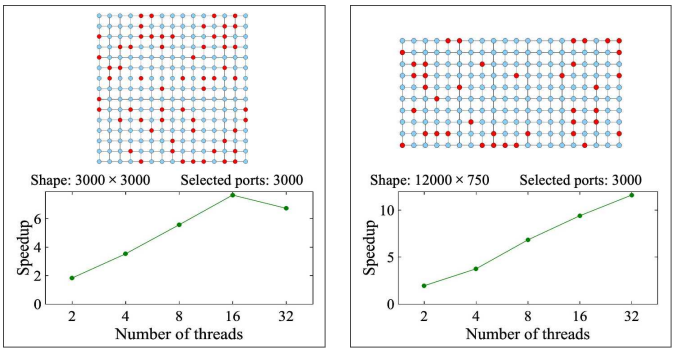
Case	N	Serial	CKTSO	DDM			Proposed nested DDM					$Sp1$	$Sp2$	$Sp3$
		T	T	T_{loc}	T_{top}	T	T_{loc}	T_{mid}	T_{top}	T_{mt}	T			
ibmpg8	4	22.7	28.1	4.03	0.40	4.43	3.89	0.22	0.01	0.23	4.12	5.51×	6.82×	1.08×
	8		22.6	2.53	1.65	4.18	2.47	0.65	0.01	0.66	3.13	7.25×	7.22×	1.34×
	16		21.9	1.88	3.62	5.50	1.80	1.46	0.03	1.49	3.29	6.90×	6.66×	1.67×
	32		23.2	1.38	2.95	4.33	1.45	0.80	0.36	1.16	2.61	8.70×	8.89×	1.66×
thupg1	4	101	148	24.1	3.60	27.7	23.9	2.23	0.44	2.67	26.6	3.80×	5.56×	1.04×
	8		132	8.16	7.96	16.1	8.56	4.41	0.45	4.86	13.4	7.54×	9.85×	1.20×
	16		124	5.32	13.9	19.2	5.32	6.88	0.37	7.25	12.6	8.02×	9.84×	1.52×
	32		118	3.89	14.6	18.5	3.70	2.96	2.76	5.72	9.42	10.7×	12.5×	1.96×
thupg2	4	224	348	53.1	6.90	60.0	52.7	4.76	0.79	5.55	58.3	3.84×	5.97×	1.03×
	8		279	25.3	15.7	41.0	25.7	8.66	1.22	9.88	35.6	6.29×	7.84×	1.15×
	16		260	12.5	22.3	34.8	12.9	13.5	0.70	14.2	27.1	8.27×	9.59×	1.28×
	32		251	7.95	25.6	33.6	8.10	11.6	5.79	17.4	25.5	8.78×	9.84×	1.32×
thupg3	4	246	460	63.2	9.60	72.8	61.7	6.38	0.98	7.35	69.1	3.56×	6.66×	1.05×
	8		355	41.8	21.1	62.9	39.9	13.8	0.80	14.6	54.5	4.51×	6.51×	1.15×
	16		337	17.7	30.0	47.7	17.7	18.0	0.81	18.8	36.5	6.74×	9.23×	1.31×
	32		335	11.2	29.6	40.8	11.1	13.4	5.26	18.7	29.8	8.26×	11.2×	1.37×
thupg4	4	360	687	91.2	11.3	103	88.6	9.82	2.02	11.8	100	3.60×	6.87×	1.03×
	8		563	51.2	25.9	77.1	51.1	13.5	1.47	15.0	66.1	5.45×	8.52×	1.17×
	16		525	23.5	46.1	69.6	22.8	28.3	1.67	30.0	52.8	6.81×	9.94×	1.32×
	32		484	16.5	55.7	72.2	16.6	22.6	11.2	33.8	50.4	7.14×	9.60×	1.43×
thupg5	4	496	1100	129	31.0	160	128	15.1	2.60	17.7	146	3.40×	7.53×	1.10×
	8		861	72.8	42.1	115	72.3	33.0	2.66	35.7	108	4.59×	7.97×	1.06×
	16		776	45.3	73.2	119	46.9	44.8	2.00	46.8	93.7	5.29×	8.28×	1.27×
	32		727	23.4	100	123	22.8	33.7	19.2	52.9	75.7	6.55×	9.60×	1.62×
Average	-	-	-	-	-	-	-	-	-	-	-	-	**8.44**×	**1.30**×

based serial solver utilizing 32 threads, due to multi-level domain decomposition that enhances parallelism by breaking the problem into smaller independent subproblems. Compared to the CKTSO parallel solver, it achieves an average speedup of 8.44× in total runtime, owing to the more structured and sparsely connected nature of power grids. In comparison to the DDM parallel solver, our approach achieves a 1.70× speedup in solving the global Schur complement. This is due to the introduction of an intermediate Schur complement, which enables additional parallelism by redistributing top-level task to the intermediate layer in nested DDM. The improvement is evident when comparing T_{top} in DDM with T_{mt} in nested DDM. As a result, our method provides a 1.30× total runtime speedup over the DDM parallel solver.

To highlight the superior scalability of our framework compared to conventional DDM, we compare the two parallel solvers. As shown in Fig. 4, the nested DDM approach exhibits significantly better scalability with an increasing number of threads, while DDM's performance plateaus early due to inefficiencies in handling the large, dense global Schur complement.

B. Evaluation of parallel Schur complement computation

We further demonstrate the effectiveness of our proposed parallel method for computing the Schur complement using two power grid cases. In these examples, the red nodes represent the selected ports predicted to experience the highest worst-case noise, for which the Schur complement must be computed. The remaining nodes are eliminated to produce an equivalent conductance matrix for the selected ports (i.e., the key points we need to focus on). As shown in Fig. 5, our partition-based parallel computation method achieves significant performance improvements, with speedups of 6.73× and 11.6× when using 32 threads. The method is applicable to PDNs with various grid shapes, and its parallel efficiency depends on the initial partitioning. Fewer interface nodes (e.g., 750 × 31 in Case 2 compared to 3000 × 10 in Case 1 for 32 threads) generally result in higher parallel efficiency.



(a) Case 1 (b) Case 2

Fig. 5: Comparison of speedup in DDM for Schur complement computation across different grid shapes.

V. Conclusion

In this paper, we have presented an efficient parallel nested domain decomposition solver for power grid analysis, aimed at enhancing the scalability of the traditional DDM parallel solver. By introducing a parallel method for computing the Schur complement and integrating it with conventional DDM, we effectively addresses the challenges of managing the large, dense global Schur complement and the computational complexity inherent in large-scale PDNs. Our results demonstrate that the proposed method not only accelerates the solution process but also optimizes resource utilization, making it highly suitable for modern IC design.

Acknowledgement

This work was supported by NSFC (Grant No. 62034007), Zhejiang Provincial NSF (Grant No. D24F040002), NSFC (Grant No. 62204265, 92473107, U23A20301 and 62234010).

REFERENCES

[1] E. Chiprout, "On-die power grids: The missing link," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 940–945.

[2] C. Zhuo, G. Wilke, R. Chakraborty, A. A. Aydiner, S. Chakravarty, and W.-K. Shih, "Silicon-validated power delivery modeling and analysis on a 32-nm ddr i/o interface," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1760–1771, 2014.

[3] C. Zhuo, K. Unda, Y. Shi, and W.-K. Shih, "From layout to system: Early stage power delivery and architecture co-exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1291–1304, 2018.

[4] C. Wen, Y. Cai, and C. Zhuo, "Impact of supply noise on nano-meter vlsi design: Hard or soft threshold?" in *2021 China Semiconductor Technology International Conference (CSTIC)*. IEEE, 2021, pp. 1–4.

[5] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate," *ACM Transactions on Mathematical Software (TOMS)*, vol. 35, no. 3, pp. 1–14, 2008.

[6] X. Chen, "Numerically-stable and highly-scalable parallel lu factorization for circuit simulation," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[7] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

[8] C. Zhuo, J. Hu, M. Zhao, and K. Chen, "Power grid analysis and optimization using algebraic multigrid," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 738–751, 2008.

[9] Z. Feng, "Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[10] Z. Liu, W. Yu, and Z. Feng, "fegrass: Fast and effective graph spectral sparsification for scalable power grid analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 681–694, 2021.

[11] Q. Zhou, K. Sun, K. Mohanram, and D. C. Sorensen, "Large power grid analysis using domain decomposition," in *Proceedings of the Design Automation & Test in Europe Conference*, vol. 1. IEEE, 2006, pp. 1–6.

[12] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 54–59.

[13] Z. Liu and W. Yu, "pgrass-solver: A graph spectral sparsification-based parallel iterative solver for large-scale power grid analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 9, pp. 3031–3044, 2023.

[14] H. Zhou, D. Yang, Y. Lin, Y. Dai, and Q. Chen, "A parallel acceleration technique based on bordered block diagonal matrix reordering for exponential integrator method," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2024, pp. 94–99.

[15] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[16] C.-H. Pao, A.-Y. Su, and Y.-M. Lee, "Xgbir: An xgboost-based ir drop predictor for power delivery network," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1307–1310.

[17] V. A. Chhabria, V. Ahuja, A. Prabhu, N. Patil, P. Jain, and S. S. Sapatnekar, "Thermal and ir drop analysis using convolutional encoder-decoder networks," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 690–696.

[18] X. Dong, Y. Chen, X. Yin, and C. Zhuo, "Worst-case dynamic power distribution network noise prediction using convolutional neural network," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1225–1230.

[19] I. Corporation. (2024) Intel oneapi math kernel library. Online. Accessed: 2024. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html

[20] S. R. Nassif. (2020) Ibm power grid benchmarks. Online. Accessed: 2024. [Online]. Available: https://web.ece.ucsb.edu/lip/PGBenchmarks/ibmpgbench.html

[21] J. Yang and Z. Li. (2012) Thu power grid benchmarks. Online. Accessed: 2024. [Online]. Available: http://tiger.cs.tsinghua.edu.cn/PGBench/

AutoPower: Automated Few-Shot Architecture-Level Power Modeling by Power Group Decoupling

Qijun Zhang, Yao Lu, Mengming Li, Zhiyao Xie*
Hong Kong University of Science and Technology
{qzhangcs, yludf, mengming.li}@connect.ust.hk, eezhiyao@ust.hk

***Abstract*—Power efficiency is a critical design objective in modern CPU design. Architects need a fast yet accurate architecture-level power evaluation tool to perform early-stage power estimation. However, traditional analytical architecture-level power models are inaccurate. The recently proposed machine learning (ML)-based architecture-level power model requires sufficient data from known configurations for training, making it unrealistic.**

In this work, we propose AutoPower targeting fully automated architecture-level power modeling with limited known design configurations. We have two key observations: (1) The clock and SRAM dominate the power consumption of the processor, and (2) The clock and SRAM power correlate with structural information available at the architecture level. Based on these two observations, we propose the power group decoupling in AutoPower. First, AutoPower decouples across power groups to build individual power models for each group. Second, AutoPower designs power models by further decoupling the model into multiple sub-models within each power group. In our experiments, AutoPower can achieve a low mean absolute percentage error (MAPE) of 4.36% and a high R^2 of 0.96 even with only two known configurations for training. This is 5% lower in MAPE and 0.09 higher in R^2 compared with McPAT-Calib, the representative ML-based power model.

I. Introduction

Power efficiency is a critical design objective in modern CPU design, and its optimization relies on efficient power evaluation. However, as the complexity of processors keeps scaling up, standard VLSI power evaluation flow requires increasing time. The standard power evaluation flow goes through RTL implementation, RTL simulation, logic synthesis, and power simulation [1]. Completing such a power evaluation flow can take weeks, seriously hindering early microarchitecture optimization and design space exploration. Architects need a fast yet accurate architecture-level power evaluation to support the early optimization of CPU microarchitecture.

Analytical Power Model: Classical architecture-level power models such as McPAT [2] and Wattch [3] are analytical models [4] designed by engineers for a specific type of processor[1]. However, as processor architecture keeps updating, the analytical power model can hardly reflect the design of new processors. As mentioned in many studies [4]–[6], directly applying McPAT on the new generation of CPUs incurs unacceptable errors [4]. Therefore, it is usually necessary to modify McPAT for each target architecture, requiring huge engineering effort and time overhead.

Machine learning (ML)-based Power Models: In recent years, ML-based architecture-level power models [6], [7] are trained to predict the CPU power consumption based on hardware parameters and event parameters. However, training an accurate ML-based model *requires a large number of samples with golden power labels*. In practice, it is time-consuming to go through a standard power evaluation flow to collect golden power labels. More importantly, the number of available configurations is also limited. Therefore, these data-hungry ML-based models are not applicable in real engineering scenarios. Some recent ML works [4] proposed to reduce required training data by unifying analytical models and ML-based models. However, [4] relies on analytical resource functions, which are design-dependent and heavily based on architect expertise. Therefore, there is still a lack of a fully automatic power modeling method that only requires limited known design configurations for training.

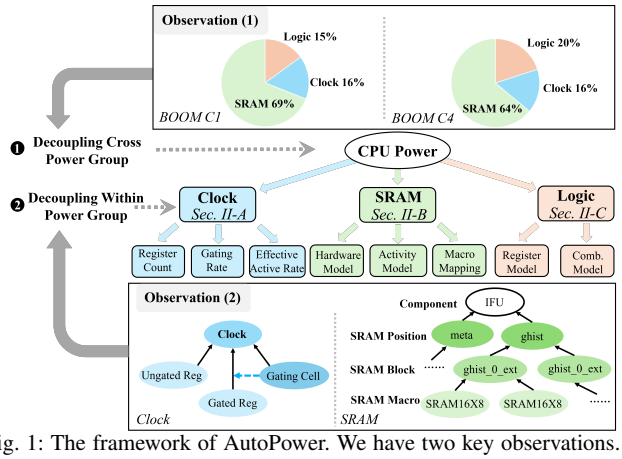


Fig. 1: The framework of AutoPower. We have two key observations. (1) Clock and SRAM dominate the power consumption. (2) Clock and SRAM power correlate with structural information available at the architecture level. Based on these observations, AutoPower decouples the power modeling across different power groups and further decouples the power model for each power group into multiple sub-models.

Goal: This work proposes our new solution named AutoPower[2], achieving both *fully automatic* and *few-shot* architecture-level power modeling, 1) *Fully automatic*: The power modeling method does not require any engineer-defined design-specific analytical power model. 2) *Few-shot*: The model can achieve high accuracy with very limited known configurations for training. As illustrated in Fig. 1. Our solution is initially based on two key observations:

- **Observation 1: Clock and SRAM dominate total power consumption.** The clock and SRAM dominate the power consumption of the processor. Observation 1 in Fig. 1 shows power percentage of each power group of RISC-V BOOM CPU [8] measured at layout stage. It demonstrates that the clock and SRAM dominate the power consumption of the processor.
- **Observation 2: Clock and SRAM power correlate with structural information available at the architecture level.**. (1) Clock power mainly depends on clock pins in registers and clock gating information. The number of registers and the percentage of registers with clock gating are predictable even at the architecture level. (2) SRAM in processors usually follows a four-level hierarchy from top to bottom: component, SRAM Position, SRAM Block, and SRAM Macro. Many design components have some SRAM Positions that will be filled

*Corresponding Author

[1]Engineers carefully analyze the hardware behavior of each component when relevant events occur to infer the activity of the hardware circuit, and then utilize their own experience and experimental data to estimate the power consumption of the component when relevant events occur.

[2]It is open-sourced at https://github.com/hkust-zhiyao/AutoPower

by SRAM-dominated structures (e.g., tables and queues). Each SRAM Position consists of multiple SRAM Blocks that will be defined in RTL implementations. SRAM Block is built up with SRAM Macros supported by technology node library (i.e., SRAM compiler).

Power Group Decoupling in AutoPower: (1) Motivated by our 1st observation, AutoPower decouples the power models across power groups, building individual power modeling methods for different power groups including clock, SRAM, and logic. (2) Motivated by our 2nd observation, within each power group, AutoPower further decouples the modeling into multiple sub-models based on our observed unique pattern of each power group.

AutoPower adopts innovative power modeling methods for each power group as shown in Fig. 1. (1) For the clock, AutoPower is the first architectural power model that explicitly considers *clock gating*, an indispensable optimization that significantly reduces clock power. Based on the structure of clock power, it proposes to decouple the clock power modeling into the prediction of register count, gating rate, and our pre-defined effective active rate. (2) For SRAM, AutoPower estimates the power based on our proposed hierarchy with a top-down approach. AutoPower transfers the feature from components to SRAM Position, then applies the hardware model and activity model to estimate hardware information and read/write frequency at the SRAM Block level, and finally combines SRAM Macro hardware information and read/write frequency with macro-level mapping. (3) For the logic power, AutoPower decouples it into register power (without clock pins) and combinational logic power, with different modeling methods.

Contributions: Our contributions are summarized below.

- We propose a new architecture-level power modeling framework, AutoPower, that decouples the power model across power groups and within each power group. It requires no design-specific engineer expertise and achieves high accuracy with only a few training samples (i.e., few-shot learning).
- AutoPower proposes innovative power models for each power group: 1) For clock power, AutoPower considers the clock gating technology and decouples the model based on the structure of clock power. 2) For SRAM power, AutoPower estimates the SRAM Macro information and read/write frequency based on our proposed four-level hierarchy. 3) For logic power, AutoPower adopts decoupled power for combinational logic and register power respectively.
- We evaluate AutoPower based on 15 out-of-order RISC-V CPUs with different configurations and 8 workloads. AutoPower achieves a low MAPE of 4.36% and a high R^2 of 0.96 on average only using two known configurations for training. It achieves 5% lower MAPE and 0.09 higher R^2 compared with representative prior works.

II. Methodology

AutoPower decouples the modeling into different power groups including clock power, SRAM power, and logic power. Within each power group, AutoPower further decouples the power models into multiple simple sub-models based on the specific structural information of each power group.

As an architecture-level power model, AutoPower only takes hardware parameters and event parameters as input to calculate the power consumption of the processor. Hardware parameters denoted as H are the parameters to determine the processor configurations, such as DecodeWidth and ICacheWay. Event parameters denoted as E are the information collected from architecture-level performance simulators, by simulating a workload with a certain CPU configuration, for example, the number of cache misses and branch mispredictions.

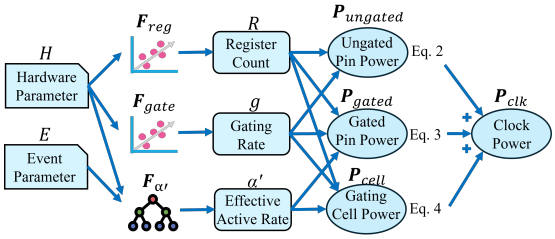


Fig. 2: The clock power includes the power of clock pins in the ungated register (Ungated Pin Power), the power of clock pins in the gated register (Gated Pin Power), and the power of clock gating cells (Gating Cell Power). These three parts can be modeled with the register count, gating rate, and effective active rate, which are estimated by ML models.

We introduce our model for clock power (Sec. II-A), SRAM power (Sec. II-B), and remaining logic power (Sec. II-C) in this section.

A. Power Model for Clock

Clock consumes a significant percentage of power, as shown in Fig. 1. We devote a customized power modeling method for the clock-related power. Such clock-related power is dominated by the internal power of the clock pins in registers. Clock gating is one of the most effective techniques for optimizing clock power, utilizing the clock gating cells to gate the clock of the register. Therefore, AutoPower takes the effect of clock gating into account.

Fig. 2 illustrates our clock power model, which decouples the clock power into the power of clock pins in the ungated register (Ungated Pin Power), the power of clock pins in the gated register (Gated Pin Power), and the power of clock gating cells (Gating Cell Power). To accurately model these three parts of the clock power, we further decouple them into formulations represented by register count, gating rate, and effective active rate. Therefore, ML models only require estimating these three values, which have simpler correlations with hardware parameters and event parameters.

Formulation of Clock Power: The clock power of a processor with the clock gating technique consists of the register clock pin power and the power of clock gating cells. The register clock pin power includes the power of clock pins in the ungated register (ungated pin power) and the power of clock pins in the gated register (gated pin power). Denoting the clock power as P_{clk}, ungated pin power as P_{ungated}, gated pin power as P_{gated}, and the power of clock gating cell as P_{cell}, the clock power can be formulated below,

$$P_{\text{clk}} = P_{\text{ungated}} + P_{\text{gated}} + P_{\text{cell}} \tag{1}$$

We further decouple the power of each part. To formulate it, we denote the number of registers as R, the ratio between registers and clock gating cells as r, the percentage of gated registers as g, the clock pin power per register with active clock as p_{reg}, the pin power of latch in clock gating cell as p_{latch}, and the average active rate across all gated registers as α. We formulate P_{ungated}, P_{gated}, and P_{cell} below,

$$P_{\text{ungated}} = R * (1 - g) * p_{\text{reg}} \tag{2}$$

$$P_{\text{gated}} = \alpha * R * g * p_{\text{reg}} \tag{3}$$

$$P_{\text{cell}} = r * R * g * p_{\text{latch}} \tag{4}$$

(1) For the ungated pin power P_{ungated} in Eq. 2, the clock pin keeps active for every cycle, regardless of the workload that is executed on the processor. Therefore, the power is the multiplication between the number of ungated registers $R * (1 - g)$ and the clock pin power per register with active clock p_{reg}. (2) For the gated pin power in Eq. 3, because of the clock gating, the clock pin can be

inactive for some cycles. Therefore, the active rate α should be considered. (3) For the power of the clock gating cell in Eq. 4, it is difficult to estimate all of these factors accurately at the architecture level. However, in processors, the number of clock gating cells is approximately proportional to the number of gated registers. Besides, the negligible effect of the toggling of its control signal and the number of registers connected with its output clock can be ignored under the architecture-level power modeling scenarios. Therefore, the power can be formulated as Eq. 4.

Putting them together, Eq. 1 can be expressed as below,

$$\begin{aligned} P_{\text{clk}} =& R\,(1-g)\,p_{\text{reg}} + \alpha R\,g\,p_{\text{reg}} + r\,R\,g\,p_{\text{latch}} \\ =& R\,(1-g)\,p_{\text{reg}} + \alpha\,R\,g\,p_{\text{reg}}\,(1 + r\,p_{\text{latch}}\,/\,p_{\text{reg}}) \end{aligned} \tag{5}$$

The reason why we do not directly estimate the number of ungated registers and the number of gated registers is that the clock gating is performed with the logic synthesis tools, which is difficult to predict at the architecture level. In contrast, estimating the total number of registers R and the gating rate g can be easier.

To facilitate the ML-based prediction, we further simplify $\alpha * (1 + rp_{\text{latch}}/p_{\text{reg}})$ as an effective active rate α',

$$\alpha' = \alpha * (1 + r\,p_{\text{latch}}\,/\,p_{\text{reg}}) \tag{6}$$

then Eq. 5 can be finalized as below,

$$P_{\text{clk}} = R\,(1-g)\,p_{\text{reg}} + \alpha' R\,g \tag{7}$$

In Eq. 7, P_{reg} can be looked up from the library file of the technology node adopted for the VLSI flow. Therefore, we totally require three prediction models. (1) register count prediction model $\boldsymbol{F}_{\text{reg}}$ to predict R, (2) gating rate prediction model $\boldsymbol{F}_{\text{gate}}$ to predict g, and (3) effective active rate prediction model $\boldsymbol{F}_{\boldsymbol{\alpha'}}$ to predict α'.

Prediction Model: We formulate the three prediction models below,

$$R = \boldsymbol{F}_{\text{reg}}(H), \quad g = \boldsymbol{F}_{\text{gate}}(H), \quad \alpha' = \boldsymbol{F}_{\boldsymbol{\alpha'}}(H, E) \tag{8}$$

Register count and gating rate of a processor are determined after logic synthesis regardless of workload executed. Therefore, register count prediction model and gating rate prediction model only take hardware parameters of the component as input features. We adopt ML models to learn the correlation. Because the correlation is usually simple and we do not have sufficient samples for training, we adopt the linear model with L2 normalization as our ML model. To collect labels for training, we can collect the number of registers and the number of gated registers from the netlists of known configurations.

Different from register count and gating rate, the active rate α depends on both configuration and workload executed, which is the same case for the effective active rate α'. Therefore, the effective active rate prediction model takes both the hardware parameters and the event parameters as input features. We also adopt the ML model to learn the correlation. In contrast to the register count and gating rate prediction where the correlation is relatively simple, this correlation can be relatively complex. Besides, samples on known configurations for training can be more than the register count and gating rate because we can get multiple samples on a configuration by executing different workloads. Therefore, we adopt a relatively advanced ML model, XGBoost [9], to capture the complex correlation.

B. Power Model for SRAM

SRAM plays an important role in modern processors as the backbone of several tables, such as the history table in branch predictor and the metadata table in IFU, and caches. Its high power consumption is caused by expensive SRAM access. To estimate the SRAM power, we propose to explore the general hierarchy of SRAM in modern CPUs. Our model targets to estimate the SRAM power based on the SRAM Macro information, including the type, number, and read/write frequency. Then the SRAM power can be accurately estimated with the technology node library. Our SRAM power model mainly consists of (1) a scaling-pattern-based hardware model to estimate the number and shape of SRAM Blocks, (2) an activity model to predict the read/write frequency of SRAM Block, and (3) a macro-level mapping model to map the SRAM Block level information to SRAM Macro level.

SRAM Hierarchy: As shown in Fig. 3(a), we propose to model the SRAM power following our proposed general four-layer hierarchy with a top-down approach:

Component → SRAM Position → SRAM Block → SRAM Macro

SRAM Position: At the architecture level, some components include one or more *SRAM Position* that are built from SRAM and serve different functionalities. For example, in IFU, the Fetch Target Queue has two SRAM Positions: 1) ghist, the SRAM structure for the queue; 2) meta, the SRAM structure to store some other metadata.

SRAM Block: At the RTL design level, to build up an SRAM Position, one or multiple identical *SRAM Block* will be used to implement a single- or multi-bank structure. SRAM Block is a logical SRAM with the shape and type required for building SRAM Position. It is the lowest-level layer that is visible to microarchitecture and independent of the support of the adopted technology node library. For example, as shown in Fig. 3(a), the *ghist* consists of multiple SRAM Blocks called ghist_0_ext.

SRAM Macro: At the VLSI implementation level, *SRAM Macro* is the macro supported by the adopted technology node library. Memory compiler in the technology node library cannot generate memory macros with arbitrary shapes. Therefore, the SRAM Block with an unsupported shape should be built up with multiple SRAM Macro supported by the memory compiler. This step is microarchitecture-invisible and is a part of the VLSI flow, where the VLSI flow usually adopts an automatic script to generate a plan to build up each input SRAM Block with the supported SRAM Macros.

Power Model Overview: For the ultimate calculation of the SRAM power, we (1) transfer the feature from the Component to the SRAM Position, (2) propose a hardware model and activity model to estimate the hardware information and read/write frequency for the SRAM Block based on the feature of SRAM Position, and (3) then propose macro-level mapping to transform the SRAM Block level estimation to the SRAM Macro level for ultimate power calculation.

Information at the SRAM Position Level: Similar to the Component, the SRAM Position is also visible at the architecture level. Therefore, the information at the SRAM Position level includes the hardware parameter H and event parameter E of its component.

SRAM Block Hardware Model: The hardware model takes the hardware parameter of an SRAM Position as input to estimate the hardware information at the SRAM Block level, including the width, depth, and count of SRAM Blocks to build the SRAM Position.

The estimation is challenging because input information only includes high-level hardware parameters for SRAM Position. With our observation of SRAM scaling in processors, we propose a scaling-pattern-based hardware model for estimation.

Our insight is that the scaling of SRAM Blocks with respect to hardware parameters H mainly follows two general patterns: (1) Capacity scaling, the total SRAM capacity scales linearly with a hardware parameter, and (2) Throughput scaling, the width or

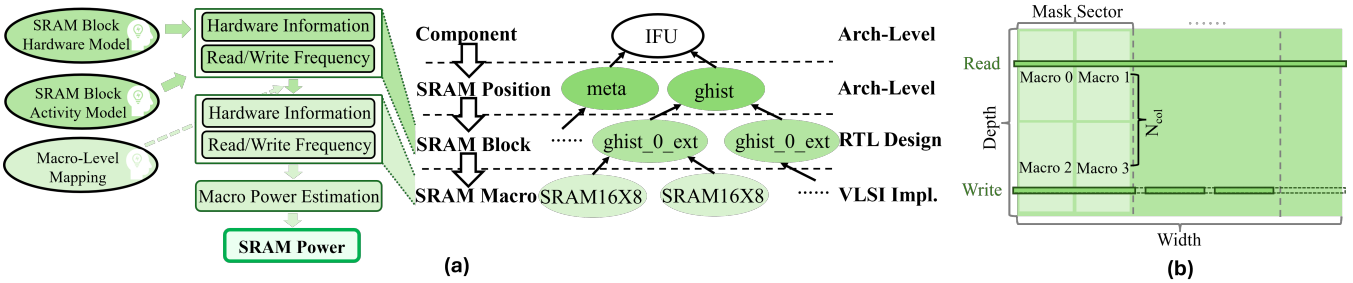


Fig. 3: (a) SRAM power model follows a top-down four-level SRAM hierarchy of processor. For each SRAM Position in a component, SRAM Block hardware model estimates hardware information of SRAM Blocks, and SRAM Block activity model estimates read/write frequency of SRAM Blocks. Based on the estimation, macro-level mapping calculates hardware information and read/write frequency of SRAM Macros, then SRAM Power can be estimated. (b) Illustration of macro-level mapping. It shows the relation between SRAM Block read/write and SRAM Macro read/write.

Training Config	Hardware Parameter			Block Information		
	FetchWidth	DecodeWidth	FetchBufferEntry	width	depth	count
C1	4	1	5	120	8	1
C15	8	5	40	240	40	1

TABLE I: The metadata table example for SRAM Block hardware model. It includes two known configurations C1 and C15 for training.

number of SRAM blocks scales linearly with a hardware parameter to provide more throughput. Based on the two scaling patterns, we can automatically infer the number and shape of SRAM Block even with limited known configurations during training.

The core of our hardware model is to detect which specific hardware parameters the throughput or capacity linearly scales with and scaling coefficient. Based on the information, we can get formulations of width, depth, and count of the SRAM Block.

To detect these hardware parameters, the model tries all hardware parameter combinations to fit a directly proportional function based on known configurations for training and selects the best combination with minimal error. We take the metadata table (meta) in IFU as an example. The hardware parameters of the metadata table include FetchWidth, DecodeWidth, and FetchBufferEntry. Table I shows the hardware parameters and SRAM Block information of two known configurations for training. The hardware parameter combinations include {FetchWidth}, {DecodeWidth}, {FetchBufferEntry}, {FetchWidth, DecodeWidth}, {FetchWidth, FetchBufferEntry}, {DecodeWidth, FetchBufferEntry}, and {FetchWidth, DecodeWidth, FetchBufferEntry}. The model checks each combination by fitting a directly proportional function and gets its error. For example, when checking if the capacity scales with {FetchWidth, DecodeWidth}, the model fits the function below,

$$\text{Capacity} = k * \text{FetchWidth} * \text{DecodeWidth}$$

The capacities, calculated by width * depth * count, of known configurations are 120 * 8 * 1 = 1920 bit and 240 * 40 * 1 = 19200 bit, the FetchWidth * DecodeWidth are 4 * 1 = 4 and 8 * 5 = 40. Therefore, the model can get the k as 240 and the error on two configurations for training as 0. This indicates the capacity scales linearly with FetchWidth and DecodeWidth, and results in the formulation of capacity as below,

$$\text{Capacity} = 240 * \text{FetchWidth} * \text{DecodeWidth}$$

Similarly, we can get the formulation of throughput and width,

$$\text{Throughput} = 30 * \text{FetchWidth}, \quad \text{Width} = 30 * \text{FetchWidth}$$

Because count is the throughput divided by width, and depth is the capacity divided by throughput, we can get the formulation of count and depth,

$$\text{Count} = 1, \quad \text{Depth} = 8 * \text{DecodeWidth}$$

Based on the formulation of width, depth, and count, we can estimate the SRAM Block information with only the hardware parameters.

The reason why we do not directly fit the width, depth, and count is that these three values usually do not scale linearly with hardware parameters in many SRAM Positions such as the table of ROB.

SRAM Block Activity Model: The activity model takes the hardware and event parameters of the SRAM Position as input to estimate the average read/write frequency of the SRAM Blocks.

Because the factors that determine the read/write frequency are complex, we utilize an ML model for the activity model. The input feature includes three parts: (1) hardware parameters of the SRAM Position, (2) event parameters of the SRAM Position, and (3) program-level features that are independent of microarchitecture, such as the number of branch instructions. All prior works do not take the program-level features into consideration. However, we find that the inaccurate performance simulator is one of the root causes of the low accuracy of the ML-based power model, therefore, some microarchitecture-independent features that are not affected by the performance simulator should be taken into account to improve the model accuracy. To collect the labels, we extract the traces of the read and write enable signals in RTL simulation to calculate the read and write frequency. The ML model we utilized is XGBoost [9].

Macro-Level Mapping: Macro-level mapping takes the hardware information and average read/write frequency of the SRAM Block as input to calculate the hardware information and read/write frequency of SRAM Macro. Hardware information of SRAM Macro includes the shape and count of required SRAM Macros to build SRAM Block.

The hardware mapping takes the SRAM Block shape (i.e. width and depth) as input to calculate the shape and count of SRAM Macros to build up the SRAM Blocks. The mapping rule is a part of VLSI flow such as BOOM VLSI flow [8]. It is available and unchanged for all processors implemented with the same VLSI flow. Therefore, given the estimated SRAM Block information, we can use the rule to get the shape and count of SRAM Macros.

To map the read and write frequency to the SRAM Macro level, we carefully analyze the building up of SRAM Block from SRAM Macro and result in a formulation to estimate the SRAM Macro level activity. Fig. 3(b) illustrates the activity of SRAM Macros when an SRAM Block read or write is triggered. When reading the SRAM Block, only the row of SRAM Macros that correspond to the address will be read. Therefore, the average read frequency of SRAM Macro f^R_{macro} is that of SRAM Block f^R_{block} divided by the number of SRAM Macros used to form a column denoted as N_{col}.

Different from the reading, when writing the SRAM Block, because of the masking adopted for writing, only the SRAM Macros that correspond to a valid mask sector will be written. To get a unified formulation for the read and write, we define "one write" to SRAM Block as a write to SRAM Block with all masks valid. For example, if an SRAM Block has two mask sectors, a write to this SRAM Block has one mask valid and another invalid, we will record it as

0.5 write. This will be taken into consideration when collecting the labels of write frequency for SRAM Block. Denoting the average read and write frequency per SRAM Macro as f^R_{macro} and f^W_{macro} and that per SRAM Block as f^R_{block} and f^W_{block}, based on the analysis in Fig. 3(b), the macro-level mapping can be formulated below,

$$f^R_{\text{macro}} = f^R_{\text{block}}/N_{\text{col}}, \quad f^W_{\text{macro}} = f^W_{\text{block}}/N_{\text{col}} \tag{9}$$

Ultimate SRAM Power Calculated at the Macro Level: With the number and types of SRAM Macros and the read and write frequency, we can estimate the SRAM power given the read and write power, denoted as P_{R} and P_{W} looked up from the technology node library. The toggling of address and data pins also has power consumption, but it only has a small contribution to the SRAM power compared with the read and write. Therefore, it can be approximately regarded as a constant. Denoting the SRAM power as P_{SRAM} and power caused by the toggling of address and data pins as C, we formulate the SRAM power as below,

$$P_{\text{SRAM}} = f^R_{\text{macro}} * P_{\text{R}} + f^W_{\text{macro}} * P_{\text{W}} + C \tag{10}$$

where C can be estimated based on the golden power of an SRAM Block collected from power simulation.

C. Power Model for Logic

AutoPower models the logic power by estimating the register power and combinational logic power individually.

Power Model for Register: The register power is the power of registers excluding its clock pin. Our register power model includes a hardware model and an activity model.

In detail, hardware model $\boldsymbol{F}_{\text{reg}}$ takes hardware parameters H as input features and golden number of registers as labels. The activity model $\boldsymbol{F}_{\text{act}}$ takes both hardware parameters H and event parameters E of the component as input features. The active rate label is P_{reg}/R. The register power is the multiplication of $\boldsymbol{F}_{\text{reg}}$ and $\boldsymbol{F}_{\text{act}}$,

$$P_{\text{reg}} = \boldsymbol{F}_{\text{reg}}(H) * \boldsymbol{F}_{\text{act}}(H, E) \tag{11}$$

Power Model for Combinational Logic: Estimating the power consumption of the combinational logic is challenging. Compared with the clock, the SRAM, and the register, the pattern of combinational logic power is more complex. Besides, the combinational logic has a variety of cell types with different power characteristics, incurring the difficulty of decoupling.

Therefore, compared with directly decoupling along physical information, we build a combinational logic power model that decouples the power into a stable power and a variation. The stable power is the power value that can reflect the general combinational logic characteristics from the power aspect, where we adopt the average power across all workloads in the fixed benchmarks for training. It is a purely hardware-related model, only depending on hardware parameters. The variation is the ratio between combinational logic power and stable power, which represents workload-specific information.

In detail, the stable model $\boldsymbol{F}_{\text{sta}}$ is trained based on the average combinational power across all workloads P_{sta}, with the hardware parameters as input features. The variation model $\boldsymbol{F}_{\text{var}}$ takes both hardware parameters and event parameters as the input features and the ratio between power and the stable power $P_{\text{comb}}/P_{\text{sta}}$ as labels. The combinational logic power prediction can be formulated as below,

$$P_{\text{comb}} = \boldsymbol{F}_{\text{sta}}(H) * \boldsymbol{F}_{\text{var}}(H, E) \tag{12}$$

Hardware Parameter	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
FetchWidth	4	4	4	4	4	8	8	8	8	8	8	8	8	8	8
DecodeWidth	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5
FetchBufferEntry	5	8	16	8	16	24	18	24	30	24	32	40	30	35	40
RobEntry	16	32	48	64	64	80	81	96	114	112	128	136	125	130	140
IntPhyRegister	36	53	68	64	80	88	88	110	112	108	128	136	108	128	140
FpPhyRegister	36	48	56	56	64	72	88	96	112	108	128	136	108	128	140
LDQ/STQEntry	4	8	16	12	16	20	16	24	32	24	32	36	24	32	36
BranchCount	6	8	10	10	12	14	14	16	16	18	20	20	18	20	20
Mem/FpIssueWidth	1	1	1	1	1	1	1	1	2	1	2	2	2	2	2
IntIssueWidth	1	1	1	1	2	2	2	3	3	4	4	4	5	5	5
DCache/ICacheWay	2	4	8	4	4	8	8	8	8	8	8	8	8	8	8
DTLBEntry	8	8	16	8	8	16	16	16	32	32	32	32	32	32	32
MSHREntry	2	2	4	2	2	4	4	4	4	4	4	8	8	8	8
ICacheFetchBytes	2	2	2	2	2	4	4	4	4	4	4	4	4	4	4

TABLE II: The CPU configurations used in our experiment. It includes 15 BOOM CPU configurations with different scales.

Component	BPTAGE	BPBTB	BPOthers	ICacheTagArray	ICacheDataArray	ICacheOthers
Hardware Parameter	FetchWidth BranchCount	FetchWidth BranchCount	FetchWidth BranchCount	ICacheWay ICacheFetchBytes	ICacheWay ICacheFetchBytes	ICacheWay ICacheFetchBytes
Component	RNU	ROB	Regfile	DCacheTagArray	DCacheDataArray	DCacheOthers
Hardware Parameter	DecodeWidth	DecodeWidth RobEntry	DecodeWidth IntPhyRegister FpPhyRegister	DCacheWay MemIssueWidth DCacheTLBEntry	DCacheWay MemIssueWidth	DCacheWay MemIssueWidth DCacheTLBEntry
Component	FP-ISU	Int-ISU	Mem-ISU	I-TLB	D-TLB	FU Pool
Hardware Parameter	DecodeWidth FpIssueWidth	DecodeWidth IntIssueWidth	DecodeWidth MemIssueWidth	ICacheTLBEntry	DCacheTLBEntry	MemIssueWidth FpIssueWidth IntIssueWidth
Component	Other Logic	DCacheMSHR	LSU	IFU		
Hardware Parameter	All	MSHREntry	LDQEntry STQEntry MemIssueWidth	FetchWidth DecodeWidth FetchBufferEntry		

TABLE III: Architecture-level hardware parameters for each component.

III. Experimental Results

A. Experiment Setup

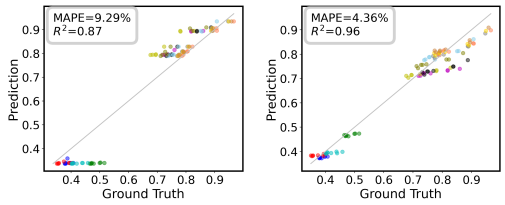
In our experiment, we adopt Chipyard [10] v1.8.1 to generate RTL code. We utilize RISC-V BOOM CPU [8] with 15 configurations with different scales for evaluation, as shown in Table II. Table III lists the hardware parameters for each component. We have eight workloads from riscv-tests [11] for evaluation, including dhrystone, median, multiply, qsort, rsort, towers, spmv, and vvadd.

We performed RTL simulation, logic synthesis, and power simulation with Synopsys VCS® [12], Synopsis Design Compiler® [13], and PrimePower [1] respectively. Our VLSI flow is based on TSMC 40nm standard cell library and associated Memory Compiler to generate SRAM. To collect event parameters, we utilize gem5 [14] as our performance simulator.

B. Power Prediction Results

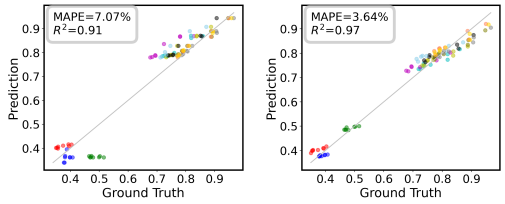
1) Summary of Baseline Methods: We compare AutoPower with the state-of-the-art architecture-level automatic power modeling method McPAT-Calib [6] as our baseline. Besides McPAT-Calib, we also include an extra baseline for the ablation study, McPAT-Calib + Component. McPAT-Calib + Component adopts the McPAT-Calib as a building block and builds power models for each component respectively. We select XGBoost [9] as the ML model adopted in McPAT-Calib and McPAT-Calib + Component, as it is the best ML model reported by McPAT-Calib [6].

2) End-to-End Power Model Accuracy: Fig. 4 and 5 visualize the comparison between AutoPower with our baseline method, McPAT-Calib, and the ablation study, McPAT-Calib + Component, when only 2 and 3 configurations are known for training when building the power model. Each point represents a configuration with a workload, points with the same configuration are in the same color. The comparisons demonstrate that AutoPower can consistently outperform McPAT-Calib. Under the two scenarios, AutoPower can achieve the highest R^2 of 0.96 and 0.97, which is 0.07 higher on average than McPAT-Calib with R^2 of 0.87 and 0.91. AutoPower can also achieve the lowest MAPE of 4.36% and 3.64%, which is 4.2% lower on average than McPAT-Calib with MAPE of 9.29% and 7.07%. This superiority is achieved by the suitable decoupling of AutoPower. Fig. 6 shows the comparison results when comparing AutoPower



(a) McPAT-Calib (b) AutoPower

Fig. 4: Accuracy Comparison with 2 configurations for training.



(a) McPAT-Calib (b) AutoPower

Fig. 5: Accuracy comparison with 3 configurations for training.

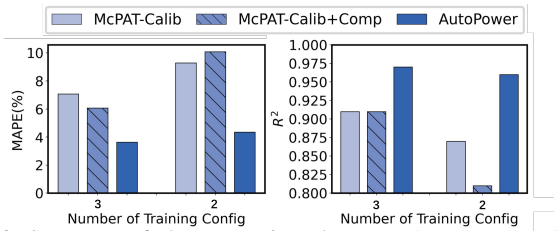


Fig. 6: Summary of the comparison between AutoPower and other methods under different numbers of known configurations for training. "Comp" stands for Component.

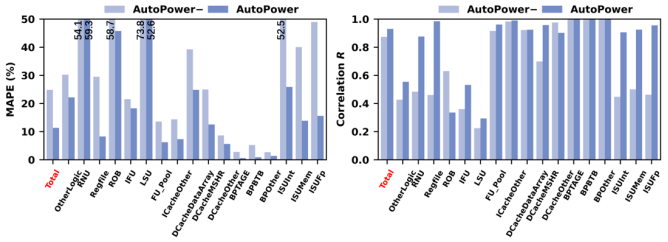


Fig. 7: Detailed comparison of clock power prediction between AutoPower− that directly utilizes the ML model and the AutoPower.

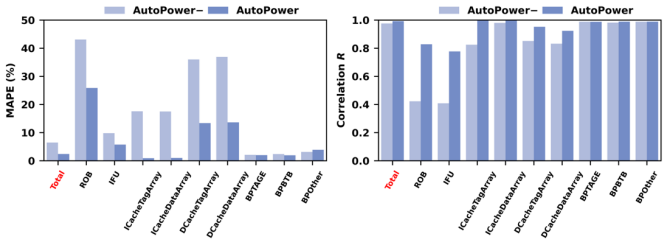


Fig. 8: Detailed comparison of SRAM power prediction between AutoPower− that directly utilizes the ML model and the AutoPower.

with the extra baseline McPAT-Calib + Component. It demonstrates that AutoPower can also outperform McPAT-Calib + Component consistently. This ablation study further verifies that power group decoupling is critical for accurate power estimation.

3) Clock Model Accuracy: To demonstrate the superiority of the AutoPower compared with directly applying the ML model for each power group, we provide another baseline, AutoPower−. It only decouples the model across different power groups and only directly adopts the ML model for the estimation of each power group.

Fig. 7 shows the prediction accuracy for the clock power and the comparison with AutoPower−. It demonstrates that our proposed clock power model can surpass AutoPower− which directly estimates the clock power for most components. Clock power prediction is challenging because of the lack of RTL-level information. Fig. 7 also shows that, with our proposed clock power modeling method, even with limited features and limited training data, AutoPower can still achieve a relatively low MAPE of 11.37% and a high correlation coefficient R of 0.93 with 2 known configurations for training.

Large Workload	Max Power Error (%)			Min Power Error (%)			Average Error (%)		
	C2	C3	C4	C2	C3	C4	C2	C3	C4
GEMM	7.7	9.9	8.1	10.2	11.6	14.9	3.5	2.3	11.0
SPMM	10.6	13.3	14.1	14.7	18.8	20.7	2.0	3.5	8.7

TABLE IV: Summary of fine-grained time-based power trace prediction results for large workloads with millions of cycles. It shows the percentage errors for maximal power, minimal power, and the average time-based power prediction error.

The high accuracy of clock power is contributed by the accurate prediction of each sub-model. The prediction of the number of registers R and gating rate g can achieve a low MAPE on average with 6.93% with 2 known configurations.

4) SRAM Model Accuracy: Fig. 8 shows prediction accuracy for SRAM power and comparison with AutoPower−. The results demonstrate that SRAM power model in AutoPower can achieve a higher accuracy compared with AutoPower− for most components. It can achieve high accuracy even if only the architecture-level information is available, with a low MAPE of 7.60% and a high correlation coefficient R of 0.94, under 2 known configurations.

Such a high accuracy benefits from the accurate estimation of the number of macros and the read/write frequency. Especially, with our insight based on the scaling pattern of SRAM in processor designs, our hardware model can accurately predict the SRAM Block hardware information with a nearly 0 MAPE, and then it can accurately model the SRAM Macro with the rule from VLSI flow.

5) Time-Based Power Trace Prediction for Large Workloads: Besides average power, fine-grained time-based power trace prediction is also critical for modern processor design. However, because of the fine-grained information required for prediction, architecture-level power trace prediction is challenging.

We adopt two large workloads with millions of cycles, GEMM and SPMM, as our target workloads for time-based power trace prediction. The length of the time step is 50 cycles. The model is trained based on only two known configurations. Table. IV shows the accuracy when applied to the C2, C3, and C4. It demonstrates that, even for fine-grained time-based power trace prediction, our model trained on two known configurations can still achieve a low MAPE without tuning on additional time-based power trace data.

IV. Conclusion

In this work, we propose AutoPower targeting fully automated architecture-level power modeling with limited known design configurations. It decouples the modeling into different power groups and further decouples the power models into multiple simple sub-models based on the specific structural information of each power group. AutoPower lowers the data requirement barrier, which is a compelling addition to architects' toolbox.

Acknowledgement

This work is supported by National Natural Science Foundation of China (NSFC) 62304192, Hong Kong Research Grants Council (RGC) ECS Grant 26208723, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and National Supercomputing Center in Guangzhou Nansha Sub-center for computational resources.

REFERENCES

[1] Synopsys, "Primepower: Rtl to signoff power analysis," 2023. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html

[2] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*, 2009, pp. 469–480.

[3] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 83–94, 2000.

[4] Q. Zhang, S. Li, G. Zhou, J. Pan, C.-C. Chang, Y. Chen, and Z. Xie, "Panda: Architecture-level power evaluation by unifying analytical and machine learning solutions," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 01–09.

[5] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, "Quantifying sources of error in mcpat and potential impacts on architectural studies," in *2015 IEEE 21st International symposium on high performance computer architecture (HPCA)*. IEEE, 2015, pp. 577–589.

[6] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "Mcpat-calib: A risc-v boom microarchitecture power modeling framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 243–256, 2022.

[7] J. Zhai, Y. Cai, and B. Yu, "Microarchitecture power modeling via artificial neural network and transfer learning," 2023.

[8] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[10] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[11] *risc-v tests*, https://github.com/riscv-software-src/riscv-tests.

[12] "VCS® functional verification solution," https://www.synopsys.com/verification/simulation/vcs.html, 2021.

[13] "Design Compiler® RTL Synthesis," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html, 2021.

[14] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

Fast End-to-End Simulation and Exploration of Many-RISCV-Core Baseband Transceivers for Software-Defined Radio-Access Networks

Marco Bertuletti
ETH Zürich
Zürich, Switzerland
mbertuletti@iis.ee.ethz.ch

Yichao Zhang
ETH Zürich
Zürich, Switzerland
yiczhang@iis.ee.ethz.ch

Mahdi Abdollahpour
Università di Bologna
Bologna, Italy
mahdi.abdollahpour@unibo.it

Samuel Riedel
ETH Zürich
Zürich, Switzerland
sriedel@iis.ee.ethz.ch

Alessandro Vanelli-Coralli
ETH Zürich
Zürich, Switzerland
Università di Bologna
Bologna, Italy
avanelli@iis.ee.ethz.ch

Luca Benini
ETH Zürich
Zürich, Switzerland
Università di Bologna
Bologna, Italy
lbenini@iis.ee.ethz.ch

***Abstract*—The fast-rising demand for wireless bandwidth [1] requires rapid evolution of high-performance baseband processing infrastructure. Programmable many-core processors for software-defined radio (SDR) have emerged as high-performance baseband processing engines, offering the flexibility required to capture evolving wireless standards and technologies [2]–[4]. This trend must be supported by a design framework enabling functional validation and end-to-end performance analysis of SDR hardware within realistic radio environment models. We propose a static binary translation based simulator augmented with a fast, approximate timing model of the hardware and coupled to wireless channel models to simulate the most performance-critical physical layer functions implemented in software on a many (1024) RISC-V cores cluster customized for SDR. Our framework simulates the detection of a 5G OFDM-symbol on a server-class processor in 9.5s-3min, on a single thread, depending on the input MIMO size (three orders of magnitude faster than RTL simulation). The simulation is easily parallelized to 128 threads with 73-121× speedup compared to a single thread.**

***Index Terms*—RISC-V, SDR, 5G, 6G, SBT.**

I. Introduction

The worldwide demand for more wireless subscriptions (8.5 billion in 2023) and network data traffic (145EB in Q1-2024) [1] fostered the development of new Radio-Access-Networks (RAN) technologies. The Massive Multiple Input Multiple Output (MIMO) paradigm ratified by 3rd Generation Partnership Project (3GPP) Release-17 [5] gained momentum and will extend to more users and antennas in the upcoming 6th Generation (6G) releases [6]. Alongside increased performance requirements, channel traffic characteristics will also change, resulting in a dynamic range from a few kBs for a voice call to GBs for high-quality multimedia [7].

The flexibility required to address demanding worst-case specifications and wide operation ranges can be obtained by implementing the MIMO processing workload on programmable cores, an approach known as Software Defined Radio (SDR). Multi-core and many-core systems promise high performance on SDR 5th Generation (5G) workloads at the edge, implementing the full Physical Layer (PHY) on basestations (BSs) [2]–[4]. Additionally, the RISC-V Instruction Set Architecture (ISA) emerged as a powerful domain-specialization tool to fight back the energy-efficiency loss of instruction processors vs. custom hardware, enabling the co-design of efficiency-boosting processors features and domain-specific instruction extensions [3], [4].

Evaluating the algorithmic (functional) performance of software running on SDR-hardware and its execution performance under varying network traffic, spatial topology, and transmission channel conditions is critical to the design of future BSs. To this purpose, a model of the SDR receivers must be simulated as Device Under Test (DUT) in an end-to-end (E2E) transmission, matching the following requirements: deterministic behavior, flexibility to evolving standards, and awareness of timing to synchronize with other elements of the processing chain [7]. Additionally, a fast simulation framework is needed to evaluate the receiver performance via extensive numerical Monte Carlo (MC) simulation.

TABLE I
Simulation methods for SDR baseband hardware.

	Device	Freq.	Speed	Design Effort	Multi-Core
[8], [9] RTL	-	-	↓	↓	✗
[10] TLM	Intel Core2	3.00GHz	↑	↑	✗
[11] FPGA	XCZU28DR	128MHz	↑	↓	✗
[2] FPGA	ZCU102	120MHz	↑	↑	✓
Ours SBT	AMD EPYC-7742	3.25GHz	↑	↓	✓

Table I summarizes different options to include SDR hard-

ware as DUT in an E2E wireless transmission testbench. Event-based (e.g., QuestaSim [12]) and compiled (e.g., Verilator [13]) Register Transfer Level (RTL) simulators, require low effort to directly simulate Hardware Description Language (HDL) code of baseband receivers [8], [9], [14]. Unfortunately, they are too slow for MC simulation of complex workloads. Transaction Level Modeling (TLM) in SystemC allows the creation of detailed virtual prototypes of entire digital systems [10] and can also be used for the simulation of RF analog transceivers [15]–[17]. However, TLM simulations are limited in speed for the sequential execution of SystemC processes managed by a centralized kernel [18] and require long compilation time, constraining design iterations. Field Programmable Gate Array (FPGA) [2], [11] or Application Specific Integrated Circuit (ASIC) prototyping can not be used to implement a fast hardware-software co-design loop, because it requires a fully designed system and can be used only late in the design process. In the case of complex baseband many-core receivers, the prototyping on FPGA also requires considerable design effort. High Level Synthesis (HLS) tools, such as Python Productivity for Zynq (PYNQ) [11], are used to emulate small dataflows on FPGA, but the approach does not extend to many-core baseband processors with hundreds of processing elements. Moreover, implementing multi-core systems on FPGA must trade off limited hardware resources and operating frequencies, as clearly shown by the implementation of 36 RI5CY cores on Xilinx XCVU9, presented in [2].

This paper proposes a lightweight Static Binary Translation (SBT) based framework to simulate SDR many-core programmable transceivers based on the RISC-V ISA. Our simulator can be connected to channel models for extensive MC analysis and is easily parallelized on Commercial Off The Shelf (COTS) multi-core servers. The following contributions quantify these claims:

- We propose the open-source[1] SBT-based simulator Banshee [19] for deterministic and instruction accurate emulation of many-core transceivers. The simulator is connected to wireless channel models [20] for E2E MC analysis of the receiver performance. As a challenging benchmark, we emulate the 1024 independent instruction streams of the open-source[2] programmable TeraPool-SDR [3] on a 128-cores AMD EPYC-7742 server.
- We demonstrate fast design space exploration on the Minimum Mean Squared Error (MMSE) detection in different arithmetic precisions. A single thread of the 128-core AMD EPYC-7742 server simulates the detection of an Orthogonal Frequency Division Multiplexing (OFDM) symbol in $<$ 3min, depending on the MIMO size. A small 4x4 input only requires 9.5s, compared to 13h:44min for RTL simulation of the same binary (Questasim-2022.3). Independent symbols are easily parallelized, with 73-121$\times$ speedup, compared to a single thread.
- We show that augmenting the SBT-based simulator with an approximate timing model of the hardware offers first-order estimates of the receiver processing time, with 30% average error over the measured RTL cycle-count for different MMSE implementations and MIMO sizes.

Running MMSE detection with a speedup over RTL simulation exceeding 1000$\times$, the developed framework enables fast MC analysis of programmable baseband transceivers.

[1]https://github.com/pulp-platform/banshee
[2]https://github.com/pulp-platform/mempool

II. Many-Core SDR Architecture

TeraPool [3], the largest shared-memory many-core cluster for SDR presented in the open literature, is a challenging simulation benchmark: emulating its 1024 independent instruction streams, diverse ISA, and large hierarchical interconnect is time-consuming in RTL simulation and would require consistent modeling effort in TLM. Moreover, TeraPool's open-source RTL design enables comparisons with cycle-accurate simulation.

The TeraPool architecture, illustrated in Figure 1, features *Snitch* processing elements that support the RV32IMAF ISA. The single-stage core decodes instructions and executes the RV32I ISA, excluding load&stores. Other instructions are offloaded to co-processing functional units, including a Load&Store Unit (LSU), an Integer Processing Unit (IPU) for integer multiplication and division, and a Floating Point Unit (FPU) implementing the *zfinx* and *zhinx* extensions [21].

Snitch's functional units also implement custom integer and Floating Point (FP) RISC-V extensions for Digital Signal Processing (DSP). The *Xpulpimg* set includes post-increment load&stores, Multiply and Accumulates (MACs), and Single Instruction Multiple Data (SIMD) compute and data-shuffling operations on 8b-16b integer types. The *Smallfloat* and *Minifloat* sets [22], [23] include SIMD operations on FP 8b-16b types, widening-dotproduct (wDotp) with 8b-16b operands and 16b-32b accumulators, and complex 16b FP MAC.

The base of TeraPool's hierarchical design is the *Tile*, where 8 cores share 32 KiB of scratchpad memory with 1-cycle access latency and 4 KiB of I$. The cores access the scratchpad of other Tiles via shared ports to the cluster interconnects. One 8x8 crossbar connects 8 Tiles in the *SubGroup* hierarchy. Three 8x8 crossbars connect Tiles of the SubGroup to Tiles of 3 other SubGroups in the *Group* hierarchy. Three 32x32 crossbars connect the Tiles of a Group to Tiles in 3 other Groups, summing up to 128 Tiles and 4 MiB of scratchpad in a Cluster. Pipeline stages at hierarchy boundaries configure non-uniform memory access (less than 9 cycles without contentions). A cluster-level AXI interconnect and a Direct Memory Access (DMA) engine allow explicit data transfers from L2 memory.

III. Co-simulation of DUT with TX-Channel

We describe a wireless E2E MIMO transmission and the developed framework for co-simulation of the transmission channel model and of the RISC-V DUT.

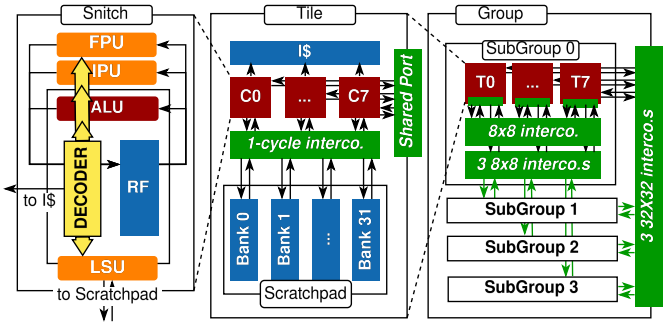


Fig. 1. Snitch core, Tile, SubGroup, and Group of TeraPool.

A. End-to-end MIMO wireless transmission

MIMO is a key enabling technology for 5G and beyond networks, where the BS is equipped with N_{RX} antenna elements, which receive signals from N_{TX} user equipments (UEs). During uplink, the transmission bit sequence maps to a Quadrature-Amplitude Modulation (QAM) constellation, and blocks of QAM-symbols are transmitted through the channel on N_{SC} OFDM subcarriers, namely an OFDM-symbol [5]. At the BS, the detection algorithm uses channel state information (CSI) to estimate the transmitted QAM-symbols. The BS processes a Transmission Time Interval (TTI) with 14 OFDM-symbols in <1ms. The evolution from 5G to 6G will increase in the N_{SC}, N_{TX}, and N_{RX}, affecting the Key Performance Indicators (KPIs) of DSP detection algorithms as well as their hardware implementation.

We therefore identify MIMO detection as a reference application to prove the capabilities of our co-simulation approach. We implement the MMSE detector, which offers optimal performance for transmission through a Additive White Gaussian Noise (AWGN) channel, on the target SDR-hardware.

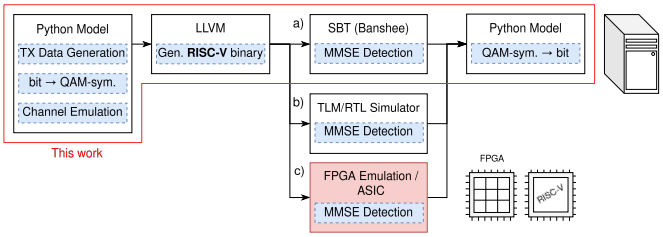


Fig. 2. Different approaches to add RISC-V hardware in the loop of a telecommunication application: c) execution on prototype ASIC, emulation on FPGA, b) cycle-accurate RTL/TLM simulation and a) simulation on Banshee.

B. Simulation of the MIMO wireless transmission

The proposed framework and flow are described in Figure 2. A Python model of the MIMO transmission based on [20] runs on the simulation host. The model includes bit-sequence transmission generation for different users, QAM-symbol mapping, and wireless channel emulation. It also implements a double-precision reference of the detection MMSE algorithm.

The SDR hardware running MMSE processing is simulated on Banshee, an open-source LLVM-based binary translator for instruction-accurate emulation of multi-core systems. The simulation flow in Banshee is divided into two phases. During *translation*, Banshee converts a RISC-V binary into LLVM Intermediate Representation (IR) via SBT. The generated IR references to Banshee runtime functions to keep track of the emulated device state. LLVM optimizes and translates the IR into host code. During *emulation*, Banshee maps the translated code to multiple threads on the host platform for each hardware thread of the emulated RISC-V multi-core.

Besides offering an instruction-accurate simulation of the host and taking advantage of fully parallel execution, Banshee assigns a static latency to each instruction to estimate the program runtime. To model outstanding instruction execution, Banshee implements a scoreboard that keeps track of the Read After Write (RAW) dependencies between instructions. This feature allows a fast software performance estimation by accounting for instruction stalls caused by long-latency instructions such as loads&stores.

We modified Banshee to translate *zfinx*, *zhix*, *Smallfloat*, and *Minifloat* FP extensions, which compute on operands in the integer register file, as specified in [21]. The latency of FP instructions is also annotated in the simulator for first-order estimation on the emulated code runtime.

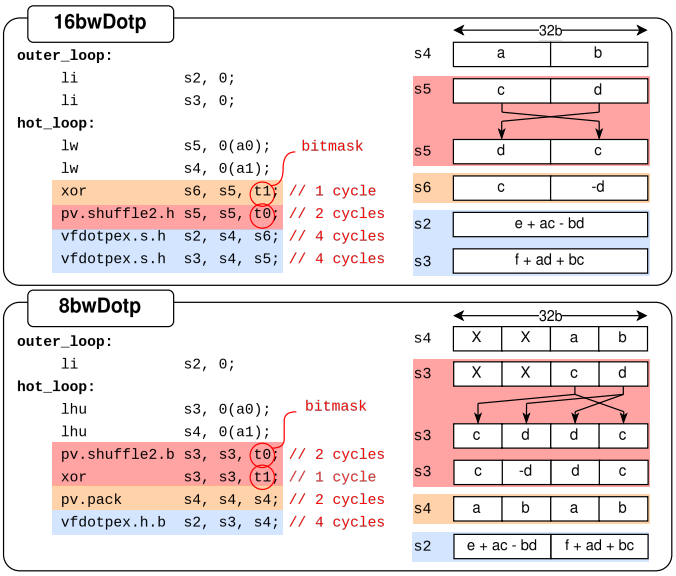


Fig. 3. Execution of the complex MAC in different arithmetic precisions using *Smallfloat* and *Xpulpimg* instruction sets.

IV. Software-defined MMSE on TeraPool

This section describes the SDR implementation of a MMSE detector on TeraPool. Given y the received signal, $\hat{H}$ and σ an estimation of the channel and noise at the BS, and $\hat{x}$ the detected signal, the analytical MMSE detector is:

$$\hat{x}_i = (\hat{H}_i{}^H \hat{H}_i + \sigma_i^2 I)^{-1} \hat{H}_i{}^H y_i \quad \forall i \in [1, N_{SC}] \qquad (1)$$

The operator in Equation (1) requires a matrix inversion. Given $G = (\hat{H}^H \hat{H} + \sigma^2 I)$ and $z = \hat{H}^H y$, we decompose $G = L^H L$ in its lower and upper triangular components, using the Cholesky algorithm. The linear system $G\hat{x} = z$ is solved by

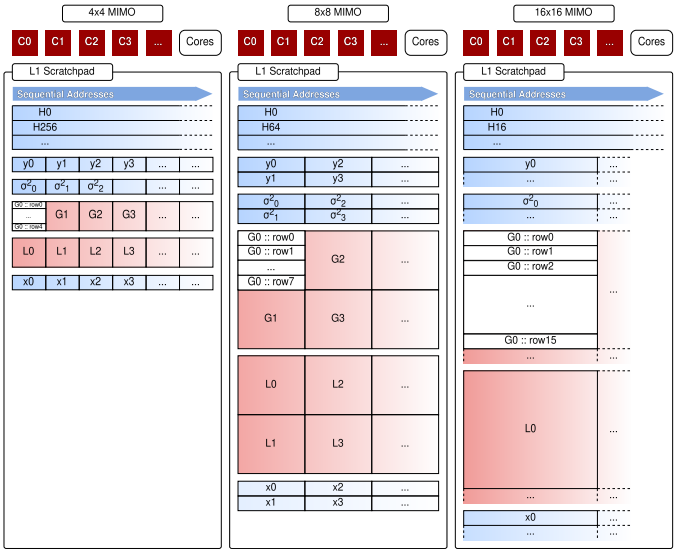


Fig. 4. Implementation of parallel MIMO-MMSE on the TeraPool many-core cluster for different sizes of the MIMO problem.

inverting two lower triangular matrices $\hat{x} = L^{-1}\left((L^H)^{-1}z\right)$. This reduces the overall arithmetic complexity of the inversion, as described in [24]. The MMSE detection then requires four operators: the Hermitian of the channel, the Matrix Vector Multiplication (MVM) between the complex-conjugate of the channel and the received symbol, the Cholesky decomposition, and the solution of a triangular system.

TeraPool has a *fork-join* type programming model (common trait of many-core clusters). In the *fork* phase of the program, each core executes an independent instruction stream in parallel. The *join* phase consists of a synchronization barrier: cores wait for each other before executing the next parallel section.

In an OFDM transmission a different MMSE problem must be solved for each subcarrier of the transmission spectrum. In the parallel execution on TeraPool, the kernels of MMSE are implemented for a single Snitch core, and an independent MMSE problem is then distributed in parallel to each core. The single-core MMSE is implemented in different arithmetic precisions, as this is a key exploration parameter for trading off functional accuracy with execution speed and energy.

1) 16bHalf: uses the RISCV *zhinx* set. We load 16b words. A complex MAC requires four `fmadd.h` instructions.

2) 16bwDotp: uses the wDotp extension of the *Smallfloat* set, with 32b accumulation. We load 32b words, and we use two wDotps and a SIMD `shuffle` instruction from *Xpulpimg* set to implement the complex MAC, as described in Figure 3.

3) 16bCDotp: uses the complex dotproduct operation. The multiplication internal precision is 32b, and real-imaginary accumulators are in 16b, fitting a 32b word.

4) 8bQuarter: uses *Smallfloat* 1b sign, 4b exponent, 2b mantissa FP format. After the execution of the $\hat{H}^H\hat{H}$ matrix product and of the $\hat{H}^H y$ MVM, we cast the outputs to 16b to solve the $G\hat{x} = z$ linear system in higher numerical precision.

5) 8bwDotp: uses *Smallfloat* wDotp for the matrix products and the MVM, with accumulators in 16b. A complex MAC requires a wDotp instruction and a SIMD `shuffle` instruction from the *Xpulpimg* set, as described in Figure 3. The linear system is solved in 16b precision.

For the parallel execution, each core runs an independent MMSE problem (1024 in total). The operands are allocated in the shared memory of TeraPool to reduce the cores' contentions to shared interconnect resources [25]. In Figure 4, we represent TeraPool cores and the corresponding input-outputs of independent MIMO problems. The elements of vectors in Equation (1) (y, σ^2, $\hat{x}$, and $\hat{H}$, flattened to a one-dimension array in row-major order) lay in consecutive addresses of L1. This matches the allocation of data in L2 and does not require relocating elements after explicit DMA transfers to L1. On the contrary, intermediate outputs of the computation L matrix and G matrix have rows on different memory rows. This keeps the data local to the processing cores. For example, in the 4x4 MIMO, each core sends requests to different banks, fetching y, σ^2, $\hat{x}$, and G, and in the 32x32 MIMO only 8 cores contend for the same scratchpad banks.

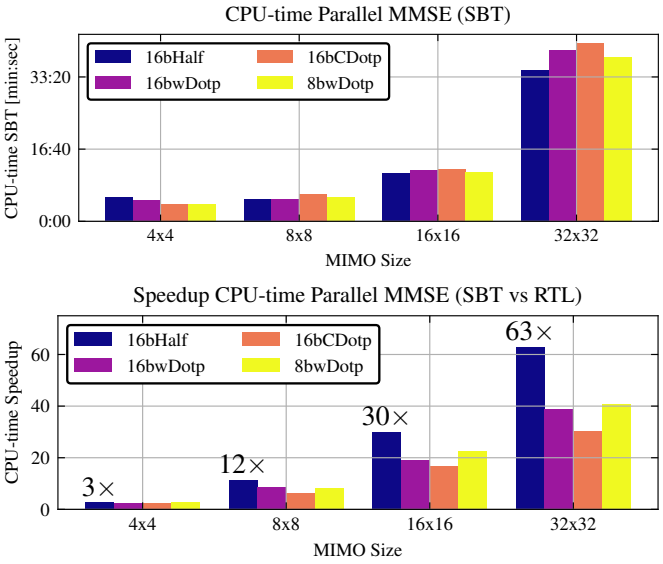


Fig. 5. Parallel MMSE in different precisions and input sizes: speedup on 128-cores AMD EPYC-7742 of multi-thread Banshee simulation against single-thread RTL simulation with QuestaSim-2022.3.

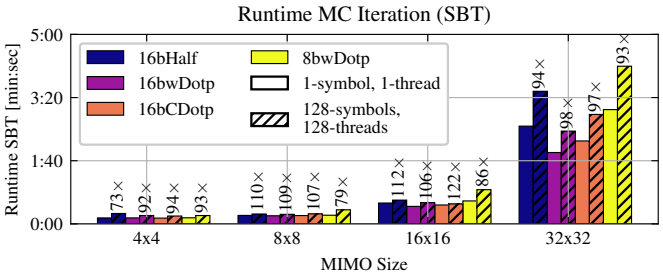


Fig. 6. Batched execution on a Snitch of $N_{SC} = 1638$ MMSE problems of an OFDM-symbol: single-thread runtime of Banshee simulation and multi-thread simulation of independent symbols with corresponding speedup, on a 128-cores AMD EPYC-7742 server.

V. BER & Runtime Analysis

This section explains how SBT can be used for fast design-space exploration of MMSE in different arithmetic precisions. It presents Banshee runtime, then shows how to measure the KPIs of software-defined MIMO applications.

A. Simulator runtime performance

We execute 1024 MMSE problems in parallel on 1024 TeraPool cores. We simulate a total of 1024 independent instruction streams. The experiment runs on a 128-CPUs COTS server processor at 3.25GHz (AMD EPYC-7742). Each CPU has 32MiB L3-cache and shares 1TB of system memory. We compare the total CPU-time of a multi-thread Banshee simulation with the CPU-time of a single-thread RTL simulation (Questasim-2022.3 being intrinsically single-threaded). For multi-thread, we average the results of 10 independent runs to filter the effect of competing processes on the server. Figure 5 shows the runtime (top) and speedup (bottom) results: the total CPU-time for a Banshee call amounts to average $<$ 35min, while the average wall-clock runtime is $<$ 2min:45s, over all the input sizes. Comparing CPU-times, Banshee simulation is up to 63$\times$ faster than single-thread QuestaSim-2022.3 RTL simulation. Comparing wall-clock runtimes, Banshee multi-thread simulation is up to 1582$\times$ faster than single thread RTL simulation.

To speed up the simulation, the MC simulation of independent MMSE problems is batched on a single TeraPool core. We consider a New Radio (NR) transmission in a 50MHz bandwidth, with $N_{SC} = 1638$, 30kHz subcarrier spacing, and 0.5ms TTI duration. We simulate a MC iteration, consisting of $N_{SC} = 1638$ MMSE problems running on a single Snitch, with a single AMD EPYC-7742 thread. The maximum speed for a single-threaded Banshee simulation of a single Snitch is 3.57 Million Instructions Per Second (MIPS). The runtime in Figure 6, shows $<$ 3min per MC iteration, and down to 9.44s for an input 4x4 MIMO, compared to 13h:44min required by RTL simulation of the same binary (5237$\times$ faster).

Independent OFDM symbols can be easily parallelized over the 128 cores of the server. Figure 6 shows the runtime with 128 active threads and the speedup compared to a single thread. We measure 12.5s-4min:10s simulation runtime per 128 MC iterations, depending on the MIMO size and arithmetic precision: 73-121$\times$ speedup compared to single-thread execution. The achieved simulation throughput is fully compatible with a fast HW-in-the-loop exploration of the E2E 5G application under exam.

B. MMSE cycle-count on TeraPool

Figure 7 reports the cycle-count of the complete parallel *16bHalf* MMSE application, measured by RTL simulation in QuestaSim-2022.3. The cycle-accurate RTL simulation gives precise information on the speedup of lower arithmetic precisions compared to *16bHalf*. SIMDs reduce the total number of instructions issued. In particular, for a 32x32 input, the ratio between instructions issues in the *16bHalf* compared to *16bwDotp*, *16bCDotp*, and *8bwDotp* corresponds to 1.15$\times$, 1.84$\times$, and 1.37$\times$. These values are slightly smaller than the measured speedup (1.29$\times$, 1.86$\times$, 1.38$\times$): requiring fewer loads, compared to *16bHalf*, SIMD instructions also reduce the stalls from contentions in the shared interconnect.

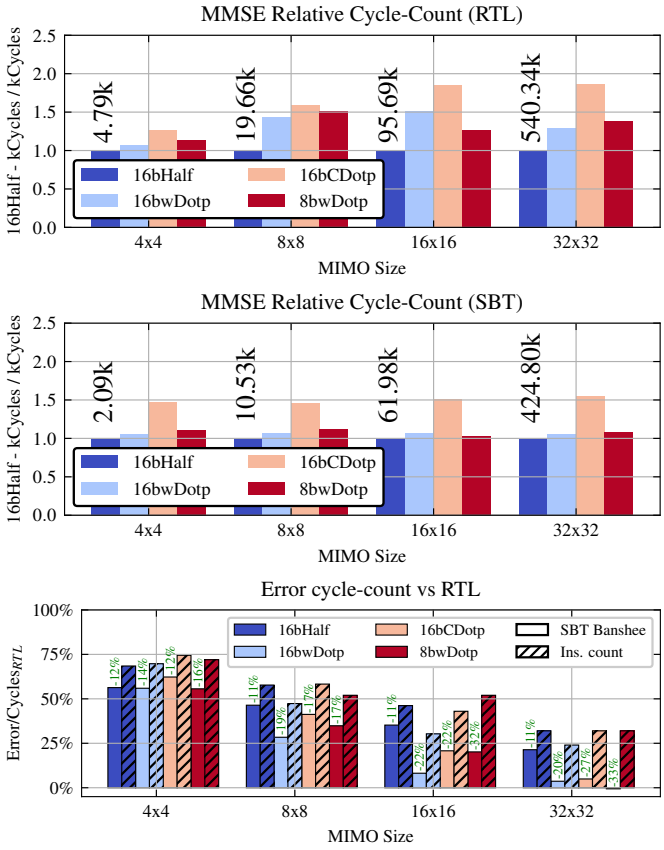


Fig. 7. Parallel implementation of the MMSE in different precisions for various sizes of MIMO. Error between the cycles measured by RTL simulation with QuestaSim-2022.3 and estimated by Banshee or by counting instructions.

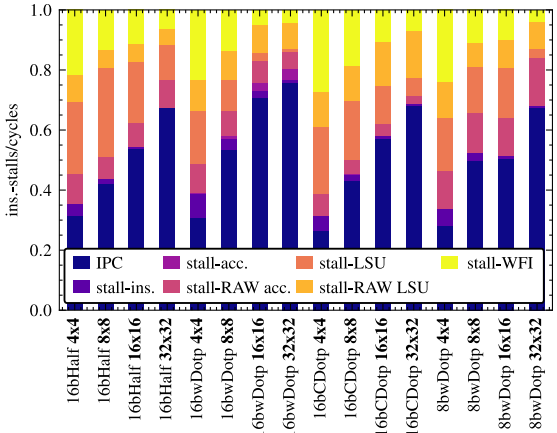


Fig. 8. Breakdown of instructions and architectural stalls over the cycle-count. Result of cycle-accurate experiments in QuestaSim-2022.3.

The breakdown of instructions and stalls over the cycle count for different arithmetic precisions and input problem sizes is in Figure 8. We measure few *stall-ins.* for I$ refill and

stall-acc., occurring when the pipelines of Snitch functional units are full. Loops are unrolled to minimize RAW stalls, with increasing benefits at higher problem sizes. The amount of stalls due to contentions in the interconnect (*stall-LSU*) is the highest for the low arithmetic intensity *16bHalf* application. Finally, *stall-WFI* refers to the time spent by cores idling at the synchronization during the fork-join MMSE application.

Figure 7 reports a comparison between the cycle count measured by RTL simulation, the cycle count estimated by Banshee, and a rough estimation based on the instruction count. Privileging multi-thread fast simulation of parallel programs on the host, Banshee omits priorities between loads&stores targeting the same memory bank or Tiles' shared-ports to interconnects, as well as priorities between atomic memory operations, used to implement synchronization barriers. Modeling these effects would require expensive synchronization between hardware threads or computationally heavy statistical models. Our approximate solution, oriented to simulation speed, is conservative in assigning statically to all the transactions the largest memory access latency without contentions (9 cycles).

On average, we obtain 30% error compared to RTL simulation cycle-count. The largest errors measured (20-62%) correspond to the *16bHalf* implementation, which loads&stores separately the real-imaginary parts of the operands (twice the number of memory operations, compared to other kernels). Small MIMO problems with large synchronization overhead are more affected by the permissive latency modeling of atomics. Nevertheless, in this unfavorable case, Banshee obtains 12-16% improvement (in green) on the rough estimation considering the instruction count only. For a large 32x32 MIMO input, the error of Banshee is only 0.6-20%.

Despite the cycle performance over-estimation, in Figure 7 we note that measuring the cycle count in Banshee does not invalidate the key insight on the speedup of SIMD implementations, compared to *16bHalf*. The values measured for 32x32 inputs, 1.05×, 1.54×, 1.07×, confirm *16bCDotp* as the fastest option, followed by *8bwDotp* and *16bwDotp*. The tool is useful for rough estimations of the E2E cycle count. Targeting an open-source DUT facilitates further calibration and reduction of performance over-estimation based on RTL simulations results.

C. BER extraction via fast MC simulation

A classic telecommunications KPI is the Bit Error Rate (BER), which measures the ratio of bit errors compared to the original transmitted bit sequence. We compute the BER of the MMSE implemented on TeraPool simulating MC iterations with Banshee and leveraging its bit-true functional modeling accuracy. For different input Signal to Noise Ratio (SNR), we iterate to a target error count. We explored the performance of the MMSE detector in different arithmetic precisions according to three variables: MIMO problem size, bit-sequence modulation, and wireless channel type.

The input bit-sequence maps to different QAM constellations. In Figure 9, we compare the BER vs. SNR, obtained for a 4x4 or a 32x32 MMSE problem with 16QAM and a 64QAM modulation. We observe the *16bHalf*, *16bwDotp*, and *16bCDotp* implementations achieving the same results as the *64bDouble* Python model. The *8b* implementations both suffer the truncation of results precision before the *16b* matrix inversion, corresponding to a 10× loss at 18dB.

In Figure 10, we examine the BER of the receiver, assuming a flat-fading Rayleigh channel. While the AWGN channel assumes zero attenuation and interference from other transmitters, the Rayleigh channel considers the effects of multi-path fading and models a 5G-MIMO transmission more closely. The MC co-simulation of TeraPool and the wireless channel gives immediate feedback on the performance of low-bit arithmetic precision RISC-V extensions on the full wireless application: only the *16bwDotp* and *16bCDotp* implementations follow the performance of the *64bDouble* golden model. The fast co-simulation approach revealed the benefits of accumulating in *32b* (*16bwDotp*), or execution of the complex MAC in *32b* internal precision (*16bCDotp*).

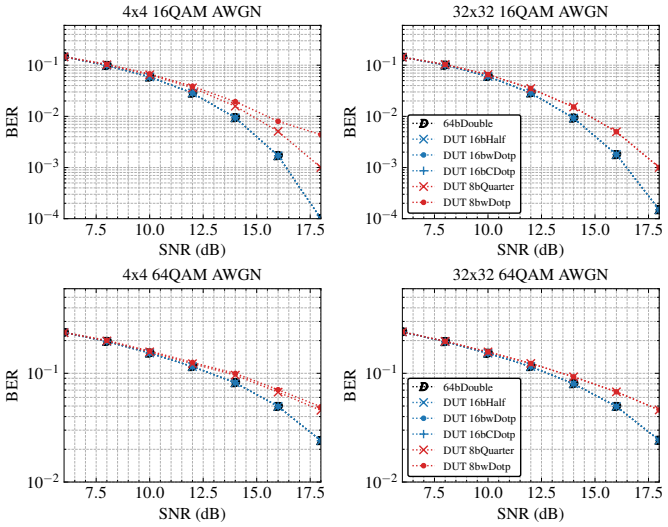


Fig. 9. BER for 16QAM and 64QAM modulation, AWGN channel.

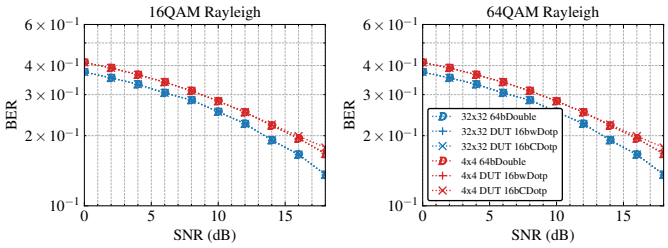


Fig. 10. BER for 16QAM and 64QAM modulation, Rayleigh channel.

VI. Conclusions

This work presented an open-source SBT-based lightweight simulator coupled to wireless channel models to extract the E2E performance of RISC-V SDR-transceivers. The simulation framework fulfills the requisites [7] to model BS programmable receivers in large-scale RAN testbed. SBT offers

fast, functional modeling of SDR workloads on the open-source TeraPool 1024-core cluster, accounting for the low-bit precision of the hardware. It enables fast design space exploration of E2E 5G transmissions, considering different modulation schemes and wireless channels: MMSE on a 5G OFDM-symbol is simulated in only 9.5s-3min, depending on the input MIMO size, three orders of magnitude faster than RTL simulation. Independent symbols are parallelized on up to 128 threads of a COTS server with 73-121$\times$ speedup compared to single-thread simulation. The translation approach is agnostic of the algorithm used, which makes the method flexible to an evolving telecommunication standard. Finally, augmenting the simulator with a simple hardware timing model gives first-order estimates of the runtime on the DUT, with only 30% average error, for different MIMO sizes. The fully open-sourced RTL of the DUT ease further calibration and reduction of the performance over-estimation.

ACKNOWLEDGMENT

This work is funded in part by the COREnext project supported by the EU Horizon Europe research and innovation program under grant agreement No. 101092598.

REFERENCES

[1] P. Jonsson and et al., "Ericsson Mobility Report 2024," Ericsson, Tech. Rep., 2024. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report

[2] A. Kamaleldin, S. Hesham, and D. Göhringer, "Towards a Modular RISC-V Based Many-Core Architecture for FPGA Accelerators," *IEEE Access*, vol. 8, pp. 148 812–148 826, 2020.

[3] Y. Zhang, S. Riedel, M. Bertuletti, A. Vanelli-Coralli, and L. Benini, "TeraPool-SDR: An 1.89TOPS 1024 RV-Cores 4MiB Shared-L1 Cluster for Next-Generation Open-Source Software-Defined Radios," in *proceeding of 2024 Great Lakes Symposium on VLSI (GLSVLSI)*, 2024, p. 86–91.

[4] S. Rajagopal, "EDGE Q 5G with an EDGE," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–13.

[5] 3GPP, "NR and NG-RAN Overall Description," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.211, 2017, Release 17.

[6] L.-H. Shen, K.-T. Feng, and L. Hanzo, "Five Facets of 6G: Research Challenges and Opportunities," *ACM Comput. Surv.*, vol. 55, no. 11, pp. 1–39, 2023.

[7] R. Wittig, A. Goens, C. Menard, E. Matus, G. P. Fettweis, and J. Castrillon, "Modem Design in the Era of 5G and Beyond: The Need for a Formal Approach," in *2020 27th International Conference on Telecommunications (ICT)*, 2020, pp. 1–5.

[8] H. Rezgui, F. Rouissi, and A. Ghazel, "Design Methodology Proposal of Digital Predistorter Using Matlab and Modelsim Cosimulation," in *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, 2018, pp. 1–5.

[9] C. Y. Park and J. Kim, "Event-driven modeling and simulation of 5g nr-band rf transceiver in systemverilog," in *SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME*, 2021, pp. 1–4.

[10] A. Barreteau, S. Le Nours, and O. Pasquier, "A Case Study of Simulation and Performance Evaluation of a SDR Baseband Architecture," *Journal of Signal Processing Systems*, vol. 73, no. 3, pp. 267–279, 2013.

[11] S. Cheng, O. A. Topal, M. Ozger, C. Cavdar, O. Edfors, and L. Liu, "A Hardware-in-the-Loop Simulator for mmWave Massive MIMO Using PYNQ Framework," in *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2023, pp. 698–702, iSSN: 1558-3899. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10405907

[12] "QuestaSim." [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/

[13] "Verilator." [Online]. Available: https://www.veripool.org/verilator/

[14] "5G SoC Design: Enabling Virtual Verification with Keysight." [Online]. Available: https://www.synopsys.com/blogs/chip-design/5g-soc-design-virtual-verification-with-keysight.html

[15] G. Miorandi, F. Stefanni, F. Fraccaroli, and D. Quaglia, "A SystemC-based Simulator for design space exploration of smart wireless systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 666–671.

[16] B. Beichler, M. Rethfeldt, H. Raddatz, L. Steffen, D. Timmermann, and C. Haubelt, "A SystemC Model of the IEEE 802.11 EDCA Protocol for Virtual Prototyping," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021, pp. 1–7.

[17] G. Gailliard, E. Nicollet, M. Sarlotte, and F. Verdier, "Transaction Level Modelling of SCA Compliant Software Defined Radio Waveforms and Platforms PIM/PSM," in *2007 Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1–6.

[18] A. Lonardi and G. Pravadelli, "On the Co-simulation of SystemC with QEMU and OVP Virtual Platforms," in *VLSI-SoC: Internet of Things Foundations*, L. Claesen, M.-T. Sanz-Pascual, R. Reis, and A. Sarmiento-Reyes, Eds. Cham: Springer International Publishing, 2015, pp. 110–128.

[19] S. Riedel, F. Schuiki, P. Scheffler, F. Zaruba, and L. Benini, "Banshee: A Fast LLVM-Based RISC-V Binary Translator," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[20] J. Hoydis, S. Cammerer, F. A. Aoudia, A. Vem, N. Binder, G. Marcus, and A. Keller, "Sionna: An Open-Source Library for Next-Generation Physical Layer Research," 2023. [Online]. Available: https://arxiv.org/abs/2203.11854

[21] "Announcing public review for RISC-V standard extensions Zfinx, Zdinx, Zhinx, and Zhinxmin." [Online]. Available: https://riscv.org/blog/2021/08/announcing-public-review-for-risc-v-standard-extensions-zfinx-zdinx-zhinx-and-zhinxmin/

[22] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "Design and Evaluation of SmallFloat SIMD extensions to the RISC-V ISA," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 654–657.

[23] L. Bertaccini, G. Paulin, M. Cavalcante, T. Fischer, S. Mach, and L. Benini, "MiniFloats on RISC-V Cores: ISA Extensions with Mixed-Precision Short Dot Products," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–16, 2024.

[24] S. Shahabuddin, M. H. Islam, M. S. Shahabuddin, M. A. Albreem, and M. Juntti, "Matrix Decomposition for Massive MIMO Detection," in *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2020, pp. 1–6.

[25] M. Bertuletti, Y. Zhang, A. Vanelli-Coralli, and L. Benini, "Efficient Parallelization of 5G-PUSCH on a Scalable RISC-V Many-Core Processor," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

NDFT: Accelerating Density Functional Theory Calculations via Hardware/Software Co-Design on Near-Data Computing System

Qingcai Jiang[1]*, Buxin Tu[1]*, Xiaoyu Hao[1], Junshi Chen[1],[2] and Hong An[1],[2],†
[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]Laoshan Laboratory, Qingdao, China

***Abstract*—Linear-response time-dependent Density Functional Theory (LR-TDDFT) is a widely used method for accurately predicting the excited-state properties of physical systems. Previous works have attempted to accelerate LR-TDDFT using heterogeneous systems such as GPUs, FPGAs, and the Sunway architecture. However, a major drawback of these approaches is the constant data movement between host memory and the memory of the heterogeneous systems, which results in substantial *data movement overhead*. Moreover, these works focus primarily on optimizing the compute-intensive portions of LR-TDDFT, despite the fact that the calculation steps are fundamentally *memory-bound*.**

To address these challenges, we propose NDFT, a Near-Data Density Functional Theory framework. Specifically, we design a novel task partitioning and scheduling mechanism to offload each part of LR-TDDFT to the most suitable computing units within a CPU-NDP system. Additionally, we implement a hardware/software co-optimization of a critical kernel in LR-TDDFT to further enhance performance on the CPU-NDP system. Our results show that NDFT achieves performance improvements of 5.2x and 2.5x over CPU and GPU baselines, respectively, on a large physical system.

***Index Terms*—Near-Data Processing, Density Functional Theory, Workload Scheduling, Hardware/Software Co-Design**

I. Introduction

First-principle calculations, also known as *ab initio* methods, predict the properties of materials by solving the electronic motion equations derived from the fundamental principles of quantum mechanics. Among these methods, density functional theory (DFT) is widely used to determine the ground-state properties of systems by focusing on electron density rather than wavefunctions. Building on DFT, **linear response time-dependent density functional theory (LR-TDDFT)** is particularly notable for its ability to accurately describe the excited states of complex systems. LR-TDDFT is extensively applied in material analysis, condensed matter physics, and quantum chemistry. However, it often encounters significant computational challenges, including high execution time, memory complexity, and frequent data accesses, which can easily lead to memory bottlenecks. Given the crucial role of LR-TDDFT in scientific research and the computational challenges it presents, exploring optimization techniques for this method has become a prominent topic in the field of high-performance computing.

Recent studies have tried to accelerate LR-TDDFT calculations using heterogeneous systems such as graphics processing units (GPUs) [20] and the Sunway architecture [22]. These architectures leverage parallel processing capabilities to execute a large number of computations simultaneously, significantly speeding up the *compute-intensive* aspects of the software. However, executing LR-TDDFT on these systems requires frequent data transfers between main memory and local memory. This overhead from this data movement can negate the performance gains achieved by heterogeneous architectures, leading to a phenomenon known as the data movement bottleneck.

Near-data processing (NDP) is a promising solution to this bottleneck in modern computing systems. By positioning computing units closer to the main memory, the NDP mechanism reduces the overhead associated with data access [28]. With recent advancements in memory technology [1]–[4], [6], [9]–[11], [13] and the growing demand from data-intensive applications [6], [7], the near-memory computing paradigm has become increasingly feasible [14], [17].

However, effectively harnessing the potential of NDP systems to accelerate LR-TDDFT calculations presents several key challenges. First, NDP systems typically operate alongside CPU systems, forming a CPU-NDP system. For LR-TDDFT calculations, which involve both compute-intensive and memory-intensive kernels, such systems require a precise scheduling and partitioning scheme to ensure efficient execution. Second, in the pseudopotential computation of LR-TDDFT (see Section III-B), each process stores its copy of the data in memory, leading to significant redundancy given the large number of NDP cores and corresponding processes. Therefore, we need to rethink the data organization of pseudopotential in the NDP scenario to mitigate this issue.

To address these challenges, **our goal** in this work is to design a partitioning and scheduling mechanism specifically tailored for DFT calculations to achieve better performance, along with a hardware/software co-design solution that optimizes the data organization and memory access in the pseudopotential calculations.

This paper makes the following key contributions:

- We conduct a detailed study of the computation and memory characteristics in the LR-TDDFT application, identifying the key performance bottlenecks in its execution and

*Equal contribution, listed alphabetically
†Corresponding Author

determining LR-TDDFT as a promising application for NDP systems.

- We design NDFT, a near-data density functional theory calculation framework, featuring a novel workload partitioning and scheduling mechanism, along with a hardware/software co-design based on a CPU-NDP heterogeneous architecture.
- We evaluate NDFT across physical systems of varying sizes and compare it with CPU and GPU baselines. The results show that NDFT achieves a performance speedup of 5.2x and 2.5x over CPU and GPU baselines on a large physical system.

II. Background

A. LR-TDDFT

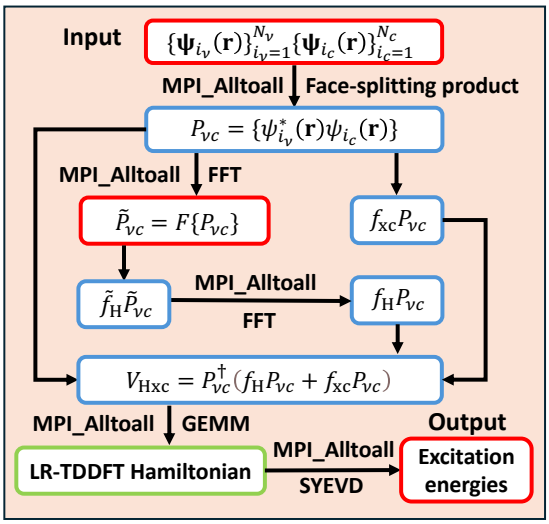


Fig. 1. Computation flowchart of LR-TDDFT. $\psi_{i_v}(\mathbf{r})$ and $\psi_{i_c}(\mathbf{r})$ stand for the valence and conduction orbitals in real space $(\{\mathbf{r_i}\}_{i=1}^{N_r}$.

LR-TDDFT [30] is a widely used computational method for studying the excited-state properties of molecules and solids in condensed matter physics, computational chemistry, and materials science. In recent years, advancements in both computational hardware and numerical algorithms have significantly improved the scalability and performance of LR-TDDFT, enabling its application to large and complex physical systems [26], [33]. As shown in Figure 1, the computation of LR-TDDFT involves several major operations: (1) fast Fourier transform (**FFT**), (2) face-splitting product, i.e., **point-to-point multiplication** between two vectors, (3) general matrix multiplications (**GEMM**), (4) **MPI_Alltoall** between all processes to transpose data, and (5) diagonalization (**SYEVD**).

Due to its scientific significance, numerous prior works have focused on accelerating the *computation-intensive* aspects of LR-TDDFT using heterogeneous architectures, such as GPUs [20], [34], the Sunway supercomputer [21], [22], and field-programmable gate arrays (FPGAs) [29]. However, accelerating LR-TDDFT on these heterogeneous architectures presents two major problems: (1) These approaches primarily target the *compute-intensive* components, neglecting the *memory-intensive* parts, which constitute a significant portion of LR-TDDFT's overall execution time; (2) Executing kernels on heterogeneous architectures requires frequent data transfers between the host device's memory and the memory of the heterogeneous device, resulting in substantial bottlenecks.

B. NDP Architecture

The fundamental idea of NDP is to place computing logic close to where data resides, offering significant potential for improving the performance of *memory-intensive applications*. NDP can be implemented by integrating processing logic within DRAM banks [1], [13], in the logic layer of 3D-stacked memory [2], [6], [10], or within the buffer chip of DDR-DIMMs [3], [4], [9], [11]. As shown in Figure 2, in such architectures, the bottom-most layer, known as the logic layer, can be equipped with **wimpy cores** that perform simple, parallelizable tasks close to the data. These cores leverage the high internal bandwidth and low data movement latency of the memory stacks to improve efficiency. Typical 3D-stacked NDP systems, such as Hybrid Memory Cube (HMC) [8] and High Bandwidth Memory (HBM) [18], [19], consist of multiple memory stacks interconnected in a memory network. By bringing computation closer to the data, NDP systems take advantage of the high bandwidth within the memory structure and reduce data access latency.

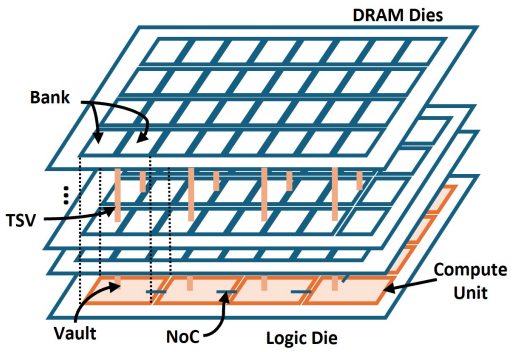


Fig. 2. An example of 3D-stacked memory.

C. CPU-NDP Hybrid Execution

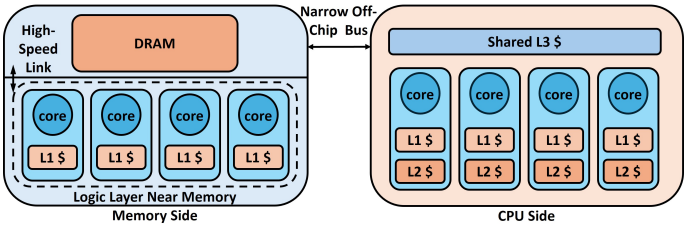


Fig. 3. A high-level CPU-NDP architecture.

In a typical configuration, NDP cores operate alongside CPU cores within a computing system, forming what we refer to as the CPU-NDP architecture in this paper, as illustrated in Figure 3. This architecture includes two types of computing units: the CPU, which excels at complex, compute-intensive tasks but suffers from high-latency and low-bandwidth memory access, and the NDP cores, which are more numerous and benefit from lower-latency, higher-bandwidth memory access, but feature simpler logic. Thus, CPU cores are suitable for *compute-intensive* workloads, while NDP cores handle

memory-intensive tasks more efficiently. In such systems, a key challenge is determining which code regions of a program should be offloaded to NDP for optimal performance. Several prior works have attempted to solve this problem [12], [23], [35]. However, none of these approaches have been applied to exploit the performance potential of DFT applications in CPU-NDP systems.

III. Motivational Study

In this section, we conduct a detailed study on several key characteristics of LR-TDDFT, aiming to identify the key bottlenecks in CPU-NDP hybrid execution and demonstrate its potential for performance improvement on CPU-NDP heterogeneous architecture.

A. LR-TDDFT kernel characteristics

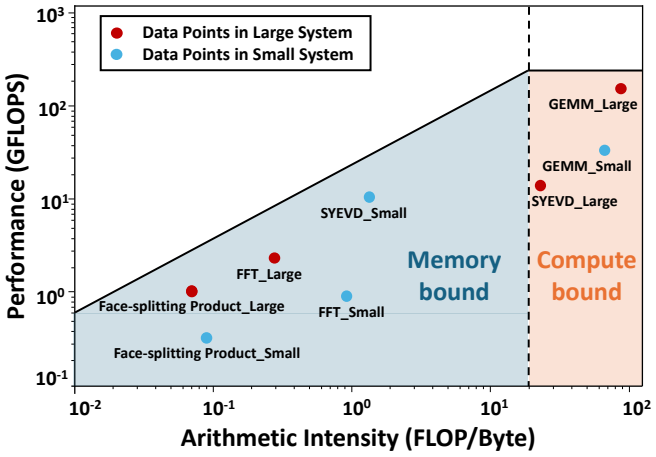


Fig. 4. Roofline model analysis of LR-TDDFT kernels across two different system sizes.

We analyze the performance characteristics of representative kernels in LR-TDDFT calculations on the Roofline model, as introduced in Section II-A, using two different physical system sizes: Si_64 (small system) and Si_1024 (large system). The Roofline model [36] integrates floating-point operations (FLOPs) and arithmetic intensity (AI) to assess the computational and memory characteristics of a given kernel. We use Intel VTune Profiler [16] for the analysis, with system configurations described in Section V. From the results shown in Figure 4, we make three key observations: (1) LR-TDDFT calculations are fundamentally **memory-bound**: most computing kernels in LR-TDDFT fall within the memory-bound region, and the MPI_Alltoall operations used in LR-TDDFT introduce a large amount of irregular memory accesses [22]. (2) Different kernels in LR-TDDFT calculations exhibit distinct compute-bound or memory-bound characteristics: FFT kernels are memory-bound, while GEMM kernels are compute-bound across different physical system sizes. (3) The compute- or memory-bound nature of LR-TDDFT kernels depends on the system size: for large physical systems, the SYEVD operation is compute-bound, whereas, for small systems, SYEVD is memory-bound; GEMM becomes more compute-bound as the system size increases.

Based on these observations, we conclude that LR-TDDFT is well-suited for CPU-NDP acceleration due to its diverse kernel characteristics. To fully leverage the strengths of both CPU and NDP cores, the system requires an intelligent scheduling mechanism that can dynamically allocate tasks to the appropriate computing units based on their characteristics. For example, memory-bound kernels such as FFT could benefit from execution on NDP units optimized for memory access, while compute-bound kernels like GEMM and SYEVD (in large systems) would perform better on CPU cores, which excel at handling complex computations. However, designing a precise task scheduling framework for LR-TDDFT in a CPU-NDP system is not trivial, as it faces two main challenges: (1) It must account not only for the suitability between kernels and computing units but also for the *data movement overhead* between the two types of computing units. The data required by the kernels must also be scheduled accordingly to the appropriate execution units. (2) The selection of offloading granularity, ranging from fine-grained instructions to basic blocks or entire kernels, requires careful consideration.

B. Analysis of pseudopotential calculations

Pseudopotentials are commonly used in DFT calculations to simplify the treatment of inner electrons, as the outer valence electrons primarily determine an atom's physical and chemical properties. In LR-TDDFT, pseudopotentials simulate the influence of the atomic nucleus and inner electrons. During this procedure, each wavefunction is updated based on the pseudopotential information associated with each atom. In a multi-process execution, each process must maintain a complete copy of the pseudopotential information in its memory space. However, in systems with numerous cores, such as NDP architectures [12], [32], these redundant data copies can lead to significant memory inefficiency.

TABLE I
Memory Footprint of Pseudopotentials in CPU and NDP Systems

Memory Footprint	size (GB)	Percentage (%)
NDP in Small system	4.43	6.92
CPU in Small system	1.84	2.88
NDP in Large system	35.3	55.15
CPU in Large system	13.8	21.56

We profile the memory footprint during LR-TDDFT calculations on typical NDP and CPU systems using two physical systems of different scales, where the NDP system and the CPU system are isolated. The configuration is detailed in Section V. As shown in Table I, our observations are as follows: (1) In the small physical system, the memory footprint of pseudopotentials in NDP execution occupies 6.92% of the system memory, which is 140.2% higher than that in CPU execution; (2) In the large physical system, the memory footprint reaches 55.15%, representing an increase of 155.7% compared to CPU execution. We conclude that the large number of NDP cores leads to an increase in parallel processes, resulting in pseudopotentials consuming a significant amount of memory space. Moreover, in more complex physical systems, the traditional

per-process pseudopotential approach causes an out-of-memory (OOM) problem. Therefore, optimizing the data structure of pseudopotential information in memory is critical to reducing the substantial memory footprint in the CPU-NDP architecture.

IV. NDFT Design

In this section, we elaborate on the design of NDFT, aiming to maximize the performance potential of the CPU-NDP system for DFT calculations. This includes a novel workload partitioning and scheduling mechanism, which is introduced in Section IV-A, as well as a hardware/software co-design strategy to better orchestrate an important computational scheme in LR-TDDFT calculations, which is covered in Section IV-B and IV-C, respectively.

A. Workload Partition and Scheduling

1) Offload Granularity: To provide a better offloading granularity for LR-TDDFT, it is crucial to understand the costs associated with scheduling contiguous code segments to different computing units. We identify two primary sources of overhead: (1) The *data transfer overhead*, which arises from maintaining the consistency of data across two computing units. This overhead is proportional to the amount of data that needs to be transferred. (2) The *context switch overhead*, which involves synchronizing the context (e.g., register values) between threads on different units, is a constant time cost.

In this work, we choose *function-level* offloading granularity based on two key observations: (1) Dividing the program into too many small segments introduces significant offloading overhead. (2) Most functions in LR-TDDFT exhibit consistent compute/memory characteristics throughout their execution. For example, GEMM and FFT maintain the same computing and memory access patterns across the entire function.

2) Cost-Aware Offloading mechanism: To fully leverage the performance benefits of a heterogeneous CPU-NDP system, we design a kernel offloading mechanism in NDFT based on a static code analyzer (SCA) [15], [25]. The SCA analyzes the intrinsic properties of a given code segment, such as estimated execution time, memory access patterns, and instruction dependencies. These insights help inform decisions on whether each code segment is compute- or memory-bound, and provide an estimation of the data transfer overhead when scheduling different code segments across CPU and NDP units. In this work, we leverage the SCA to profile the compute and memory intensity of each function and use this information, along with a cost model that accounts for the overhead of scheduling between CPU and NDP cores, to formulate a cost-aware offloading mechanism.

$$\text{Scheduling Overhead} = \sum_{i \in \text{NDP}} \sum_{j \in \text{CPU}} (DT(i,j) + CXT) \quad (1)$$

By evaluating the suitability of each function for execution on either CPU or NDP cores, the SCA allows us to determine whether it is more efficient to offload the function to the NDP or retain it on the CPU. The overall scheduling strategy then incorporates the scheduling overhead, as shown in (1), which is defined as the sum of data transfer (DT) overhead and context switch (CXT) overhead between the CPU and NDP cores. This approach enables a cost-aware offloading mechanism that balances the performance gains from offloading with the associated overheads, optimizing the performance of the heterogeneous system.

B. Optimization of Pseudopotential Algorithm

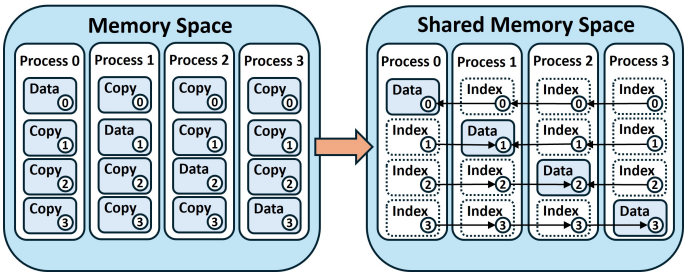


Fig. 5. The data structure optimization to eliminate the data redundancy of pseudopotential.

To reduce the memory footprint in LR-TDDFT and enhance the scalability for large physical systems, we propose an optimization based on an improved pseudopotential data structure. As shown in Figure 5, each process stores **only a portion of the pseudopotentials**, along with indices pointing to the pseudopotentials in other processes, accessing the shared data via these indices. Compared to the original approach, our optimization enables seamless data sharing between processes, significantly reducing the memory footprint and making LR-TDDFT more suitable for our CPU-NDP heterogeneous design.

In our optimization, we reduce the memory overhead of each process by addressing the complexity of each atom's pseudopotential data, which includes arrays of integers and double-precision floating-point matrices, through the design of a **shared block** to efficiently manage and share this information. This data structure facilitates unified management of pseudopotential data and simplifies data sharing. As shown in Algorithm 1, the highlighted gray section represents the main optimization of the algorithm: (1) In lines 7-9, the algorithm reorganizes the pseudopotential information of each atom into a shared block and writes it into shared memory. (2) In line 13, the algorithm retrieves the addresses of pseudopotential data in shared memory based on the distribution of the shared block. (3) In line 19, the algorithm accesses the data from shared memory for wavefunction updating.

Although this approach significantly reduces memory usage, it introduces the need for inter-process data communication, leading to some performance overhead. We design a novel shared memory hardware structure to mitigate this overhead, detailed in Section IV-C.

C. Shared Memory Design for Pseudopotential

As we improve the data structure of the pseudopotential in each process, data communication is needed to synchronize the pseudopotential. To mitigate this overhead, we design shared memory based on scratchpad memory (SPM) for processes to share the pseudopotential information within the same stack, and a hierarchical communication scheme to manage the communication between the shared memory of different stacks.

Algorithm 1 Pseudopotential Algorithm in LR-TDDFT, with the gray background highlighting our optimizations.

Input: Pseudopotential
Output: Wavefunction updated using pseudopotential

```
1:  for each local atom do
2:      calculate pseudopotential
3:  end for
4:  for each global atom do
5:      if this process holds the information then
6:          for each atom's pseudopotential do
7:              calculate the data size
8:              allocate a continuous space in shared memory
9:              write local pseudopotential information as
                a block into shared memory
10:         end for
11:     else
12:         for each shared memory block do
13:             obtain the address of the shared block
14:         end for
15:     end if
16: end for
17: for each local wavefunction do
18:     for each pseudopotential of every atom do
19:         access pseudopotential via shared block address
20:         apply pseudopotential to the wavefunction
21:     end for
22: end for
```

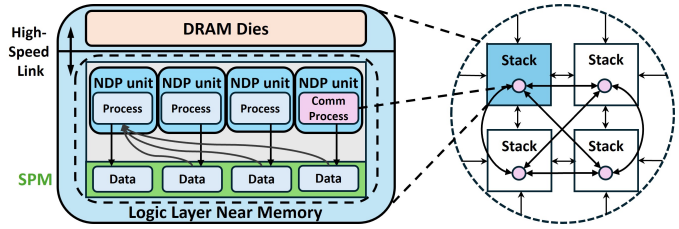


Fig. 6. Shared Memory Hardware Design and Hierarchical Communication Scheme. Green color highlights the shared memory architecture.

SPM-based Shared Memory. The left part of Figure 6 shows our shared memory hardware design. We design an SPM in the logic layer of each memory stack to enable the shared memory functionality. The processes within the same stack can access local pseudopotentials and also access the pseudopotentials of other processes in SPM-based shared memory. The high read/write speed and low access cost of SPM [5] allow these processes to efficiently access pseudopotentials, avoiding the expensive memory access in DRAM.

In addition, we develop a set of communication APIs to support shared memory functionality. As shown in Table II, these APIs implement multiple synchronization semantics, providing the flexibility and comprehensiveness needed for essential communication primitives. For example, `NDFT_Alloc_Shared` enables NDP units to allocate memory for pseudopotential data in `sharedBL`. The function's parameters include the pseudopotential data (`pseu_info`) and the NDP unit ID (`stID* id`), and it returns a `sharedBL`. The definition of `sharedBL` corresponds to the shared block introduced in Section IV-B. We also design primitives such as `NDFT_Read` and `NDFT_Write` to facilitate reading and writing pseudopotentials within the shared block. By managing pseudopotentials with SPM-based shared memory, our design reduces communication overhead between NDP units within the same stack. To further enhance scalability, we implement a hierarchical communication scheme.

TABLE II
PROGRAMMING INTERFACES OF NDFT

NDFT's Message Programming Interface(i.e., API)
sharedBL NDFT_Alloc_Shared (Var* pseu_info, stID* id);
void NDFT_Read (sharedBL* data, void* address, int length);
void NDFT_Write (sharedBL* data, void* address, int length);
void NDFT_Read_Remote (sharedBL* data, void* address, int length, stID* source_id, stID* dest_id);
void NDFT_Write_Remote (sharedBL* data, void* address, int length, stID* source_id, stID* dest_id);
void NDFT_Broadcast (sharedBL* data);
.....

Hierarchical Communication Scheme. For pseudopotential data communication between different stacks, we designate one NDP unit per stack as a communication arbiter, which runs a *comm process* (indicated in pink in Figure 6) to manage inter-stack communication. When a process requests data from another stack via `NDFT_Read_Remote`, it submits the request to the *comm process* within its own stack. The `dest_id` parameter specifies the stack ID from which the data is needed. After exchanging data with the communication process of the destination stack, the local *communication process* writes the data into shared memory and returns the index to the requesting process. This hierarchical design acts as a filter, maximizing intra-stack communication and only transmitting essential data across stacks, thereby reducing overall communication overhead.

V. EVALUATION METHODOLOGY

System model. We implement NDFT in zsim [31], a fast and accurate Pin-based simulator, and use Ramulator [24] as the memory simulator. Table III summarizes the NDFT's CPU-NDP system configurations.

TABLE III
CPU-NDP SYSTEM CONFIGURATION

CPU	8 General purpose core; 3 GHz, 4-way superscalar; 32 KB L1I/D, 256 KB L2, 2 MB L3
NDP	8 NDP units per stack, 2GHz, in order; 64 GB in total, 512 MB per unit; 2 GHz, 2 cores per NDP unit, 32KB L1I/D Shared Memory: 16 KB per core, 256 KB per stack
Memory	HBM2, 4 × 4 stacks in mesh, 8 channels per stack; 128-bit bus width, 1000Mhz, 64 GB capacity

Baseline Architectures. We compare NDFT with two baselines: (1) CPU: 2 CPUs of Intel Xeon E5-2695@2.40 GHz, 12 cores per socket, with 64 GB DDR4 memory; (2) GPU: 2 GPUs of NVIDIA V100 in DGX-1 server.

Physical Systems. We use various crystal silicon systems with 16, 32, 64 (small system), 128, 256, 1024 (large system), and 2048 silicon atoms, denoted as Si_16, Si_32, Si_64, Si_128, Si_256, Si_1024, and Si_2048, respectively.

VI. Evaluation Result

A. Performance Analysis on Execution Time

Figure 7 shows the performance on CPU, GPU, and NDFT, along with the time distribution across different kernels in 2 different sizes of physical systems. The execution time breakdown includes the execution time of FFT, point-point multiplication, Global Communication (Global Comm) phase, SYEVD, etc. The NDFT design includes the additional *scheduling* overhead between CPU and NDP, as mentioned in Section IV-A. We compare NDFT with both GPU and CPU baselines, and obtain the following evaluation results:

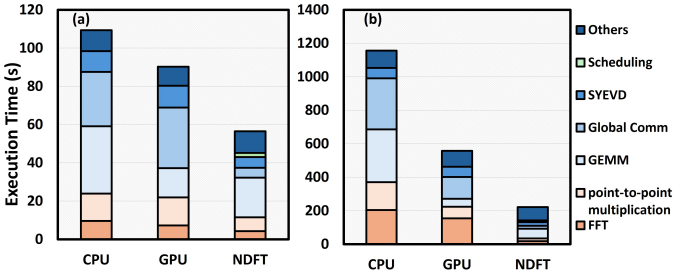


Fig. 7. Performance comparison between CPU, GPU, and NDFT design using a small physical system (a) and a large physical system (b).

Comparison with CPU. (1) Despite the scheduling overhead, NDFT achieves 1.9x and 5.2x speedups in small and large physical systems, respectively, thanks to our scheduling mechanism and software/hardware co-design. (2) The performance of memory-intensive kernels is significantly improved; for example, FFT achieves an 11.2x speedup in the large system, while the Face-splitting Product achieves a 1.99x speedup in the small system.

Comparison with GPU. (1) NDFT achieves 1.6x and 2.5x speedups compared to the GPU baseline in small and large physical systems, respectively. (2) While compute-bound kernels like GEMM on the GPU outperform those on NDFT by 35.9% and 22.2% in the two physical systems, respectively, memory-bound kernels show significant improvements with NDFT, achieving 2.1x and 5.2x speedups in the small and large physical systems, respectively.

Other Discussion. (1) The scheduling overhead accounts for only 3.8% and 4.9% in small and large physical systems, respectively, demonstrating that NDFT's scheduling strategy carefully keeps this overhead minimal while improving kernel performance. (2) We study the memory footprint of pseudopotential information in NDFT. We observe that, in the large physical system, NDFT reduces memory footprint by 57.8% compared to NDP in Table I, bringing it close to CPU execution (1.08x). Global Comm only increases by 3.2% over the GPU baseline, showing that NDFT's hardware/software co-design resolves the OOM issue in NDP systems with low communication overhead.

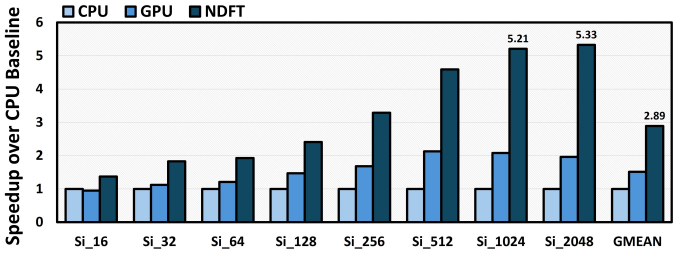


Fig. 8. Speedup provided by NDFT, GPU over CPU baseline in several physical system scales.

B. Scalability Analysis

To explore NDFT's performance across different physical system scales, we conduct a study ranging from Si_16 to Si_2048. Figure 8 demonstrates that as the size of the physical system increases, NDFT's performance advantage becomes more pronounced (up to 5.33x speedup in Si_2048). We conclude that NDFT improves the performance of LR-TDDFT calculations in most cases, showing NDFT's great potential for addressing large-scale computational problems in the field of high-performance computing.

VII. Related Work

To our knowledge, NDFT is the first work to accelerate DFT calculations through optimizations of a general NDP architecture. A prior work [27] proposes a heterogeneous approach focused on alleviating data movement bottlenecks in quantum chemistry simulations based on *ab initio* methods. That work analyzes two specific components—FFT and time-consuming loops—proposing a custom hardware design tailored to these kernels. In contrast, NDFT emphasizes task scheduling and hardware-software co-design, offering a more comprehensive solution within general-purpose NDP-CPU heterogeneous systems, thereby providing greater applicability and scalability.

VIII. Conclusion

In this work, we propose NDFT, a Near-Data Density Functional Theory framework to address performance bottlenecks in LR-TDDFT calculations on CPU-NDP heterogeneous systems. NDFT involves a task partitioning and scheduling mechanism that efficiently assigns memory- and compute-bound kernels, minimizing data movement and fully leveraging CPU-NDP heterogeneous architectures, and a hardware-software co-design optimizing memory access for pseudopotential calculations, solving OOM issues in NDP systems. Evaluations show speedups of 5.2x and 2.5x over CPU and GPU baselines in a large physical system, as well as NDFT's scalability across different system sizes.

Acknowledgment

We thank the reviewers of DATE 2025 and DAC 2025 for their feedback. This work is partly supported by Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB0500102). The computing resources are financially supported by Laoshan Laboratory (LSKJ202300305).

REFERENCES

[1] S. Aga, N. Jayasena, and M. Ignatowski. Co-ml: a case for co llaborative ml acceleration using near-data processing. In *Proceedings of the International Symposium on Memory Systems*, pages 506–517, 2019.

[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 105–117, 2015.

[3] B. Asgari, R. Hadidi, J. Cao, S.-K. Lim, H. Kim, et al. Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 908–920. IEEE, 2021.

[4] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim. Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems. In *2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[5] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*, pages 73–78, 2002.

[6] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, et al. Google workloads for consumer devices: Mitigating data movement bottlenecks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 316–331, 2018.

[7] C. P. Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[8] H. M. C. Consortium. Hmc specification 2.0, 2014.

[9] G. Dai, Z. Zhu, T. Fu, C. Wei, B. Wang, X. Li, Y. Xie, H. Yang, and Y. Wang. Dimmining: pruning-efficient and parallel graph mining on near-memory-computing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 130–145, 2022.

[10] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 283–295. IEEE, 2015.

[11] S. Feng, X. He, K.-Y. Chen, L. Ke, X. Zhang, D. Blaauw, T. Mudge, and R. Dreslinski. Menda: A near-memory multi-way merge solution for sparse transposition and dataflows. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 245–258, 2022.

[12] N. M. Ghiasi, N. Vijaykumar, G. F. Oliveira, L. Orosa, I. Fernandez, M. Sadrosadati, K. Kanellopoulos, N. Hajinazar, J. G. Luna, and O. Mutlu. Alp: Alleviating cpu-memory data movement overheads in memory-centric systems. *IEEE Transactions on Emerging Topics in Computing*, 11(2):388–403, 2022.

[13] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie. ipim: Programmable in-memory image processing accelerator using near-bank architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 804–817. IEEE, 2020.

[14] X. Hu, D. Stow, and Y. Xie. Die stacking is happening. *IEEE micro*, 38(1):22–28, 2018.

[15] Intel. Intel architecture code analyzer user's guide, 2017.

[16] Intel Corporation. *Intel VTune Profiler*, 2024. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html.

[17] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *2012 symposium on VLSI technology (VLSIT)*, pages 87–88. IEEE, 2012.

[18] JEDEC. High Bandwidth Memory (HBM) DRAM. https://www.jedec.org/standards-documents/docs/jesd235a, 2021.

[19] JEDEC Solid State Technology Assn. JESD23-5D: High Bandwidth Memory (HBM) DRAM Standard, March 2021.

[20] W. Jia, L.-W. Wang, and L. Lin. Parallel transport time-dependent density functional theory calculations with hybrid functional on summit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–23, 2019.

[21] Q. Jiang, Z. Cao, J. Chen, X. Qin, W. Hu, H. An, and J. Yang. Pwdft-sw: Extending the limit of plane-wave dft calculations to 16k atoms on the new sunway supercomputer. *arXiv preprint arXiv:2406.10765*, 2024.

[22] Q. Jiang, Z. Cao, X. Cui, L. Wan, X. Qin, H. Cao, H. An, J. Chen, J. Liu, W. Hu, et al. Extending the limit of lr-tddft on two different approaches: Numerical algorithms and new sunway heterogeneous supercomputer. *Parallel Computing*, 120:103085, 2024.

[23] Q. Jiang, S. Tan, J. Chen, and H. An. A^3pim: An automated, analytic and accurate processing-in-memory offloader. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[24] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer architecture letters*, 15(1):45–49, 2015.

[25] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *CGO*, 2004.

[26] J. Liu and W. Liang. Analytical hessian of electronic excited states in time-dependent density functional theory with tamm-dancoff approximation. *The Journal of chemical physics*, 135(1), 2011.

[27] Z. Liu, Z. Xie, W. Dong, M. Yuan, H. You, and D. Li. A heterogeneous processing-in-memory approach to accelerate quantum chemistry simulation. *Parallel Computing*, 116:103017, 2023.

[28] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

[29] A. Ramaswami, T. Kenter, T. D. Kühne, and C. Plessl. Efficient ab-initio molecular dynamic simulations by offloading fast fourier transformations to fpgas. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 353–354. IEEE, 2020.

[30] E. Runge and E. K. Gross. Density-functional theory for time-dependent systems. *Physical review letters*, 52(12):997, 1984.

[31] D. Sanchez and C. Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news*, 41(3):475–486, 2013.

[32] B. Tian, Q. Chen, and M. Gao. Abndp: Co-optimizing data access and load balance in near-data processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 3–17, 2023.

[33] L. Wan, X. Liu, J. Liu, X. Qin, W. Hu, and J. Yang. Hybrid mpi and openmp parallel implementation of large-scale linear-response time-dependent density functional theory with plane-wave basis set. *Electronic Structure*, 3(2):024004, 2021.

[34] L. Wang, Y. Wu, W. Jia, W. Gao, X. Chi, and L.-W. Wang. Large scale plane wave pseudopotential density functional theory calculations on gpu clusters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2011.

[35] Y. Wei, M. Zhou, S. Liu, K. Seemakhupt, T. Rosing, and S. Khan. Pimprof: An automated program profiler for processing-in-memory offloading decisions. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 855–860. IEEE, 2022.

[36] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

FastPath: A Hybrid Approach for Efficient Hardware Security Verification

Lucas Deutschmann* — *RPTU Kaiserslautern-Landau* — Kaiserslautern, Germany — lucas.deutschmann@rptu.de

Andres Meza* — *UC San Diego* — San Diego, USA — anmeza@ucsd.edu

Dominik Stoffel — *RPTU Kaiserslautern-Landau* — Kaiserslautern, Germany — dominik.stoffel@rptu.de

Wolfgang Kunz — *RPTU Kaiserslautern-Landau* — Kaiserslautern, Germany — wolfgang.kunz@rptu.de

Ryan Kastner — *UC San Diego* — San Diego, USA — kastner@ucsd.edu

***Abstract*—Many verification methods have been proposed to detect microarchitectural information leakage in response to the surge of security breaches in hardware designs. These sophisticated efforts have gone a long way toward preventing attackers from breaking the system's confidentiality. However, each approach has its own set of weaknesses: it may not be scalable enough, exhaustive enough, flexible enough to meet changing requirements or fit well into existing verification flows.**

We propose FastPath, a hybrid verification methodology that combines the efficiency of simulation with the exhaustive nature of formal verification. FastPath employs a structural analysis framework to automate the method further. Our experimental results compare FastPath to a state-of-the-art formal approach, showing a significant reduction in manual effort while achieving the same level of exhaustive confidence. We also discovered and contributed a fix for a previously unknown leak of internal operands in cv32e40s, a RISC-V processor intended for security applications.

***Index Terms*—Hardware Security, Information Flow Tracking, Simulation, Formal Verification, Data-Oblivious Computing.**

I. Introduction

A seemingly endless flood of timing side channel attacks [1], [2], [3], [4], [5] severely disrupted the view of hardware as a root of trust. This necessitates rethinking architectures and fortifying them to meet ever-changing security requirements. In response, software and hardware communities are actively proposing novel mitigation techniques [6], [7], [8], [9], [10], [11] to address these requirements.

Most countermeasures against microarchitectural timing side channels demand that specific basic hardware operations are *data-oblivious*. Data-obliviousness requires these hardware primitives to process data without any measurable data-dependent side effects, e.g., the operation's timing must be independent of its inputs. A prominent software paradigm, called *Constant-Time Programming* [12], [8], [13], assumes that simple instructions, such as an addition, do not leak data via timing side channels. More complex instructions, such as a division or floating point operations, are assumed to be data-dependent and replaced with data-oblivious primitives [14]. Making matters worse, whether an instruction is data-oblivious or not depends on the underlying microarchitecture. Intel [15], ARM [16] and RISC-V [17] have therefore added ways to refine the instruction set architecture (ISA) with information about the data-obliviousness of individual instructions.

However, recent insights [18], [5] have shown that, despite these precautions, sophisticated optimizations in modern processors can invalidate these assumptions. Chen et al. [5] exploit an optimization feature of recent Apple devices, called data memory-dependent prefetcher, to attack constant-time cryptographic routines by tricking the prefetcher into misinterpreting secret data as addresses. This attack demonstrates that it is insufficient to simply assume a certain behavior of the hardware layer. Instead, it is necessary to provide *exhaustive* security guarantees upon which software countermeasures can be built.

As part of the effort to restore trust in hardware, we propose FastPath, a hybrid security verification approach. The method exhaustively verifies *non-interference* [19], [20] properties, e.g., the property that confidential information does not leak into publicly visible states of the system. In this paper, we specifically verify that hardware operates in a data-oblivious manner. However, FastPath is not limited to this threat model.

The key observation behind FastPath is that recent formal methods [21], [22] achieve high scalability through a *semantic partitioning* of the computational problem. Rather than partitioning the design *structurally* into paths to be checked formally, these methods logically decompose the global proof problem. While the new techniques scale even to complex out-of-order processors, deriving this logical partitioning can require a considerable amount of manual refinement.

With FastPath, we identify and leverage a unique synergy between these formal methods and simulation-based security verification: simulating a design with Information Flow Tracking (IFT) [23] efficiently derives the bulk of the problem partitioning required by the formal methods, eliminating most manual effort. Formal verification, on the other hand, complements the weakness of simulation in finding subtle leaks in corner-case scenarios. To round out our hybrid approach, FastPath employs static structural analysis to detect when simulation and formal are not necessary and to generate property templates when they are needed. This novel way of combining these methods creates a highly automated and scalable, yet exhaustive verification approach.

In summary, this paper makes the following contributions:

- We propose FastPath, a hybrid verification approach for hardware security that leverages the complementary nature of simulation and formal methods (Sec. IV). Instead of combining the two approaches in a straightforward way by applying a formal analysis to certain paths not covered during simulation, our method uses a global property formulation and a seamless communication of intermediate results between the different methods.
- We integrate a structural analysis and IFT framework called HyperFlow Graph (HFG) [24] (Sec. III-A) into FastPath. The structural information can be used to exclude trivial cases where no syntactic information flow paths exist and to generate templates for the formal properties.
- We apply FastPath to a variety of accelerator and processor designs (Sec. V) and compare the manual effort with a formal-only approach [22]. Our case studies show a 36% to 100% reduction in manual refinement. In an experiment with the cv32e40s [25] processor, FastPath was able to detect a previously unknown leak of internal operands.

*Both authors contributed equally to this research.

This work has been supported in part by funding from the Agentur für Innovation in der Cybersicherheit GmbH (Cyberagentur), in part by BMBF ZuSE (Scale4Edge), 16ME0122K-16ME0140+16ME0465, and in part by Intel Corp. (Scalable Assurance).

II. Threat Model

FastPath verifies *non-interference* properties [19], [20]. *Non-interference* divides the system into *high*-security and *low*-security domains and can be used to model a variety of different threat models. Confidentiality, for example, requires that information never leak from *high* to *low*. For the purposes of this paper, we adopt the threat model and definitions from [22]. In particular, we consider a hardware system that processes confidential input data. The attacker exploits violations of data-obliviousness in the hardware and observes any data-dependent side-effect, i.e., any deviation in the control behavior caused by the values of the operands being processed. This is usually reflected in a difference in timing, resulting in different values at the control related output of the system. Note that our method is not limited to this threat model and can be extended to other attack scenarios.

Our threat model is described more precisely as follows: We model a hardware system as a standard Mealy-type Finite-State Machine (FSM) $M = (I, O, S, S_0, \delta, \lambda)$, where we denote sets of input symbols I, output symbols O, states S, and initial states $S_0 \subseteq S$. The state transition function is given by $\delta : S \times I \mapsto S$ and the output function is given by $\lambda : S \times I \mapsto O$. In addition, we denote the set of Register-Transfer Level (RTL) input signals by X, output signals by Y, and state signals by Z. We partition the set of input signals X (output signals Y) into disjoint sets of control inputs X_C (control outputs Y_C) and data inputs X_D (data outputs Y_D). In this threat model, as described in previous works [26], [22], [27], [28], the system processes the *high*-security data inputs X_D, while the attacker is able to observe the *low*-security control outputs Y_C. If a data-dependent control behavior is observed, such as a different timing of a handshake signal (e.g., `valid`), an attacker can infer information about the data being processed. This breaks confidentiality and can, for example, severely weaken encryption techniques. This security threat is eliminated if the design is proven to be *data-oblivious*.

Definition 1 (*Data-Obliviousness*)**.** A hardware module is called *data-oblivious* if, under a set of system-level constraints, the sequence of values at its control outputs Y_C is uniquely determined by the sequence of values at its control inputs X_C. In other words, its control inputs X_D have no influence on its control outputs Y_C.

Data-obliviousness of hardware designs is the verification target pursued throughout this paper.

III. Background

A. HyperFlow Graph

A HyperFlow Graph (HFG) [24], [29] is a graph-based intermediate representation which models how information flows in a hardware design. Derived automatically from a static analysis of RTL code, an HFG $G(N, E)$ contains a set of nodes N and a set of directed, labeled edges E. An HFG node $n_x \in N$ represents a unique hierarchical design signal `sig_x`. An HFG edge $e(ui, n_s, n_d, C) \in E$ represents a flow scenario in which information flows, explicitly or implicitly, from a source signal `sig_s` to a destination signal `sig_d`. Each edge can be precisely mapped to an originating RTL code construct. If the originating code construct has no guarding conditions ($C = \emptyset$), then the flow is always "active". Otherwise, the flow is only active if all guarding conditions $c \in C$ are simultaneously satisfied. Note that multiple edges are allowed between two nodes so each edge also has a unique identifier ui.

HFGs enable a variety of queries that produce security-relevant information. In this work, we primarily utilize the HFG path query $q(n_s, n_d)$ which returns a set of HFG paths P where each path $p \in P$ comprises a finite sequence of HFG edges $p = (e_1, \ldots, e_k)$ that could *potentially* enable information flow from the source signal `sig_s` to the destination signal `sig_d`. Emphasis is placed on the *potential* nature of the returned HFG paths due to the possibility for false positives. By design, the static analysis employed in the HFG construction process is intended to enable speed and scalability, but this comes at the price of producing an over-approximation of the information flow in the system. This means that the HFG path query can produce false positives but it will never produce false negatives. In other words, if there are no HFG paths ($P = \emptyset$) connecting the source signal `sig_s` to the destination signal `sig_d`, then it is guaranteed that the source signal `sig_s` cannot influence or leak information to the destination signal `sig_d`. However, if there are any HFG paths ($P \neq \emptyset$), then each path $p \in P$ has the potential to be an unrealizable sequence of design states (a false positive) and no guarantees on signal influence can be made without further analysis. As discussed in Sec. IV, our hybrid verification strategy employs simulation-based IFT and formal verification to address this issue.

B. Information Flow Tracking

Hardware Information Flow Tracking (IFT) [23] is a powerful verification strategy that allows designers to reason about information movement through a hardware design. Hardware IFT properties can be used to verify trace and hyperproperties related to confidentiality, integrity, and availability. These properties can specify where, when, and how information should or should not flow from source signals to destination signals, with relatively few operators and expressions. IFT properties can be written as assertions using the no-flow operator (`=/=>`) as follows: {`src_sigs`} `=/=>` {`dst_sigs`}. Hardware IFT tools determine whether the hardware adheres to the properties.

In simulation-based IFT, a given design is instrumented with additional logic (i.e., an IFT circuit) that tracks information flow by calculating and updating security labels (e.g., HIGH and LOW). An IFT-enhanced simulation produces a simulation trace with functional values and security labels for all signals. All design signals are initially labeled LOW, except for the source signals from which information is being tracked, which are labeled HIGH. Throughout the simulation, the IFT circuit updates signal label values considering the functional values from the simulation testbench and security labels. Any signal that starts with a LOW label and eventually reaches a HIGH label has received information from a source signal, i.e., the source signal is affecting it.

The ability to determine signal influence from merely tainting a source signal as HIGH is extremely valuable when dealing with large, complex designs. However, IFT-enhanced simulations are completely dependent on the underlying testbench to produce a property-relevant information flow. As a result, IFT properties that could be falsified may not be falsified due to an insufficient testbench. FastPath overcomes issues related to exhaustiveness via its formal verification step (Sec. IV).

C. Unique Program Execution Checking

Unique Program Execution Checking (UPEC) [21] is a formal verification methodology to exhaustively detect the propagation of confidential (or malicious) information at the RTL. The approach was originally created to detect transient execution side channel (TES) attacks, but has since been extended to cover a variety of different threat models. According to our threat model (see Sec. II), we focus on UPEC for data-independent timing (UPEC-DIT) [22].

UPEC is based on a technique called Interval Property Checking (IPC) [30] and uses a 2-safety computational model, i.e., the design under verification (DUV) is instantiated twice. At the start of the proof, each signal is initialized with the same value as its counterpart in the other instance. The only discrepancy between both instances is the information to be tracked, usually a proprietary secret that should not be revealed to the attacker. Any malicious information flow is detected by following the propagation of this difference through the system. In addition, IPC allows the equal initial state to be *symbolic*, and thus implicitly models every possible history of the system. Therefore, UPEC can "fast-forward" to the point where the propagation occurs, allowing scalability to complex systems.

UPEC-DIT is a variant of UPEC that can exhaustively verify the data-obliviousness of hardware systems according to Def. 1. In its computational model, the control inputs X_C are constrained to be equal between the two instances, while the data inputs X_D remain unconstrained. Also, both instances start from the same symbolic state. UPEC-DIT then verifies that for any arbitrarily long sequence of inputs, the control outputs Y_C of the two instances never diverge, certifying that the control behavior is independent of the processed data. The approach achieves scalability by an iterative partitioning of the internal state-holding signals based on counterexamples. Every signal declared as data is allowed to assume different values in the following iteration of the algorithm, thus implicitly modeling any previous propagation path. This process is repeated until a malicious propagation is found, or until a fixed point is reached where no new propagation occurs.

The exhaustive nature of UPEC allows to provide formal security guarantees, and thus achieve a much higher level of confidence compared to simulation-based approaches. However, the iterative manual inspection of counterexamples can be tedious and requires a certain level of design and verification expertise. In FastPath, we offload a majority of the manual work to fully-automated structural analysis and IFT-based simulation. We then leverage the intermediate simulation results to perform a UPEC-based proof, achieving the same level of exhaustiveness as the original approach combined with an improved scalability.

IV. Methodology

Fig. 1 gives an overview of FastPath. FastPath has three main steps: structural analysis based on the HFG [24], efficient bug hunting with IFT-enhanced simulation [23], and exhaustive formal verification based on UPEC-DIT [22]. We start with the RTL description of the DUV and a security specification that defines what data is considered confidential. This paper focuses on data-dependent side effects (as described in Sec. II), but the proposed methodology can also be extended to other threat models. We assume an attacker is able to measure the timing of a system by observing control-related outputs, e.g., by monitoring the signals of a bus.

Our goal is to exhaustively verify (with as little manual inspection as possible) that sensitive data inputs (X_D) do not affect the control behavior of the system (Y_C). In addition, by inspecting violations of data-obliviousness, FastPath systematically derives a set of software constraints under which the hardware is guaranteed to operate in a data-oblivious manner. The verified hardware, in combination with these derived restrictions for the software, forms a root of trust for the higher levels of the system stack.

A. Structural Analysis

The first step is to create an HFG of the design. Because this process considers only structural information, it is easily scalable to complex systems. The HFG serves two primary purposes. First, it is a starting point for the other two approaches, providing structural information that can be used to generate property templates or speed up proofs with optimizations such as cone-of-influence reduction. Second, we can leverage it to skip the proof procedure in trivial cases. There can be no information flow if there is no structural connection between sensitive data inputs X_D and publicly visible control outputs Y_C. This may be the case if the design is largely data-oriented and does not implement complex control behavior (cf. Sec. V). In particular, FastPath terminates early if

$$\forall n_x \in X_D, \forall n_y \in Y_C : q(n_x, n_y) = \emptyset$$

where $q(n_x, n_y)$ is an HFG path query (cf. Sec. III-A).

In many cases, however, structural connections exist, but the design still operates data-independently in the given application scenario. For example, data-dependent instructions are explicitly avoided in constant-time programming. Even though they are never used, the functionality of these instructions is still implemented, creating a structural path through which information could flow. Another example of such an application scenario is the limited functionality of an unprivileged user. Structural analysis alone is not sophisticated enough to account for such usage restrictions. Therefore, we need to be able to verify a system's data-obliviousness under a given set of software constraints.

B. IFT-Enhanced Simulation

In the next step, an IFT-enhanced simulation of the design is performed. This can be done using any existing testbench or random simulation approach. We seek to verify the following IFT property:

$$X_D \; =/=> \; Y_C$$

At the start of the simulation, all sensitive data inputs X_D are tainted (labels are set to HIGH), and the labels of the remaining signals are set to LOW. If any control output $y_c \in Y_C$ has its labels transition from LOW to HIGH, this constitutes a counterexample and indicates that it is influenced by the sensitive data.

In the case of a counterexample, the verification engineer must investigate the root cause. The counterexample may represent a scenario that contradicts the given application scenario. In this case, the setup must be adjusted, usually by introducing constraints on the simulation or restricting the flow policy, and the simulation is restarted. However, if an actual security problem is found, the leakage must be fixed in the RTL design, and the methodology starts over.

The main advantage of running an IFT-enhanced simulation is the ease of setup and the efficiency of debugging. Given a stimulus source and an interface partitioning (cf. Sec. II), no manual effort is required to run the simulation. In addition, each counterexample shows a complete propagation path from data input to control output, starting from reset. This avoids false alarms and makes root cause analysis easier than in a formal approach where the counterexamples may start from an arbitrary run-time state. However, if a signal does not become tainted during simulation, this does not mean that no information can flow into that signal. It only indicates that there is no information flow for the scenarios exercised by the testbench. Subtle corner-case scenarios can easily be missed, which often happens in the presence of unknown vulnerabilities. Therefore, FastPath uses formal verification to ensure that data cannot propagate further than what was found by simulation.

Definition 2 (*Untainted State Signals*)**.** For a given design with an input partitioning $X = X_D \cup X_C$ and a given testbench, we define

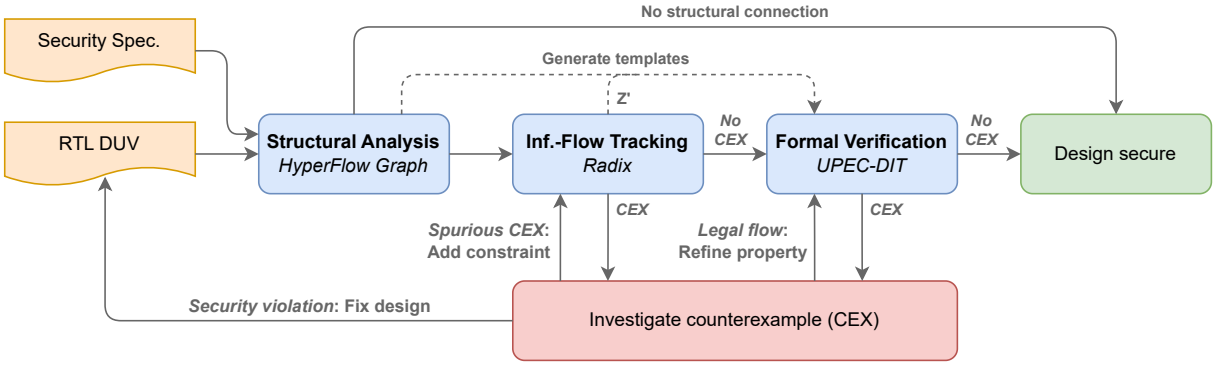


Fig. 1. The FastPath Verification Flow determines if the RTL design adheres to the security specification. FastPath uses a three-step process to efficiently compute counterexamples and update the specification with new constraints and property refinements. The output is a security violation that must be fixed in the design, or a guarantee that the design is secure.

the subset of all state-holding signals Z that have not been influenced by X_D during simulation as the set of untainted state signals $Z' \subseteq Z$.

Def. 2 describes the set of signals Z' whose labels remained LOW during simulation. The goal of the subsequent formal step is to verify that Z' constitutes a semantic partitioning of the computational problem. In particular, FastPath formally verifies by inductive reasoning that no sequence of inputs to the design exists for which a state variable $z' \in Z'$ becomes tainted. Hence, no taint can reach any output $y_c \in Y_C$, which means that X_D never influences the attacker-observable control outputs Y_C.

Finally, we emphasize that FastPath does not require sophisticated testbenches to achieve significant efficiency improvements. Because the formal step catches any remaining leaks, the verification engineer does not have to worry about missing corner cases during simulation. In addition, refinements to the IFT flow policy can be applied without worrying about over-constraining the policy.

C. Unique Program Execution Checking (UPEC)

In the final step, we apply an inductive UPEC-DIT [22] property to achieve exhaustive coverage. In the original UPEC-DIT approach, the verification engineer is required to manually inspect *every* propagation originating from the data inputs X_D. This process can be tedious, especially for large data-driven designs, as most propagation alerts are legal, but still require the verification engineer to manually remove the newly reached signal from consideration. In this work, however, we already possess knowledge about possible propagation paths. We use it to skip most of the manual steps of the UPEC-DIT procedure by jumping directly to the, potentially final, inductive step.

```
1 UPEC-DIT(Z', Y_C):
2 assume:
3   at t:              two_safety_eq(Z')
4   during [t, t+1]: software_constraints()
5 prove:
6   at t+1:            two_safety_eq(Z')
7   during [t, t+1]: two_safety_eq(Y_C)
```

Listing 1. Generalized UPEC-DIT interval property

At the end of the IFT-simulation, the set of all untainted state signals Z' is returned to the HFG analysis. Using the structural information, the HFG stage then generates both the 2-safety computational model (see Sec. III-C) and the verification framework. The key formal property is shown in Lst. 1. It employs the following macros:

- *two_safety_eq*(A) specifies that all signals of the given set A are equal between the two instances of the computational model. We use this macro in both the assumption and commitment parts of the property to verify that, when considering all previous propagation paths, no further information flow is possible.
- *software_constraints*() specifies the given usage constraints under which the system operates data-obliviously. These are inherited from the simulation, but may need to be extended if new violations are found using the formal property (see below). However, this did not happen in our experiments.

In case the property check fails, the verification engineer must examine the returned counterexample and identify the root cause. If the counterexample shows a security vulnerability, the design must be fixed and FastPath starts over. If the counterexample shows a scenario which violates data-obliviousness, but can be prevented with a software constraint, we update *software_constraints*(). After adding such a new software constraint, we backtrack to the simulation step (Sec. IV-B), since a new constraint may increase the set of untainted state signals Z'. If a spurious counterexample is found (i.e., an unreachable scenario caused by the symbolic initial state), we refine the property with an invariant (see [22]). Finally, if the counterexample shows a legal data propagation (that was missed by IFT-simulation), we refine the property by removing the corresponding signal from Z'. After refining the property, we simply repeat the formal property check.

If the property is verified successfully, it means that the resulting Z' is a partitioning of the problem and will never be affected by the data inputs X_D under the derived constraints. This results in a fixed point where no control output Y_C is ever influenced by confidential data X_D, and our system is thus data-oblivious according to Def. 1.

V. Experiments

We conducted experiments on a diverse set of designs, including various accelerators and processors. Table I gives an overview of our experiments. All experiments are available in our GitHub repository [31]. We used Cycuity's Radix-S tool [32] for IFT simulation and Siemens EDA's OneSpin 360 tool [33] for formal verification.

For each experiment, we show:

- The result of the analysis. In particular, we report whether the design operates data-obliviously either always (*True*), only

Design	Data-Oblivious	Method	State Size		Data Prop. Found		Manual Inspections		
			Signals	Bits	IFT	+ UPEC	Original [22]	FastPath	Reduction (%)
SHA512	True	HFG	37	2162	–	–	33	0	100.0
AES (opencores)	True	HFG	24	554	–	–	19	0	100.0
AES (secworks)	True	HFG	26	2470	–	–	11	0	100.0
CVA6-DIV	Constrained	UPEC	15	217	10	10	12	3	75.0
FWRISCV-MDS	Constrained	UPEC	13	331	5	8	9	4	55.5
ZipCPU-DIV	False	IFT	14	142	14	–	9	1	88.8
cv32e40s	Constrained	UPEC	471	5624	16	17	30	19	36.6
BOOM	Constrained	UPEC	5340	41948	163	177	185	24	87.0

TABLE I
CASE STUDIES

under certain, reasonable software constraints (*Constrained*), or not at all (*False*). FastPath systematically finds such constraints under which the hardware operates data-obliviously. In our experiments, we also performed a more fine-grained analysis that considers a propagation path for each combination $(x_D, y_C) \in X_D \times Y_C$ of a data input x_D and a control output y_C. For some of the designs, we present these results in more detail in the text below, highlighting interesting findings.

- The step in the FastPath methodology at which the experiment becomes complete, whether by simple structural proof using the HFG (Sec. IV-A), by finding a security issue using IFT simulation (Sec. IV-B), or by exhaustive formal verification using UPEC-DIT (Sec. IV-C).
- The number of state-holding word-level signals and the total number of bits consumed by these signals.
- The number of signals influenced by the data inputs. In particular, we present how many propagations were found by IFT alone and how many were found after employing formal verification, i.e., UPEC-DIT. If the method terminates earlier, e.g., due to a structural proof by the HFG, this step is not required.
- Finally, we measure the amount of manual signal inspection required. We count the number of divergent signals, i.e., signals with different values in the two instances of the 2-safety UPEC model, that are found in counterexamples during the execution of FastPath and require manual inspection. Counterexamples that cause the IFT flow policy to be updated or a constraint or an invariant [21] to be created are also counted. We compare this value to the manual effort of the formal-only approach presented in [22].

A. Cryptographic Accelerators

We analyzed three different crypto-accelerators, one implementing the Secure Hash Algorithm (SHA) [34] and the other two implementing the Advanced Encryption Standard (AES) [35], [36]. We investigated whether any plaintext or key input could affect the control behavior of the system.

Our proposed methodology was able to prove the data obliviousness of all designs through a simple structural analysis using the HFG. The HFG showed that there is no path, implicit or explicit, starting from any of the data inputs X_D to any of the control outputs Y_C. While the absence of such a path is not trivially guaranteed, it can be expected because the round-based nature of the underlying algorithms makes a data dependency very unlikely.

The ability to perform structural analysis trivializes these proofs by eliminating the need for simulation or formal verification. As a result, no manual effort is required beyond the initial annotation of the interface. In contrast, the original UPEC-DIT method [22], for the SHA accelerator, requires a manual inspection of 33 signals influenced by X_D before reaching a fixed point.

B. Division Modules

We verified three division modules taken from processor designs. A structural path between X_D and Y_C exists for all designs, which means that structural analysis alone is not sufficient to verify data-obliviousness.

For the division unit of the ZipCPU [37] project, IFT simulation detected several data dependencies, such as an early termination for a divisor of zero. Since there is no reasonable software constraint under which the design operates data-obliviously, there is no need to proceed to the formal analysis.

The *Featherweight RISC-V* [38] processor employs a single module for multi-cycle multiplication, division and shifting. Using a random pattern testbench, IFT simulation was able to confirm the findings of [22] that the timing of shift operations is dependent on the shift amount. We introduced a constraint to exclude shifting, rerun the simulation and generated formal properties based on the untainted signals Z'. In the formal step, UPEC was able to detect three additional data propagations that were missed due to the simplicity of the testbench. After refining the formal property, FastPath verified that the design can operate data-obliviously if the software satisfies the constraint derived by our method (no shifting).

We also verified a hardened version of the CVA6 division unit taken from [39]. It implements security labels that indicate whether operands are public or confidential. Division timing is dynamically optimized to depend only on public information, but to perform better than worst-case latency. We taint the operands according to their labels, i.e., whenever they are confidential. In this experiment, the IFT simulation reported a false counterexample due to an overly conservative flow policy. In FastPath, however, we do not have to worry about restricting the IFT policy, since the formal step catches any missing propagation paths. In addition, two invariants were required to verify the security mechanism of this design.

C. In-Order Processor

We applied FastPath to cv32e40s [25], a RISC-V processor intended for security applications. It implements a variety of security mechanisms, including a data-independent timing (*data_ind_timing*) mode that ensures that normally variable-time instructions execute with a fixed latency. Some special cases, such as misaligned memory accesses, are excluded. Our results confirm the expectation that CSR accesses, jumps and branches can still influence the control behavior of the system even when in *data_ind_timting* mode. Load and store instructions that access more than a single byte can cause misalignment and thus affect control. In contrast, as intended, the latency

of division instructions becomes constant when the *data_ind_timing* mode is active.

However, FastPath was able to identify another, previously unknown, leakage affecting the operands of *all* instruction types. The method detected that the operands residing in the pipeline buffer of the ID-EX stage are always visible on the primary interface to the data memory, regardless of whether a memory access is taking place. Even if the corresponding validity signal is deasserted, the core still interprets the operands as address and data and passes them on to the outputs. Any IP, faulty or malicious, that does not comply with the protocol, can easily obtain information about internal computations, rendering security mechanisms such as *data_ind_timing* obsolete. We reported the vulnerability and worked with the processor development team to contribute a fix.

We used a fairly rudimentary testbench in our experiment, yet the IFT simulation was able to find almost all data propagations. The only missed state signal was inside the multiplier and is only tainted during MULH instructions. The formal step of FastPath showed a counterexample for this signal. In this experiment, most manual inspections involved deriving constraints for non-oblivious instructions, or writing invariants.

D. Out-of-Order Processor

Finally, to demonstrate its efficiency and scalability, we apply FastPath to the Berkeley Out-of-Order Machine (BOOM) [40] and compare the manual effort with the results of [22]. We confirm their findings regarding the data-obliviousness of certain instructions.

BOOM implements a sophisticated floating point (FP) pipeline, resulting in a large number of signals in the data path. Using a simple testbench, FastPath identifies the vast majority of these signals, as well as non-oblivious instructions, via IFT simulation. This reduces the manual effort of the formal step to corner cases, such as special cases for FP computations, and greatly reduces the overall manual effort for verifying the most complex design in our benchmark suite.

E. Discussion - Runtime and Manual Effort

IFT-simulation and formal property checking produce counterexamples that provide violations of the specified and desired security behavior as described in the security properties. These counterexamples require a manual analysis, during which the verification engineer must determine whether the counterexample describes a scenario that violates the intended design behavior. In other words, the verification engineer must determine if the security properties correctly describe the threat model. Property specification is challenging, and analyzing the counterexamples often provides cases where the properties must be refined. This analysis may take a considerable amount of time, depending on the verification engineer's design knowledge, the design's complexity, and the complexity of the counterexample at hand. For this reason, we use the number of signals that must be inspected manually as a metric for the verification effort. The manual inspection time is hard to quantify but can take weeks to months [21] for the designs considered in this work. As shown in Table I, compared to previous work, FastPath is able to significantly reduce the time spent manually examining counterexamples and thus the overall verification effort.

We observe a reduction of manual effort between 36% and 100%. The smallest reduction is obtained for the cv32e40s processor. This can be explained by the fact that this processor contains a relatively small data path but a complex control structure interweaved with several security features. This results in a comparably large number of counterexamples that lead to invariants or restrictions needed for security-aware software development. Remarkably, for BOOM, by far the largest and most labor-intensive design examined in this work, an 86% reduction of manual effort was achieved. In contrast to cv32e40s, BOOM employs a much larger and more sophisticated data path. In spite of its complex control for instruction scheduling, our hybrid approach is particularly effective because it clearly separates control from data path. Overall, our experiments demonstrate the immense promise of FastPath for increasing productivity in complex design projects.

Tool runtime is an important aspect when assessing the scalability of a verification approach. For FastPath, however, these runtimes were negligible. Running a single IFT-simulation, including the extraction of all tainted signals from the simulation trace, took 1–2 minutes. In the formal step, the initial design elaboration in the formal model checker took around 5 minutes for the most complex experiment. Afterward, a single check of the formal property (Lst. 1) finished in less than 10 seconds by merit of the symbolic initial state. Hence, tool runtime contributes only a small fraction to FastPath's verification time.

VI. Related Work

FastPath combines simulation and formal verification in the context of hardware security. Previous work exploring synergies between the two techniques, such as [41], has mostly attempted to improve simulation efficiency and coverage with formal methods. Consequently, such approaches inherit the limitation of simulation techniques and cannot provide formal guarantees. But there are also efforts in the other direction, i.e., the use of random simulation to improve formal approaches, such as in [42]. However, none of these methods address hardware security issues.

Significant efforts [43], [26], [22], [44], [27], [28], [45] have been made to detect leakage of confidential operands through timing or other microarchitectural side effects. Clepsydra [43] instruments an RTL design with IFT logic to discover timing flows through either simulation or formal verification. UPEC-DIT [22] relies on standard SystemVerilog Assertions (SVA) properties to detect data-dependent side-effects in the microarchitecture. While the symbolic initial state of the approach allows for a very high scalability, its main drawback lies in the considerable manual refinement effort (cf. Sec. V). More recent formal methods take a different approach [27], [28], [45], but have not been shown to scale to complex systems. We believe that also these methods could benefit from a hybrid approach along the lines proposed in this paper.

VII. Conclusion

FastPath is a hybrid verification methodology that leverages the complementary strengths of structural analysis, IFT-simulation, and formal verification to produce exhaustive security guarantees for data-oblivious architectures. FastPath uses a global formulation that automatically transfers intermediate results between the three verification techniques. The key benefit of FastPath is a drastic reduction of manual effort, achieved by combining the efficiency of simulation with the exhaustiveness of formal methods. In our case studies on functional units, FastPath was able to verify (or disprove) data-obliviousness with little to no manual interaction. Furthermore, especially for complex systems with intensive data paths like BOOM, FastPath offers unique scalability both in terms of runtime and manual effort. Thus, FastPath provides a tremendous reduction in security verification effort since the manual inspection dominates the overall verification time. Our future work aims at exploring the synergies of the proposed method for different threat models.

References

[1] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: a timing attack on OpenSSL constant-time RSA," *Journal of Cryptographic Engineering*, vol. 7, pp. 99–112, 2017.

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[4] J. R. S. Vicarte, M. Flanders, R. Paccagnella, G. Garrett-Grossman, A. Morrison, C. W. Fletcher, and D. Kohlbrenner, "Augury: Using data memory-dependent prefetchers to leak data at rest," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1491–1505.

[5] B. Chen, Y. Wang, P. Shome, C. W. Fletcher, D. Kohlbrenner, R. Paccagnella, and D. Genkin, "GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers," in *USENIX Security Symposium*, 2024, pp. 1–21.

[6] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. W. Fletcher, "Speculative taint tracking (STT): A comprehensive protection for speculatively accessed data," in *52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 954–968.

[7] J. Yu, L. Hsiung, M. El Hajj, and C. W. Fletcher, "Data oblivious ISA extensions for side channel-resistant and high performance computing." in *26th Network and Distributed System Security Symposium*, 2019.

[8] S. Cauligi, C. Disselkoen, K. v. Gleissenthall, D. Tullsen, D. Stefan, T. Rezk, and G. Barthe, "Constant-time foundations for the new spectre era," in *41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, USA, 2020, p. 913–926.

[9] M. Schwarz, M. Lipp, C. Canella, R. Schilling, F. Kargl, and D. Gruß, "ConTExT: A generic approach for mitigating spectre," in *Network and Distributed System Security Symposium 2020*, Feb. 2020.

[10] K. Loughlin, I. Neal, J. Ma, E. Tsai, O. Weisse, S. Narayanasamy, and B. Kasikci, "DOLMA: Securing speculation with the principle of transient Non-Observability," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 1397–1414.

[11] T. Jauch, A. Wezel, M. R. Fadiheh, P. Schmitz, S. Ray, J. M. Fung, C. W. Fletcher, D. Stoffel, and W. Kunz, "Secure-by-construction design methodology for CPUs: Implementing secure speculation on the RTL," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[12] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 45–60.

[13] "A beginner's guide to constant-time cryptography," https://www.chosenplaintext.ca/articles/beginners-guide-constant-time-cryptography.html, accessed: 2024-09-27.

[14] M. Andrysco, A. Nötzli, F. Brown, R. Jhala, and D. Stefan, "Towards verified, constant-time floating point operations," in *2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, New York, NY, USA, 2018, p. 1369–1382.

[15] "Data operand independent timing ISA guidance," https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/data-operand-independent-timing-isa-guidance.html, acc.: 2024-08-14.

[16] "Arm Armv8-A architecture registers," https://developer.arm.com/documentation/ddi0595/2021-06/AArch64-Registers/DIT--Data-Independent-Timing, accessed: 2024-08-14.

[17] "RISC-V cryptography extension," https://github.com/riscv/riscv-crypto, accessed: 2024-09-25.

[18] J. R. S. Vicarte, P. Shome, N. Nayak, C. Trippel, A. Morrison, D. Kohlbrenner, and C. W. Fletcher, "Opening Pandora's box: A systematic study of new ways microarchitecture can leak private data," in *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[19] J. A. Goguen and J. Meseguer, "Security policies and security models," in *IEEE Symposium on Security and Privacy (SP)*, 1982, pp. 11–20.

[20] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[21] M. R. Fadiheh, A. Wezel, J. Müller, J. Bormann, S. Ray, J. M. Fung, S. Mitra, D. Stoffel, and W. Kunz, "An exhaustive approach to detecting transient execution side channels in RTL designs of processors," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 222–235, 2023.

[22] L. Deutschmann, J. Müller, M. R. Fadiheh, D. Stoffel, and W. Kunz, "A scalable formal verification methodology for data-oblivious hardware," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 9, pp. 2551–2564, 2024.

[23] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware information flow tracking," *ACM Comp. Surveys (CSUR)*, vol. 54, no. 4, pp. 1–39, 2021.

[24] A. Meza and R. Kastner, "Information flow coverage metrics for hardware security verification," 2023. [Online]. Available: https://arxiv.org/abs/2304.08263

[25] "OpenHW Group CORE-V CV32E40S RISC-V IP," https://github.com/openhwgroup/cv32e40s, updated: 2024-10-31, accessed: 2024-11-11.

[26] K. v. Gleissenthall, R. G. Kıcı, D. Stefan, and R. Jhala, "Solver-aided constant-time hardware verification," in *2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, New York, NY, USA, 2021, p. 429–444.

[27] S. Dinesh, M. Parthasarathy, and C. W. Fletcher, "ConjunCT: Learning inductive invariants to prove unbounded instruction safety against microarchitectural timing attacks," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3735–3753.

[28] K. Ceesay-Seitz, F. Solt, and K. Razavi, "μCFI: Formal verification of microarchitectural control-flow integrity," in *2024 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24, New York, NY, USA, 2024.

[29] K. Ryan, M. Gregoire, and C. Sturton, "Seif: Augmented symbolic execution for information flow in hardware designs," in *12th Int. Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '23. New York, NY, USA: ACM, 2023, p. 1–9.

[30] J. Urdahl, D. Stoffel, and W. Kunz, "Path predicate abstraction for sound system-level models of RT-level circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 2, pp. 291–304, 2014.

[31] "FastPath repository," https://github.com/anmeza/dac_2025_fastpath.

[32] "Cycuity Radix," https://cycuity.com/.

[33] "Siemens EDA OneSpin 360," https://eda.sw.siemens.com/.

[34] "SHA cores," https://opencores.org/projects/sha_core, updated: 2018-03-06, accessed: 2024-09-26.

[35] "AES IP core," https://opencores.org/projects/aes_core, updated: 2018-06-10, accessed: 2024-09-26.

[36] "secworks aes," https://github.com/secworks/aes, updated: 2023-02-08, accessed: 2024-09-26.

[37] "The Zip CPU," https://github.com/ZipCPU/zipcpu, updated: 2024-08-26, accessed: 2024-10-25.

[38] "FWRISC," https://github.com/Featherweight-IP/fwrisc, updated: 2022-01-17, accessed: 2024-10-25.

[39] L. Deutschmann, Y. Kazhalawi, J. Seckinger, A. L. Duque Antón, J. Müller, M. R. Fadiheh, D. Stoffel, and W. Kunz, "Data-oblivious and performant: On designing security-conscious hardware," in *2024 IEEE 25th Latin American Test Symposium (LATS)*, 2024, pp. 1–6.

[40] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, May 2020.

[41] P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long, "Smart simulation using collaborative formal and simulation engines," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2000, pp. 120–126.

[42] F. Krohm, A. Kuehlmann, and A. Mets, "The use of random simulation in formal verification," in *International Conference on Computer Design*. IEEE, 1996, pp. 371–376.

[43] A. Ardeshiricham, W. Hu, and R. Kastner, "Clepsydra: Modeling timing flows in hardware designs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 147–154.

[44] Z. Wang, G. Mohr, K. von Gleissenthall, J. Reineke, and M. Guarnieri, "Specification and verification of side-channel security for open-source processors via leakage contracts," in *2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23, New York, NY, USA, 2023, p. 2128–2142.

[45] Y. Hsiao, N. Nikoleris, A. Khyzha, D. P. Mulligan, G. Petri, C. W. Fletcher, and C. Trippel, "RTL2MμPATH: Multi-μPATH synthesis with applications to hardware security verification," in *57th IEEE/ACM International Symposium on Microarchitecture (MICRO '24)*, 2024.

PracMHBench: Re-evaluating Model-Heterogeneous Federated Learning Based on Practical Edge Device Constraints

Yuanchun Guo, Bingyan Liu*, Yulong Sha, Zhensheng Xian
{gyc2001, bingyanliu, shayulong, xsin}@bupt.edu.cn
School of Computer Science , Beijing University of Posts and Telecommunications, Beijing, China

***Abstract*—Federating heterogeneous models on edge devices with diverse resource constraints has been a notable trend in recent years. Compared to traditional federated learning (FL) that assumes an identical model architecture to cooperate, model-heterogeneous FL is more practical and flexible since the model can be customized to satisfy the deployment requirement. Unfortunately, no prior work ever dives into the existing model-heterogeneous FL algorithms under the practical edge device constraints and provides quantitative analysis on various data scenarios and metrics, which motivates us to rethink and re-evaluate this paradigm. In our work, we construct the first system platform PracMHBench to evaluate model-heterogeneous FL on practical constraints of edge devices, where diverse model heterogeneity algorithms are classified and tested on multiple data tasks and metrics. Based on the platform, we perform extensive experiments on these algorithms under the different edge constraints to observe their applicability and the corresponding heterogeneity pattern.**

***Index Terms*—Federated Learning, Platform, Model Heterogeneity**

I. Introduction

Federated learning (FL) is empowering multiple real-world applications such as Google's Keyboard [1] and medical image analysis [2], whose key idea is to employ a set of personal clients to collaboratively train a deep learning model with the orchestration of a central server. In recent years, conducting FL among edge devices has become a noteworthy trend [3], [4]. Under this condition, the typical FL assumption with respect to keeping an identical model architecture is hard to hold since edge devices are usually resource-constrained.

The research community has recognized this issue and introduced the concept of *Model-Heterogeneous Federated Learning (MHFL)* [5]. In this context, each client maintains a model that accounts for the device's resource constraints, and the federated learning process operates among these heterogeneous models. MHFL can offer the following benefits: (1) For resource-constrained devices, MHFL provides the flexibility to adaptively conduct FL under diverse resource constraints (e.g., computation, communication, and memory) by adjusting the model size. (2) For individual edge users with distinctive preferences, MHFL allows them to customize the model's structure and size. Overall, MHFL is well-suited for deployment on various edge devices.

In the past few years, a large number of papers have been published to improve the performance of MHFL [6], [7]. After exploration, we find that all of them are based on simple configurations (i.e., using the proportion of the original model to satisfy device constraints). However, they do not consider real-world device constraints. As shown in Table I, on two real edge devices (Jetson Nano, Jetson Orin NX), models generated by different heterogeneous methods with the same proportion exhibit significant differences in terms of various resources, particularly for the computational cost (training time) and the memory usage. Therefore, *we argue that the current comparisons of model-heterogeneous federated learning methods are not fair, and the community should rethink and re-evaluate model-heterogeneous federated learning under practical device constraints*.

To gain a fair and comprehensive performance comparison of diverse model-heterogeneous mechanisms for real-world constraints of edge devices, we construct the first system platform **PracMHBench**. PracMHBench defines three representative heterogeneity levels, each of which includes several corresponding MHFL algorithms. The platform also contains multiple model architectures and metrics, which are evaluated on various data tasks, spanning from computer vision (CV), natural language processing (NLP), and human activity recognition (HAR). Based on the benchmark, PracMHBench further build three typical edge-side constraint cases, which we call *computation-limited MHFL, communication-limited MHFL, and memory-limited MHFL*, and conducts experiments on them. For each limitation setting, we create it based on the collected statistics of practical edge devices, such that the results can benefit and guide the MHFL deployment in real-world resource-constrained scenarios.

Based on PracMHBench, we first perform extensive experiments on different model-heterogeneous algorithms under the specific device constraint to demystify the performance on different data tasks. Next, in terms of our evaluated metrics, we analyze the pattern of different heterogeneity algorithms and levels, aiming to explore the most suitable strategy for different resource-constrained scenarios. Finally, through the experiments, we summarize several insightful observations and conclusions (details in Section V).

This paper makes the following contributions:

- We construct a system platform PracMHBench to characterize and evaluate diverse model-heterogeneous algorithms under the practical resource-constrained edge devices. To the best of our knowledge, this is the first work that delves into the deployment of MHFL on real-world scenarios [1].
- We perform extensive experiments on PracMHBench with different heterogeneity levels and device constraints, and the results reveal a comprehensive landscape of current MHFL algorithms from a real system perspective.
- Based on the experiments, we summarize the insightful observations and practical implications that can benefit the practitioners and researchers targeting resource-constrained MHFL systems.

II. Background and Related Work

Model Heterogeneity in Federated Learning. In recent years, Federated learning (FL) tailored for massive edge devices has garnered unprecedented attention [3], [4], [8]–[10]. Unlike traditional federated learning that only focuses on algorithm performance, federated learning in such scenarios pays more attention to the characteristics of heterogeneous devices. For example, training VGG16 requires over 8GB memory with a batch size of 8 [11], making it hard to

*corresponding author.

This work was partly supported by the National Natural Science Foundation of China (62302054) and CCF-Huawei Populus Euphratica Fund (System Software Track, CCF-HuaweiSY202410).

[1] https://github.com/9yc/PracMHBench

TABLE I

MODELS GENERATED BY DIFFERENT HETEROGENEOUS METHODS WITH THE SAME PROPORTION (I.E., X0.5 OF THE ORIGINAL MODEL) ON JETSON ORIN NX AND JESTON NANO. THE STATISTICS INCLUDE THE NUMBER OF PARAMETERS, TRAINING TIME (ONE ROUND) AND MEMORY USAGE. 'N' REPRESENTS THE JETSON NANO. 'O' REPRESENTS THE JETSON ORIN NX.

Method	Model	Parameters(M)	Training Time(s)	Memory Usage(M)
SHeteroFL	ResNet101 (x0.5)	10.66	N:430.24 O:212.72	593
DepthFL	ResNet101 (x0.5)	10.29	N:515.93 O:254.65	1220
FedRolex	ResNet101 (*0.5)	10.75	N:465.17 O:233.56	780
FeDepth	ResNet101 (*0.5)	10.54	N:450.64 O:222.35	631

deploy for many edge devices. The research community has proposed *model-heterogeneous federated learning (MHFL)* to address the issue. The primary aim is to enable each device to customize its model size while ensuring federation with other devices.

Achieving MHFL encompasses two categories of state-of-the-art techniques. The first category is sub-model partial aggregation [5], [6], [12]. Its fundamental idea involves extracting sub-models from a large model that adheres to the memory constraints of devices, followed by weight aggregation based on the positional information of neurons. The second category employs knowledge distillation to achieve federated aggregation [7], [13]–[15]. Its primary pipeline involves: (1) The server leverages a public dataset to compute the output logits of each model and performs logits aggregation; (2) The server sends the aggregated logits back to each client, thereby enabling them as the teacher model for the distillation process. While MHFL has been extensively studied, its level of heterogeneity and the performance of various methods under different real device constraints have not been thoroughly explored. Our work aims to fill this gap by constructing a comprehensive and practical platform for MHFL in resource-constrained edge systems.

FL Platforms. Benchmarking and evaluating the performance of FL is essential for practitioners to understand and employ associated algorithms and configurations effectively. At present, there are two main categories of platforms for FL. The first category comprises general FL platforms, which primarily aim to comprehensively assess the performance of various FL algorithms under different conditions [16]–[18]. The second category consists of specific FL platforms, tailored for in-depth exploration of certain characteristics within FL systems [19]–[21]. For example, to assess the data heterogeneity, Li *et al.* [20] proposed a comprehensive data partitioning strategy to cover typical non-IID (non-independently and identically distributed) data cases and evaluated existing FL algorithms under such settings. Furthermore, Yang *et al.* [21] developed an FL platform to explore the impact of heterogeneity on performance, particularly focusing on hardware and state heterogeneity. In our study, we pay more attention to the impact of *model heterogeneity* on FL performance and the practice of conducting diverse heterogeneous methods on resource-constrained edge devices, which is fundamentally different from previous research efforts.

III. BENCHMARK CONSTRUCTION

PracMHBench is a platform aimed to characterize and evaluate diverse model heterogeneity algorithms on practical resource-constrained devices. In this section, we first describe its benchmark construction in detail. The concrete statistics are shown in Table II.

Model Heterogeneity Levels. Due to diverse resource constraint conditions, various clients in federated learning may equip models with different capacities or architectures. In our PracMHBench, we define and categorize the model heterogeneity mechanism in an FL system into three levels (as shown in Figure 2):

- *Width Heterogeneity*, where the models employed by clients share the same topology while exhibiting differences in the number of channels in each layer, resulting in heterogeneous models with various widths.
- *Depth Heterogeneity*, where the model topology employed by clients is also identical and the difference lies in the number of layers, which can be considered as the depth-level heterogeneity.
- *Topology Heterogeneity*, where clients employ models with entirely different topologies. For example, an FL system may consist of diverse model architectures, such as ResNet [22], EfficientNet [23], MobileNet [24], and GoogleLeNet [25], in order to satisfy various user preferences and requirements.

Representative Heterogeneous Algorithms. In terms of the aforementioned heterogeneity levels, we select several heterogeneous algorithms for each level to represent its performance. Specifically, for *width heterogeneity*, we adopt Fjord [12], SHeteroFL [5] and FedRolex [6], whose key settings are allocating sub-models with different widths to clients before conducting specifically designed federation schemes. In the case of *depth heterogeneity*, we select FeDepth [26], InclusiveFL [27] and DepthFL [28], which achieves depth scaling by pruning one or more of the top layers of the global model. Towards *topology heterogeneity*, FedProto [7] and Fed-ET [14] are used to conduct FL with the help of knowledge distillation among these heterogeneous architectures.

Data Tasks. In our platform, we target three areas: computer vision (CV), natural language processing (NLP), and human activity recognition (HAR), each of which is evaluated by two representative datasets. For CV, we utilize CIFAR-10 and CIFAR100 [29]. For NLP, we employ AG-News [30] and Stack-OverFlow [31], and for HAR, we choose HAR-BOX [32] and UCI-HAR [33].

Model Architectures. To evaluate the different levels of model heterogeneity, we select various model architectures for experiments. For width and depth heterogeneity, we follow the settings outlined in related works [5], [6], conducting experiments using four different widths/depths (100%, 75%, 50%, 25%) of the ResNet-101 [22] on the CIFAR-100 dataset [29] and MobileNetV2 [24] on CIFAR-10 dataset [29]. Additionally, for NLP tasks, we employ a customized transformer model [34] on AG-news and ALBERT [35] on Stack Overflow. We partition them into various dimensions to meet different definitions of heterogeneity. For HAR tasks, we employ customized CNNs following the previous work [36] and partition it into various dimensions to meet different definitions of heterogeneity. Regarding topology heterogeneity, we conduct experiments using two model families with distinctive architectures on CV tasks, including the ResNet Family [22] (ResNet-18, ResNet-34, ResNet-50, ResNet-101) on CIFAR-100 [29], and the MobileNet Family [24] (MobileNetV2, MobileNetV3 small, MobileNetV3 large) on CIFAR-10 [29]. Considering NLP models, we choose the ALBERT Family [35] (ALBERT base, ALBERT large, ALBERT xxlarge) on Stack Overflow [31]. Additionally, we modified the structure of the customized CNNs for HAR tasks.

Evaluated Metrics. To provide a comprehensive evaluation, PracMHBench adopts the following four metrics: (i) *Global accuracy*, where we maintain a global dataset and the final federated model is evaluated on it. (ii) *Time-to-accuracy*, which is defined as the wall

TABLE II
STATISTICS OF THE PROPOSED PLATFORM. HERE 'HETERO' REFERS TO THE HETEROGENEITY LEVEL.

Hetero	Algorithm	CV		NLP		HAR	
		Model	Dataset	Model	Dataset	Model	Dataset
Width	Fjord	ResNet-101 MobileNetV2 Variants (100%,75%, 50%,25%)	CIFAR10, CIFAR100	ALBERT, Customized Transformer Variants (100%,75%, 50%,25%)	Stack-Overflow, AG-news	Customized CNN	HAR-BOX, UCI-HAR
	SHeteroFL						
	FedRolex						
Depth	FeDepth						
	InclusiveFL						
	DepthFL						
Topology	FedProto	MobileNet, Resnet Family		Albert Family			
	Fed-ET						

TABLE III
EDGE DEVICES USED IN OUR PLATFORM CONSTRUCTION.

Devices	Processors	GPU Memory
Jetson orin nx	1024-core NVIDIA Ampere GPU	16GB
Jetson tx2 nx	256-core NVIDIA Pascal GPU	4GB
Raspberry 4B	Broadcom BCM2711B0 quad-core A72 64-bit @ 1.5GHz CPU	no gpu

clock time for training a deep learning model to reach a pre-set accuracy, aiming at measuring the training speed. (iii) *Stability*, where we record the accuracy of each device and compute the variance among them to show the stability for various heterogeneous models. (iv) *Effectiveness*, where we compare the final accuracy of MHFL algorithms with a simple resource-aware homogeneous baseline (i.e., training the smallest homogeneous model across all heterogeneous devices). The effectiveness is measured by the accuracy improvement. Our goal is to test whether heterogeneous models are actually needed (i.e., whether the improvement is greater than zero) and the benefit of conducting MHFL (i.e., the improvement value).

IV. PRACTICAL DEVICE CONSTRAINTS

Given the above benchmark settings, we then manually build three typical resource-constrained cases based on real-world device properties (enumerated in Table III). As illustrated in Figure 3, we first build a model pool, where various deep learning models in the literature are measured and we pick out the suitable one in terms of the heterogeneity level and resource constraint. Our principle for selecting models from the model pool is to keep the constraint consistent for all methods to ensure fairness. In the following subsections, we describe each resource limitation on MHFL in detail. Note that for each specific limitation, other resources remain identical.

A. Case I: Computation-Limited MHFL

Description. Limited computing power means that different devices may demonstrate varying levels of computational constraints. As shown in Table I, there exists a significant difference in computing power among real edge devices, leading to distinctive training speed. Under this condition, we define *Computation-Limited MHFL* as follows:

Definition IV.1. (Computation-Limited MHFL). *Given the various computing capabilities of practical edge devices, the goal of computation-limited MHFL is to adapt the model size of its original version to ensure a same model training speed among devices for synchronous aggregation.*

Setting. To achieve such a scenario, we follow the IMA dataset settings [21], where they collected status from more than 1,000 devices (e.g., Samsung Note 10, Nexus 6, and Redmi Note 8) and recorded the real computational capabilities of these devices. We used the computing power of the IMA dataset to build the computation-limited scenarios. Specifically, we assign the diverse computational capabilities in the IMA dataset to the clients of the federated learning system as their training constraint, and select corresponding suitable models of different heterogeneity levels for each client from the model pool, with the help of the statistics of the training time for various models, which is provided by an AI Benchmark on various devices [37]. This helps ensure roughly equivalent training times for each device and achieve synchronous aggregation under a fixed submission deadline.

B. Case II: Communication-Limited MHFL

Description. The limited communication bandwidth indicates that the speed of uploading/downloading model parameters is affected by the actual device hardware and network environment. In an FL system, a client with small communication bandwidth may spend a lot of time for uploading or downloading model parameters, resulting in its inability to upload model parameters in time for aggregation. The *Communication-Limited MHFL* can be defined as follows:

Definition IV.2. (Communication-Limited MHFL). *Given the various communication bandwidth of different devices, the goal of communication-limited MHFL is to adjust the model size of its original version to ensure a same uploading/downloading speed among devices for synchronous aggregation.*

Setting. In the case of communication constraints, we control the communication time of each round in the federated learning system to a certain time (e.g., 200s) based on the real network bandwidth information provided by the IMA dataset. We then select appropriate models and corresponding quantities from the model pool to deploy different heterogeneity levels. Note that the parameter sizes corresponding to the various models under the different methods vary, and thus the model proportion of different sizes in the different methods also varies.

C. Case III: Memory-Limited MHFL

Description. Unlike the above, limited memory capacity focuses more on the actual memory requirements of training each model on different devices. Limited memory may have a direct impact on whether a client can conduct training and changing the model size can directly satisfy a specific memory limit. In this case, we define *Memory-Limited MHFL* as follows:

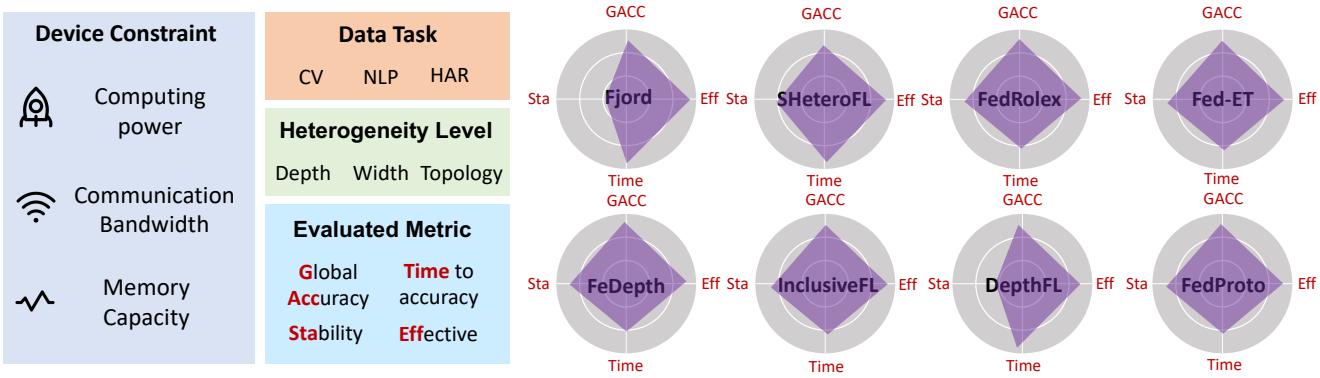


Fig. 1. Evaluation track of our platform PracMHBench. Here the information in radar charts is just for demonstration. The concrete performance will be compared in the subsequent experiments.

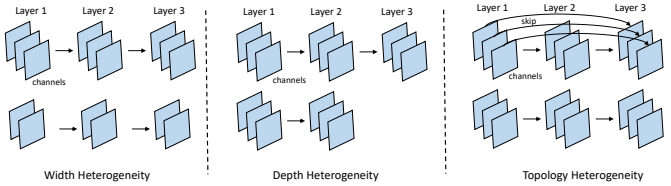


Fig. 2. Demonstration of different heterogeneity levels. Here 'skip' denotes the skip connection used in ResNets.

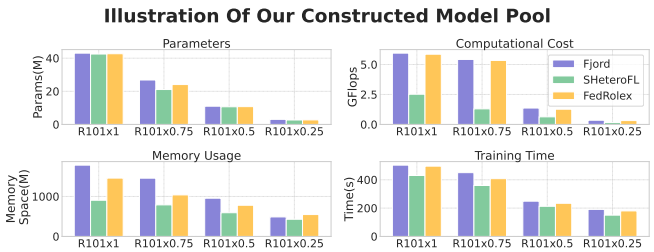


Fig. 3. Illustration of our constructed model pool. We use the tested system statistics (e.g., Parameters, GFlops) for three algorithms on Jetson Orin NX as an example. 'R' refers to ResNet.

Definition IV.3. (Memory-Limited MHFL). *Given the various memory capacity of different devices, the goal of memory-limited MHFL is to change the model size of its original version to ensure available local training for completing FL.*

Setting. We choose three typical GPU memory sizes of real devices: 16GB (e.g., Jetson Orin nx), 4GB (e.g., Jetson tx2 nx), and no GPU (e.g., Raspberry Pi). We then conduct experiments on real devices to observe whether the models in the model pool can be trained for federated learning. The largest trainable model is assigned to the client for meeting the corresponding memory size. In addition, the client proportion of different memory sizes is determined by the real-world proportion distribution depicted in [38].

V. Evaluation and Analysis

Based on PracMHBench, we perform extensive experiments and analyze the results. The theme of this measurement is to quantitatively understand the performance discrepancy of different heterogeneity algorithms. As shown in Figure 1, the evaluation track includes: (1) selecting a targeted resource constraint; (2) conducting experiments on a variety of data tasks for different heterogeneity levels and algorithms; (3) recording and comparing the performance based on the evaluated metrics. In the following parts, we will describe the results of each resource constraint in detail. Note that some methods are not compatible with NLP tasks, and thus we omit the corresponding results. In addition, for the memory-limited scenario, we only focus on large CNN models (i.e., Resnet-101) and Transformer models (i.e., ALBERT) since most of the current edge devices can hold small models for HAR tasks.

Specifically, for CIFAR10, CIFAR100 and AG-News, we adopt IID (Independent and Identically Distributed) partition. For Stack Overflow, HAR-BOX and UCI-HAR, the dataset is partitioned over user IDs, making the dataset naturally non-IID distributed. The number of clients for CIFAR10, CIFAR100, AG-News, Stack-OverFlow, HAR-BOX, and UCI-HAR is 100, 100, 50, 500, 100, 30. The sampling ratio is 10%. All of the experiments are conducted for 1000 federated rounds to ensure convergence. Finally, we run each experiment 3 times and average them as the reported results.

A. Results on Computation-Limited MHFL

Global accuracy and time-to-accuracy. The top row of Figure 4 shows the comprehensive results of global accuracy and time-to-accuracy. From the figure, we can observe that: (1) Overall, SHeteroFL and DepthFL achieve better performance on both global accuracy and time-to-accuracy. Regardless of the type of the data tasks, these two methods ensure relative superiority while Fjord and FedProto consistently exhibit the poorest performance across all data tasks. This suggests that under the computational constraint, the method's performance is unrelated to the type of data. (2) Although the two top-performing methods belong to different heterogeneity levels, we find that all depth-level methods (i.e., FeDepth, InclusiveFL, DepthFL) demonstrate favorable outcomes. This implies that adjusting the model size based on depth is a suitable choice in this scenario.

Stability and effectiveness. The bottom row of Figure 4 demonstrates the performance of stability and effectiveness. The following observations can be drawn: (1) Different from global accuracy and time-to-accuracy, which can identify the best algorithm for all data tasks, achieving a simultaneous satisfaction of stability and effectiveness is challenging on most datasets except for CIFAR-10, where FeDepth can be considered the best performer, which means that we can enhance the performance across nearly all devices. For other datasets, these two metrics tend to exhibit a trade-off, meaning that as stability increases, effectiveness tends to decrease. (2) For different heterogeneous levels, their performance on these two metrics does not follow any specific pattern and varies according to the differences in datasets.

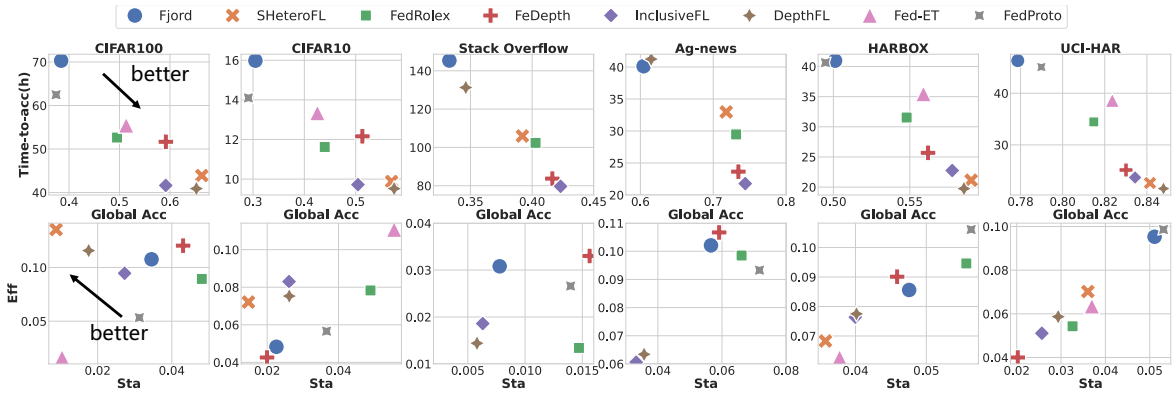


Fig. 4. Results on computation-limited MHFL. The top row refers to the comprehensive performance of global accuracy and time-to-accuracy. The bottom row represents the performance of stability and effectiveness.

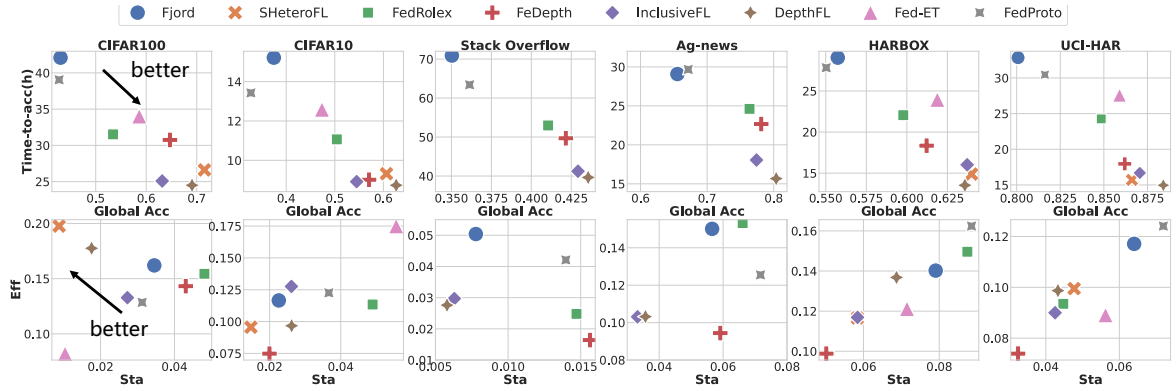


Fig. 5. Results on communication-limited MHFL.

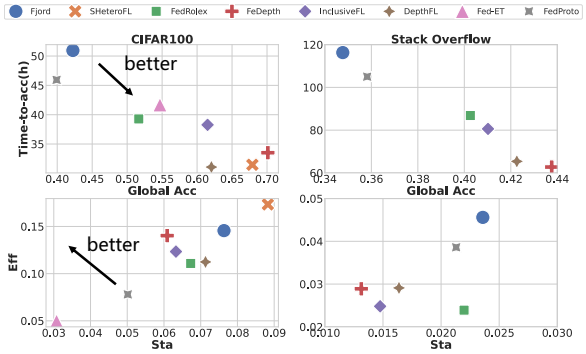


Fig. 6. Results on memory-limited MHFL.

> **Observation**: *In computation-limited MHFL, the superiority in terms of global accuracy and training speed remains a consistent pattern regardless of changes in data tasks. Among them, the methods belonging to the depth-level heterogeneity are relatively better than others. However, there is no one-size-fits-all algorithm that can achieve a desirable trade-off between the stability and effectiveness on diverse data tasks.*

B. Results on Communication-Limited MHFL

Global accuracy and time-to-accuracy. The top row of Figure 5 shows the comprehensive comparison. We can draw the following conclusions: (1) Similar to the scenario of computational constraints, under communication bandwidth constraints, SHeteroFL and DepthFL still perform optimally, and this conclusion remains consistent in all data tasks. Besides, depth-level methods outperform other baselines. This indicates that the extent of changes in model size due to computational and communication constraints is not significant. Thus, the overall superiority of various model-heterogeneous algorithms remains unchanged. (2) Although the performance ranking has not changed, there is still a noticeable variation in the training speed gap among different methods. For example, on the Stack Overflow dataset, under the communication bandwidth constraint, the gap of time-to-accuracy (i.e., training speed) is significantly smaller than it is in the computational constraint (roughly 2X). On other datasets, the conclusion also holds. We believe this is because the communication constraint is more related to the size of the model parameters, which better aligns with the original design principle of the algorithms (i.e., controlling the proportion of model size). Therefore, each method can essentially follow the original settings, and the fluctuation in training speed will not be too significant compared to the computational constraint, which not only considers the model parameters but also takes into account the computational capabilities of different devices.

Stability and effectiveness. As depicted in the bottom row of Figure 5, we record the stability and effectiveness performance under the communication limitation. We can see that: (1) At a high level, it is hard to figure out which algorithm can simultaneously ensure stability and effectiveness across all data tasks. In other words, these two metrics do not follow any specific pattern. (2) We discovered that heterogeneous methods at the depth level perform poorly in terms of effectiveness. The reason may be twofold: on the one hand, architectures related to depth itself exhibit good performance (with high global accuracy as mentioned in the figure); on the other hand, as stated above, the communication constraint does not differ significantly from proportional splitting. Therefore, the overall state aligns well with the original design of the corresponding paper. Therefore, the improvement after heterogeneity is not as pronounced.

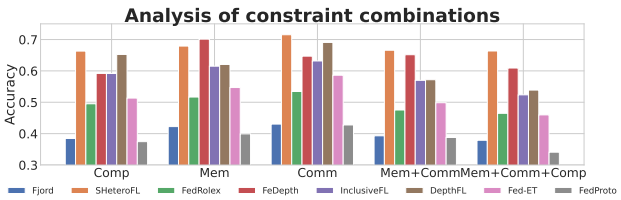


Fig. 7. Analysis of constraint combinations.

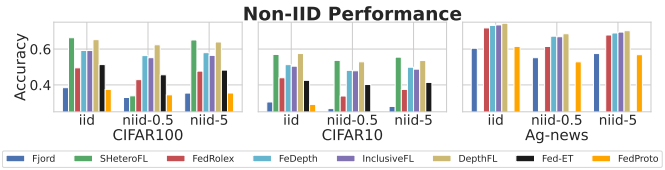


Fig. 8. Non-iid performance on the computation-limited scenario. Here '0.5' and '5' are the alpha of the non-iid dirichlet partition.

> **Observation**: *Communication-Limited MHFL somewhat resembles the proportional splitting in existing literature, as it pays more attention to the number of model parameters. Therefore, the gaps among various algorithms, especially in terms of training speed, are not significant. Additionally, compared to other heterogeneous levels, although depth-level heterogeneity still holds an advantage in global accuracy and training speed, the performance improvement (i.e., effectiveness) brought by heterogeneity is relatively limited.*

C. Results on Memory-Limited MHFL

Global accuracy and time-to-accuracy. Figure 6 shows the results. From the top row, the following phenomenon is noticeable: We *surprisingly* find that DepthFL, which performs well in the above two constraints, no longer exhibits superiority. Its global accuracy has significantly decreased, falling below the previously comparable SHeteroFL and inferior FeDepth. We attribute this to the fact that DepthFL itself requires a substantial amount of memory (shown in Table I). When memory is constrained, DepthFL needs to drastically reduce its size, leading to a decrease in performance. Instead, although FeDepth does not perform well under the computational and communication constraints, its small memory footprint (shown in Table I) allows it to accommodate larger models in this scenario, resulting in better performance.

Stability and effectiveness. We show the results at the bottom of Figure 6. From the figure, we find the following interesting observation: Compared to the two scenarios mentioned above, there is not much variation in the effectiveness trends of different algorithms. However, the conclusion regarding stability is almost reversed. For instance, in the case of CIFAR100, SHeteroFL exhibits the highest stability under the computation and communication constraints. However, for the memory constraint, its stability becomes the lowest among all methods. For Stack Overflow, Fjord demonstrates relatively high stability under the first two scenarios, but faced with memory limitations, its stability jumps to the lowest. We attribute this phenomenon to the significant difference between model parameters and actual device memory usage (as shown in Table I). Therefore, under the memory constraint, the model size of many methods may undergo substantial changes, leading to unstable performance.

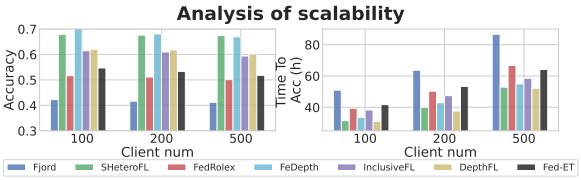


Fig. 9. Analysis of scalability.

> **Observation**: *In practical memory-constrained scenarios, the performance of global accuracy and training speed, apart from the influence of heterogeneous levels and the methods themselves, is significantly affected by the actual memory footprint occupied by the models of different methods. Moreover, it does not exhibit a strong correlation with the types of data tasks. Additionally, concerning the stability, although it remains challenging to identify specific patterns, its trends are precisely opposite to the conclusions drawn under the computation and communication constraints.*

D. Other Analysis

Analysis of constraint combinations. Here we test two types of combinations: communication+memory limited FL and computation+communication+memory limited FL on the CIFAR-100 dataset. Figure 7 demonstrates the accuracy results.We can observe that: Under the combination of communication+memory, the performance of all methods decreases compared with the single restriction and SHeteroFL performs best when various constraints are combined together due to its excellent partitioning method. SHeteroFL's slimmable partition method is well controlled in terms of calculation volume, memory consumption and communication volume, making it more adaptable to various restricted environments.

Analysis of Non-IID Performance. Figure 8 show the results of computational constraints under the non-iid scenario. From the figure, we can observe that although non-iid will degrade the performance, the conclusion is not changed, which means that existing algorithms is robust to the non-iid scenario.

Analysis of scalability. To demonstrate the scalability, here we conduct an experiment to evaluate the convergence-related metrics of various methods with different client numbers on the memory-limited MHFL setting and the CIFAR-100 dataset. As shown in Figure 9, scaling up leads to a decline in convergence performance, including both convergence effectiveness and speed. Among these methods, the width-level heterogeneity demonstrates relatively smaller performance losses, indicating better scalability.

VI. Conclusion

In this work, we construct the first comprehensive platform for MHFL on practical resource constraints and conduct extensive measurements to quantitatively evaluate their performance. The results help reveal a complete landscape among diverse model heterogeneity levels and algorithms on real-edge devices. Based on the results, we summarize strong observations and implications that can be useful to FL developers and researchers of resource-constrained federated systems.

References

[1] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[2] Quande Liu, Cheng Chen, Jing Qin, Qi Dou, and Pheng-Ann Heng. Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1013–1023, 2021.

[3] Bingyan Liu, Yifeng Cai, Ziqi Zhang, Yuanchun Li, Leye Wang, Ding Li, Yao Guo, and Xiangqun Chen. Distfl: Distribution-aware federated learning for mobile scenarios. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(4):1–26, 2021.

[4] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. Pyramidfl: A fine-grained client selection framework for efficient federated learning. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 158–171, 2022.

[5] Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *ICLR*, 2021.

[6] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *NeurIPS*, 2022.

[7] Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. Fedproto: Federated prototype learning across heterogeneous clients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8432–8440, 2022.

[8] Jaemin Shin, Yuanchun Li, Yunxin Liu, and Sung-Ju Lee. Fedbalancer: data and pace control for efficient federated learning on heterogeneous clients. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 436–449, 2022.

[9] Dongqi Cai, Shangguang Wang, Yaozong Wu, Felix Xiaozhu Lin, and Mengwei Xu. Federated few-shot learning for mobile nlp. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2023.

[10] Bingyan Liu, Yifeng Cai, Hongzhe Bi, Ziqi Zhang, Ding Li, Yao Guo, and Xiangqun Chen. Beyond fine-tuning: Efficient and effective fed-tuning for mobile/web users. In *Proceedings of the ACM Web Conference 2023*, pages 2863–2873, 2023.

[11] Yongheng Deng, Weining Chen, Ju Ren, Feng Lyu, Yang Liu, Yunxin Liu, and Yaoxue Zhang. Tailorfl: Dual-personalized federated learning under system and data heterogeneity. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 592–606, 2022.

[12] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021.

[13] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, 33:14068–14080, 2020.

[14] Yae Jee Cho, Andre Manoel, Gauri Joshi, Robert Sim, and Dimitrios Dimitriadis. Heterogeneous ensemble knowledge transfer for training large models in federated learning. *IJCAI*, 2022.

[15] Hong Zhang, Ji Liu, Juncheng Jia, Yang Zhou, Huaiyu Dai, and Dejing Dou. Fedduap: Federated learning with dynamic update and adaptive pruning using shared data on the server. *IJCAI*, 2022.

[16] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[17] Di Chai, Leye Wang, Liu Yang, Junxue Zhang, Kai Chen, and Qiang Yang. Fedeval: A holistic evaluation framework for federated learning. *arXiv preprint arXiv:2011.09655*, 2020.

[18] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. *International Symposium on Reliable Distributed Systems (SRDS)*, 2020.

[19] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*, pages 11814–11827. PMLR, 2022.

[20] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 965–978. IEEE, 2022.

[21] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *Proceedings of the Web Conference 2021*, pages 935–946, 2021.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[26] Kai Zhang, Yutong Dai, Hongyi Wang, Eric Xing, Xun Chen, and Lichao Sun. Memory-adaptive depth-wise heterogenous federated learning. *arXiv preprint arXiv:2303.04887*, 2023.

[27] Ruixuan Liu, Fangzhao Wu, Chuhan Wu, Yanlin Wang, Lingjuan Lyu, Hong Chen, and Xing Xie. No one left behind: Inclusive federated learning over heterogeneous devices. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3398–3406, 2022.

[28] Minjae Kim, Sangyoon Yu, Suhyun Kim, and Soo-Mook Moon. Depthfl: Depthwise federated learning for heterogeneous clients. In *The Eleventh International Conference on Learning Representations*, 2022.

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[30] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[31] tff. Tensorflow federated stack overflow dataset, 2019.

[32] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 54–66, 2021.

[33] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3, 2013.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[35] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[36] Sannara Ek, François Portet, Philippe Lalanda, and German Eduardo Vega Baez. Evaluating federated learning for human activity recognition. In *Workshop AI for Internet of Things, in conjunction with IJCAI-PRICAI 2020*, 2021.

[37] Ai benchmark for different devices. https://ai-benchmark.com/ranking_deeplearning_detailed.html, 2021.

[38] scientiamobile. How much ram is in smartphones, 2022.

BPUFuzzer: Effective Fuzz Testing for Branching Transient Execution Vulnerabilities of RISC-V CPU

Rihui Sun[1], Jin Wu[1], Hanyin Liu[1], Zikang Tao[1], Gang Qu[3], Dongsheng Wang[2], Yongqiang Lyu[2], Jian Dong[1*]
[1]Harbin Institute of Technology, Harbin, Heilongjiang, China
[2]Tsinghua University, Beijing, China
[3]University of Maryland, College Park, MD, US

***Abstract*—This paper presents BPUFuzzer, a fuzz testing tool for detecting branching transient execution vulnerabilities in CPU RTL design. BPUFuzzer addresses two key challenges: generating testcases that capture complex control flows, and extracting essential data from vast hardware states to guide testcase selection. Utilizing a control flow graph-based testcase generation strategy with anomaly detection and employing fitness and coverage metrics, BPUFuzzer works on testcases that cover broader program flows and deliberately selects testcases to discover transient execution vulnerabilities effectively. When applied on RISC-V Boom v3, BPUFuzzer uncovered more Spectre types than the state-of-the-arts, including a previously unidentified variant, named Spectre-LOOP.**

I. Introduction

Transient execution vulnerabilities are a significant hardware security issue in modern CPUs. Attacks like Spectre [1] and Meltdown [2], along with their variants [3]–[11], severely impact the reliability of processor hardware. The majority of these vulnerabilities are caused by the malicious exploitation of the branch prediction unit (BPU), which is a commonly implemented microarchitecture in modern processors designed to optimize pipeline performance by predicting control flow changes thus preventing pipeline stalls [12]. The discovery of these vulnerabilities and their variants has raised new concerns about processor security, as these microarchitecture-based issues cannot be easily fixed by vendors and pose serious threats to information system security.

Recent advancements have been made in the automated detection of transient execution vulnerabilities. Several approaches [13]–[17] focus on testing limited code snippets based on prior knowledge, without addressing a broad range of code patterns. SpecDoctor [18] generates testcases without prior knowledge but imposes constraints that prevent loop generation in its implementation. Furthermore, it lacks coverage feedback mechanisms specific to transient execution vulnerabilities, limiting its ability to detect issues arising from complex hardware states. Moreover, current techniques limit their vulnerabilities detection without in-depth analysis of complex BPU behaviors. Detecting vulnerabilities within BPU poses two significant challenges. First, since the combination of branch instructions that can change the BPU state is very diverse, covering a broad range of control flow patterns is difficult. Second, it is challenging to efficiently search for testcases that tend to trigger transient execution from numerous testcase patterns corresponding to complex BPU states.

In this paper, we present BPUFuzzer, a pre-silicon fuzz testing tool designed to address these challenges and effectively detect branching transient execution vulnerabilities. In order to capture complex control flow patterns, we propose an effective testcase generation and validation method based on control flow graph (CFG). In addition, we propose effective fitness and coverage functions to guide testcase selection, innovatively taking into account the states of the Reorder Buffer (RoB) and the Branch Prediction Unit (BPU). This approach extends fitness and coverage metrics to the microarchitectural component level, improving vulnerability detection efficiency. Through BPUFuzzer testing, we identified a new transient execution pattern, termed the loop pattern, which can manifest as a new Spectre variant called Spectre-Loop. The main contributions of our work are:

- We propose a CFG-based testcase generation that encompasses all possible program control flows, including loop types that were not addressed in previous work.
- We propose an effective feedback method on fuzz testing to guide testcase selection. It significantly enhances the efficiency of uncovering transient execution vulnerabilities in complex hardware designs.
- The proposed fuzz tool successfully identified a new Spectre variant called Spectre-loop on Boom v3 CPU. Experiments also demonstrate that BPUFuzzer uncovers more Spectre types than previous work.

II. Related Work

The automated detection of transient execution vulnerabilities has gained significant attention. Prior work, such as Medusa [13], uses authenticated PoC snippets as test seeds. SpecFuzz [14] simulates branch misprediction in security-sensitive libraries. Absynthe [15] and Osiris [16] focus on finding specific side-channels. These approaches mainly test limited code snippets based on prior knowledge, rather than covering a broad range of code patterns and hardware states. Furthermore, these frameworks test only on post-silicon CPUs, limiting their ability to identify potential transient execution leaks in the pre-silicon design phase before the processor is taped out. In pre-silicon RTL-stage research, Introspectre [17] uses predefined code gadgets with a focus on meltdown-type vulnerabilities. DifuzzRTL [19] and SIGFuzz [20] apply a

*Corresponding author: dan@hit.edu.cn

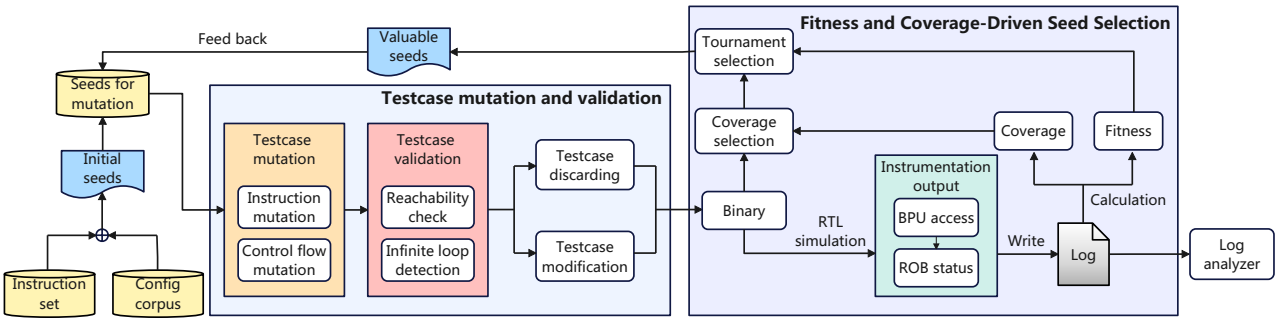


Fig. 1: Overview of BPUFuzzer

coverage based on control signals that drive the muxes to find CPU bugs and side channels. They focus on register states and ignore microarchitecture states, which is essential for transient execution. So their coverage definitions are unsuitable for detecting transient execution vulnerabilities. WhisperFuzz [21] focuses on timing behaviors changing by triggering different microarchitectural state transitions. In contrast, BPUFuzzer in-depth studies the states of microarchitectural components, including BPU and RoB. The coverage metrics formed by these states are used to search for testcases tend to trigger branching transient execution, significantly improving the efficiency of detecting vulnerabilities. SpecDoctor [18] generates testcases using CFG construction. To avoid invalid testcases in implementation, it enforces constraints that prevent loop generation. However, loop is an important control flow transfer pattern. Moreover, it does not provide coverage feedback mechanisms, limits its detection from complex hardware states. In contrast, BPUFuzzer is capable of discovering transient execution caused by any control flow transfer. Moreover, BPUFuzzer tailored coverage and fitness metrics optimized for transient execution. Owing to these two improvements, BPUFuzzer surpasses previous work in both the scope and efficiency of detecting transient execution vulnerabilities.

III. Overview of BPUFuzzer

In this section, we present an overview of the architecture of our transient execution vulnerability fuzz testing tool, BPUFuzzer. We consider the following threat model which aligns with the mainstream studies in discovering transient execution vulnerabilities such as Spectre and Meltdown:

(1) The attacker is a user without privilege, the victim is the privileged kernel. The attacker and the victim locate on the same logical core;

(2) The attacker can invoke the victim through system calls.

As shown in Figure 1, our tool includes two main modules to address the challenges outlined in Section I. The testcase mutation and validation module generates testcases for fuzz testing. To maximize the range of control flow patterns in testcase generation, We employ a modified CFG-based technique that includes instruction mutations and control flow mutations without any pattern constraints. A testcase validation submodule is also included to detect and avoid infinite loops. These methods enable us to produce a greater variety of control flow patterns compared with state-of-the-art tools.

The fitness and coverage-driven seed selection module select testcases with high tendency of transient execution and feed them back as seeds for mutation. Aimed at selecting valuable testcases efficiently, BPUFuzzer incorporates hardware information through instrumentation, including the status of BPU and RoB, which is closely related to transient execution. These pieces of information are logged and used to calculate the fitness function and testcase coverage. Specifically, testcases are first selected based on whether they cover new hardware states, as determined by coverage calculation, and then undergo tournament selection based on fitness value. Additionally, a log analyzer detects transient execution risks through the collected data and classifies them based on the originating microarchitectural component.

Sections IV and V will detail the working principles of the testcase mutation and validation module, as well as the fitness and coverage-driven seed selection module.

IV. Testcase Mutation and Validation

The testcase mutation and validation module aims to generate testcases that cover the maximum possible range of control flow patterns. This module consists of two components: mutation of instructions within a basic block and mutation of control flow instructions. To ensure that generated testcases maximize control flow coverage and execute to termination, a validation module checks each testcase after mutation.

A. Mutation Rule of a Basic Block

Figure 2 shows the structure of a basic block (BB) and all possible mutation behaviors. A BB must end with either a conditional or unconditional jump instruction, with no jump instructions allowed at any other position within the block. During mutation, various transformations may occur within the BB, including the insertion, removal, substitution of instructions, and modifications to jump instruction targets.

B. Control Flow Related Instructions on RISC-V

In RISC-V instruction sets, there are two types of instructions related to control flow transfer, unconditional jump instructions and conditional branch instructions, shown in Table I. A conditional branch adds two outgoing edges to the basic block in the CFG, while an unconditional jump only adds one outgoing edge.

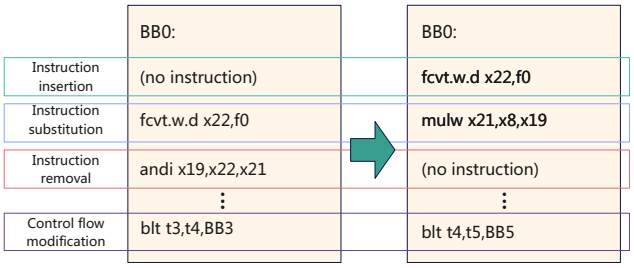


Fig. 2: Possible mutation operations on a basic block

TABLE I: Control Flow Related RISC-V Instructions

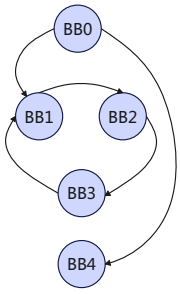
Inst	Description
beq, rs1, rs2, imm	if(rs1 == rs2) PC += imm
bne, rs1, rs2, imm	if(rs1 != rs2) PC += imm
blt, rs1, rs2, imm	if(rs1 $<$ rs2) PC += imm
bge, rs1, rs2, imm	if(rs1 $\geq$ rs2) PC += imm
bltu, rs1, rs2, imm	if(rs1 $<$ rs2) PC += imm
bgeu, rs1, rs2, imm	if(rs1 $\geq$ rs2) PC += imm
jal, rd, imm	rd = PC+4; PC += imm
jalr, rd, rs1, imm	rd = PC+4; PC = rs1 + imm

C. Testcase Validation

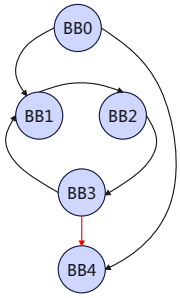
To maximize coverage of flow transfer scenarios in CFG generation, we do not restrict jump or branch targets during mutation. However, arbitrary jump combinations may create infinite loops, so we apply a validation process to detect and correct invalid testcases.

All loops in a CFG have the potential to cause infinite loops. Based on the types of jumps at the end of the BBs of a loop, we classify the causes of infinite loops into the following categories:

- **Unconditional jumps**: In this type, all BBs within the loop end with unconditional jumps. Consequently, once the control flow enters such a loop, it becomes impossible to exit. An example of this situation is illustrated in Figure 3(a), once the control flow jumps from BB0 to BB1, it will never exit. We will refer to this type of testcase as type 1 testcase in the following discussion.



(a) Infinite loop caused by unconditional jumps



(b) Infinite loop caused by conditional branches that always branch in the same direction

Fig. 3: Two types of invalid testcases

- **Conditional jumps with fixed direction**: Although at least one BB within these loops end with conditional jumps that could theoretically allow for an exit, arbitrary mutation operations may cause the direction of the conditional jump to remain unchanged, resulting in an infinite loop. As shown in Figure 3(b), although theoretically the unconditional jump at the end of BB3 could jump to BB4, improper instruction mutation may causes the value comparison between the two source registers of BB3's conditional jump to remain unchanged, preventing the jump from BB3 to BB4 from ever occurring(illustrated with a red arrow) and thus forming an infinite loop. We will refer to this type as type 2 testcase.

Notation	Description
BB.jmpIns	Last jump instruction of basic block `BB`.
BB.jmpIns.name	Name of the instruction.
BB.jmpIns.rs1	Source register 1 of the jump instruction.
BB.jmpIns.rs2	Source register 2 of the jump instruction.
instruction.dest	Destination register of arithmetic & logical instructions.

TABLE II: Notations used in Algorithm 1.

Algorithm 1: Type 2 testcases modification

```
Input : BBs containing conditional jump instructions in a cycle
Output: Modified BBs
foreach BB in BBs do
    if BB.jmpIns.name in {blt, bge, bltu, bgeu} then
        Set BB.jmpIns.rs1 to t1;
        Set BB.jmpIns.rs2 to t2;
        foreach instruction in BB do
            if instruction.dest in {t1, t2} then
                Delete instruction;
        ▷ Adjust the two source registers according to their current states;
        Insert instruction slti t3, t1, t2;
        Insert instruction xor t4, t3, 1;
        Insert instruction slli t3, t3, 6;
        Insert instruction slli t4, t4, 6;
        Insert instruction add t1, t1, t3;
        Insert instruction add t2, t2, t4;
        Insert instruction addi t1, t1, -32;
        Insert instruction addi t2, t2, -32;
    if BB.jmpIns.name in {beq, bne} then
        Set BB.jmpIns.rs1 to zero;
        Set BB.jmpIns.rs2 to t2;
        foreach instruction in BB do
            if instruction.dest = t2 then
                Delete instruction;
        Insert instruction srli t2, 1;
```

To identify and correct the two types of invalid testcases mentioned above, we employed a two-phase detection and modification approach after each round of mutation.

- **Discarding type 1 testcases**: We verify that all basic blocks reachable from the starting BB can also reach the terminating BB, any testcase that fails to meet this criterion will be discarded. Thus the type 1 invalid testcase can be avoided.

- **Modifying type 2 testcases**: After all type 1 testcases are discarded, each rest cycle in generated CFGs contains at least one conditional jump. In order to allow the direction of conditional jumps to change, we implement the following two measures: (I) Inserting instructions to adjust the two source registers according to their current states. For blt, bge, $bltu$ and $bgeu$, if the value of register a exceeds that of register b, we add instructions to reduce the value in register a and increase the value in register b, and vice versa. For bne and beq, since the probability of two 32-bit numbers being equal is extremely low, we implemented a specific approach: one source register is set to special zero register defined in RV32I [22], which always holds the value zero, while the other register is initialized with a non-zero value. Additionally, we insert a logical right shift instruction, consequently, the values in the two registers are always unequal initially, but after a finite number of loops, they will become equal. (II) Ensuring that no other instructions within the loop modify the contents of registers related to the jump. We deleted all instructions modifying any of the two resource registers before we insert the above instructions. The implementation can be found in Algorithm 1. The notations used in it are defined in Table II.

V. Fitness and Coverage-Driven Seed Selection

In this section, we present our implementation of the coverage and fitness functions and how they enhance our fuzz tool's ability to select more effective testcases.

A. Hardware Instrumentation

We collect essential microarchitectural data during the hardware instrumentation phase. The data that is interested is collected based on our study of the branching transient execution. The specific microarchitectural details are listed in Table III. We primarily gather information on BPU access and RoB status, which clearly reveals the behavior of transient execution and distinguishes them from normal program execution.

B. Design of Fitness Function and Coverage

Using the data in Table III, we designed a fitness function to estimate the likelihood of a testcase mutating into one that triggers transient execution. Additionally, we developed a coverage metric to efficiently search for testcases that tend to trigger transient execution from numerous testcase patterns corresponding to complex BPU states.

The formula of the fitness function is as follows:

$$fitness = mispredict_count \times weight + btb_hit_count \quad (1)$$

The fitness function is a weighted sum of the misprediction count in the RoB and the BTB hit count. The misprediction count plays a critical role in triggering transient execution, while the BTB hit count is associated with the attacker's training phase in vulnerabilities like Spectre and its variants. In our testcase selection, greater emphasis is placed on the misprediction count, which is assigned a larger weight.

TABLE III: Hardware instrumentation information summary

uArch Comp.	Instrumentation Output	Description
RoB	mispredict_pc	The program counter of the mispredicted branch instruction.
	mispredict_ins_code	The instruction code of the mispredicted branch instruction.
	mispredict_target_addr	Mispredicted target address.
	mispredict_count	Total count of mispredictions.
BTB	wbtb_clock	Clock count when writing BTB.
	wbtb_bank_sel	Selection of bank when writing BTB, part of BTB index rule.
	wbtb_meta_bank_sel	Selection of bank when writing BTB meta data(tag and type), part of BTB index rule.
	wbtb_way	Selection of way when writing BTB, part of BTB index rule.
	wbtb_set	Selection of set when writing BTB, part of BTB index rule.
	wbtb_branch_offset	Offset of the branch ,i.e.target_pc-current_pc
	wbtb_branch_type	Type of branch instruction, i.e.conditional/unconditional
	wbtb_branch_tag	Tag associated with the branch, i.e. part of PC
	btb_hit_count	Count of BTB hits.
	btb_hit_set	The hit BTB set.
uBTB	ubtb_hit_count	Count of uBTB hits.
	ubtb_hit_tag	Tag in hit uBTB entry.
LOOP	loop_hit_count	Count of LOOP hits.
	loop_hit_set	Set of hit LOOP entry.
TAGE	tage_hit_count	Count of TAGE hits.
	tage_hit_table_idx	Index of hit TAGE table.
	tage_counter_count	Count of TAGE 3-bit counter.

For coverage, since each control flow transfer affects BTB states, we incorporate several branch prediction-related values into our coverage metric, which is shown in Algorithm 2.

In transient execution exploitation, the prediction training and triggering rely on the branch/jump instructions with specific instruction addresses, jump directions and target addresses. These values are stored in BTB and BTB meta table. To analyze these behaviors, we extract the relevant information and hash each corresponding piece of data in the log to generate binary feature values ($t.btb$, $t.btb_meta$). $t.btb$ represents the features of jump directions and target addresses, while $t.btb_meta$ represents the features of instruction addresses. If there is a new feature value, there is a control flow transfer instruction whose address or target address has changed. It may bring new transient execution possibility, reflected in an increase in test coverage.

Based on our fitness function and coverage, we select testcases to enhance defect detection effectiveness. First, we prioritize testcases that generate new coverage. If a testcase

Algorithm 2: Coverage Calculation

Input : A set of testcase and their logs
Output: whether the testcase is a new coverage

```
cov_list = empty set;
cov_sum = 0;
foreach t in test_case_set do
    t.btb = 0;
    t.btb_meta = 0;
    foreach line in test_case.log do
        btb_info = wbtb_set + wbtb_way +
          wbtb_branch_offset + wbtb_bank_sel;
        meta_info = wbtb_set + wbtb_way +
          wbtb_branch_tag + wbtb_branch_type +
          wbtb_meta_bank_sel;
        t.btb = t.btb xor btb_info;
        t.btb_meta = t.btb_meta xor meta_info;
    if (t.btb,t.btb_meta) not in cov_list then
        cov_list.insert(t.btb,t.btb_meta);
        t.coverage = true;
        cov_sum += 1;
    else
        t.coverage = false;
```

generates new coverage, it is added to the candidate set, while those without new coverage have a 5% probability of being included to prevent local optima. All testcases in the candidate set then undergo a tournament selection based on the fitness to form the next generation of testcases. This approach ensures that testing progressively covers more hardware states and that testcases increasingly tend to produce transient execution. The discovery of a new Spectre variant, called Spectre-Loop (detailed in Section VII), is owed to this methodology.

VI. Evaluation of BPUFuzzer

In this section, we present experiments that demonstrate the advantage of our proposed BPUFuzzer in terms of testcase generation pattern, coverage and vulnerability detection. We compare them with existing state-of-the-art tool, SpecDoctor.

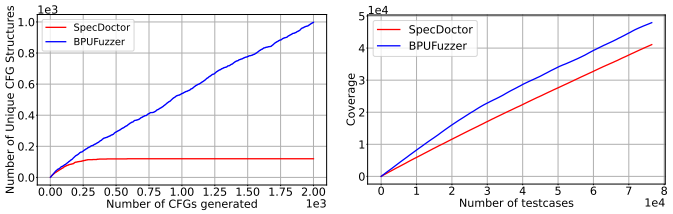
A. Experiment Setup

RISC-V CPU under test. We evaluated BPUFuzzer on the open-source out-of-order RISC-V CPU: Boom V3 [23]. It supports RV64G ISA which includes supervisor level instructions and user level instructions. It contains the out-of-order microarchitectural units including RoB and branch predictors.

Fuzz testing environment. We evaluated BPUFuzzer with a RTL simulator called Chipyard [24]. It is a modified Verilator simulator designed for Boom RISC-V CPU. All the fuzzing experiments are conducted on a machine equipped with an Intel(R) Xeon(R) E5-2670 v3 CPU at 2.30GHz with 64GB of RAM, which runs Ubuntu 22.04 LTS.

B. Testcase Generation Pattern

In section IV, we propose a testcase mutation and validation method designed to cover diverse control flow transfer patterns, corresponding to mutually non-isomorphic control flow graphs [25]. We verify the number of patterns generated and compare it with SpecDoctor.



(a) comparison of testcase pattern with SpecDoctor (b) comparison of test coverage with SpecDoctor

Fig. 4: Comparison with SpecDoctor

In Figure 4(a), we applied a basic block limit of 6 in the testcase generation process for both our tool and SpecDoctor. In SpecDoctor, basic blocks are connected by conditional jump instructions, and each block can only link to subsequent blocks, preventing loop structures. Under these constraints, SpecDoctor can generate only 120 unique testcase patterns, equivalent to 5 factorial. In contrast, our tool produces a greater variety of control flow transfer patterns, including testcases with loops that execute successfully.

C. Coverage

In Section V, we introduced a fitness function and coverage design based on microarchitectural state to guide testcase selection and mutation, and conducted a comparative experiment with SpecDoctor in Figure 4(b). Using the same hardware setup and the same testcase limit of 76,000, we tested on both tools respectively. SpecDoctor generated 34,916 testcases that do not generate new hardware states, which is unprofitable for transient execution detection. BPUFuzzer covered 6,846 more hardware states than SpecDoctor, leading by 16.7% in coverage. We also ran BPUFuzzer with and without coverage feedback under the same hardware setup for 40 hours. With coverage feedback, BPUFuzzer identified an average of 12.89 testcases with transient execution risks per minute, compared to 11.26 testcases per minute without coverage feedback. This represents a 14.5% improvement in transient execution detection efficiency, highlighting the benefits of coverage feedback.

D. Vulnerability Detection

We ran BPUFuzzer and SpecDoctor under the same hardware setup for a week and classified found vulnerability types following previous research [12]. The found vulnerability types comparison is shown in Table IV. BPUFuzzer successfully detected transient execution in micro BTB and LOOP while SpecDoctor do not.

In Boom v3, micro BTB is a prediction unit that stores local target addresses. It utilizes a fully-associative index method and only works when the corresponding BTB entry is empty. Spectre_uBTB needs to train the micro BTB while clearing the BTB entries. It requires the precise coordination of branch instructions in a wider range of instruction addresses than Spectre_BTB. However, the range of instruction addresses is relatively limited in SpecDoctor, makes it incapable of generating this vulnerability type.

TABLE IV: Found vulnerabilities summary and comparison with SpecDoctor

Category	Name	Description	BPUFuzzer	SpecDoctor
Bound Check Bypass	Spectre_BIM_IP	Exploitation of Bi-Modal Table.	✓	✓
	Spectre_uBTB_IP	Exploitation of micro BTB.	✓	×
	Spectre_TAGE_IP	Exploitation of TAGE.	✓	✓
	Spectre_TAGE_OP	Exploitation of TAGE with Table entry collision.	✓	✓
	Spectre_Loop_IP	Exploitation of Loop.	✓	×
	Spectre_Loop_OP	Exploitation of Loop with Table entry collision.	✓	×
Branch Target Injection	Spectre_uBTB_IP	Exploitation of micro BTB.	✓	×
	Spectre_BTB_IP	Exploitation of BTB.	✓	✓

VII. New Finding of BPUFuzzer: Spectre-Loop

During our analyzing on experiment results, we found some vulnerabilities have significantly large btb_hit_count in their fitness value. Through our further analysis, We found that they all have a loop structure. We eventually confirmed that they exploited a loop prediction optimization mechanism in Boom v3 and formed a new Spectre variant, termed Spectre-Loop.

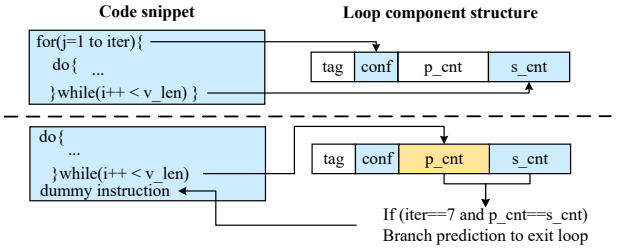
A. Details of Spectre-Loop

Figure 5(a) describes the working mechanism of LOOP, which predicts long loop exit timing of up to 1024. LOOP table has 4 parts: tag derived from the instruction's PC, is used to identify matching entries in the table. $iter$ is a three-bit saturating counter records times the loop is executed as a whole. p_cnt represents the present count of times the loop body has entered the pipeline during loop iteration, and it is updated each time the loop body enters the pipeline. s_cnt records the sum number of iterations after the loop code has fully completed execution. When the loop code runs again, if $iter$ reaches the threshold and p_cnt equals s_cnt, the prediction will indicate a loop exit. In summary, It is a prediction mechanism triggered by a loop code structure, this control flow pattern is not included in the testcase generation implementation of SpecDoctor. Thus SpecDoctor unable to detect transient execution caused by this prediction mechanism.

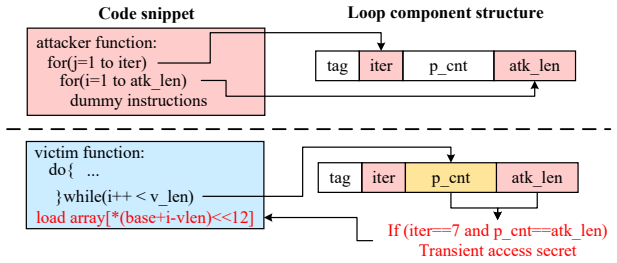
Figure 5(b) demonstrates the attack flow of spectre-loop. First, in the attacker phase, the loop is executed multiple times to train the s_cnt to the value atk_len. Then, in the trigger phase, the loop predict loop exit early after executing atk_len times. During speculative execution, this results in a transient out-of-bounds access on the offset of $atk_len - v_len$ (with $atk_len < v_len$). By controlling atk_len, an attacker can bypass the bound check and transiently access arbitrary locations, enabling access to specific secret data. The attacker can subsequently recover the transiently accessed secret data using a simple flush+reload technique.

B. Analysis of Spectre-Loop

Spectre-Loop successfully retrieved the secret of 1,024 bits with the error rate under 0.01%. Each bit leakage cost 220k clock cycles on the Chipyard simulator. The leakage rate is 4.9 Kb/s assuming a CPU clock rate of 1GHz.



(a) loop prediction optimization mechanism



(b) attack scenario of spectre-loop

Fig. 5: Loop prediction mechanism & Process of Spectre-loop

VIII. Conclusion

In this paper, we propose a pre-silicon fuzz testing tool, BPUFuzzer, to detect transient execution vulnerabilities. We propose a novel modified CFG-based test case generation method that outperforms all existing approaches in generating a broader range of control flow patterns, particularly by incorporating loop patterns that have not been considered in any prior work. Additionally, We develop novel coverage and fitness metrics tailored to the characteristics of transient execution, significantly enhancing the efficiency in detecting transient execution vulnerabilities. Finally, BPUFuzzer finds a newly discovered Spectre variant, termed Spectre-Loop, which exploits a previously overlooked prediction mechanism to perform a Bounds Check Bypass attack.

Acknowledgment

This work was supported by the Fundamental Research Funds for the Central Universities (Grant No. HIT.OCEF.2024012) and the National Natural Science Foundation of China (Grant No. U23B2041, U24A6009).

References

[1] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1–19.

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990.

[3] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. B. Abu-Ghazaleh, "Spectre returns! speculation attacks using the return stack buffer," in *12th USENIX Workshop on Offensive Technologies, WOOT 2018, Baltimore, MD, USA, August 13-14, 2018*, C. Rossow and Y. Younan, Eds. USENIX Association, 2018.

[4] J. Wikner and K. Razavi, "RETBLEED: arbitrary speculative code execution with return instructions," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 3825–3842.

[5] H. Yavarzadeh, A. Agarwal, M. Christman, C. Garman, D. Genkin, A. Kwong, D. Moghimi, D. Stefan, K. Taram, and D. M. Tullsen, "Pathfinder: High-resolution control-flow attacks exploiting the conditional branch predictor," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, R. Gupta, N. B. Abu-Ghazaleh, M. Musuvathi, and D. Tsafrir, Eds. ACM, 2024, pp. 770–784.

[6] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "RIDL: rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 88–105.

[7] M. Schwarz, M. Lipp, D. Moghimi, J. V. Bulck, J. Stecklina, T. Prescher, and D. Gruss, "Zombieload: Cross-privilege-boundary data sampling," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 753–768.

[8] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lippi, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, and F. Piessens, "Lvi: Hijacking transient execution through microarchitectural load value injection," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 54–72.

[9] G. Maisuradze and C. Rossow, "ret2spec: Speculative execution using return stack buffers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2109–2122.

[10] E. Barberis, P. Frigo, M. Muench, H. Bos, and C. Giuffrida, "Branch history injection: On the effectiveness of hardware mitigations against Cross-Privilege spectre-v2 attacks," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 971–988.

[11] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage against the machine clear: A systematic analysis of machine clears and their implications for transient execution attacks," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. D. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 1451–1468.

[12] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 249–266.

[13] D. Moghimi, M. Lipp, B. Sunar, and M. Schwarz, "Medusa: Microarchitectural data leakage via automated attack synthesis," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1427–1444.

[14] O. Oleksenko, B. Trach, M. Silberstein, and C. Fetzer, "SpecFuzz: Bringing spectre-type vulnerabilities to the surface," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1481–1498.

[15] B. Gras, C. Giuffrida, M. Kurth, H. Bos, and K. Razavi, "Absynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[16] D. Weber, A. Ibrahim, H. Nemati, M. Schwarz, and C. Rossow, "Osiris: Automated discovery of microarchitectural side channels," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1415–1432. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/weber

[17] M. Ghaniyoun, K. Barber, Y. Zhang, and R. Teodorescu, "INTROSPECTRE: A pre-silicon framework for discovery and analysis of transient execution vulnerabilities," in *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Virtual Event / Valencia, Spain, June 14-18, 2021*. IEEE, 2021, pp. 874–887.

[18] J. Hur, S. Song, S. Kim, and B. Lee, "Specdoctor: Differential fuzz testing to find transient execution vulnerabilities," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1473–1487.

[19] J. Hur, S. Song, D. Kwon, E. Baek, J. Kim, and B. Lee, "Difuzzrtl: Differential fuzz testing to find cpu bugs," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1286–1303.

[20] C. Rajapaksha, L. Delshadtehrani, M. Egele, and A. Joshi, "Sigfuzz: A framework for discovering microarchitectural timing side channels," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–6.

[21] P. Borkar, C. Chen, M. Rostami, N. Singh, R. Kande, A. Sadeghi, C. Rebeiro, and J. Rajendran, "Whisperfuzz: White-box fuzzing for detecting and locating timing vulnerabilities in processors," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024.

[22] A. Waterman and K. Asanović, "The risc-v instruction set manual, volume i: User-level isa, document version 20191214-draft," *RISC-V Foundation*, December 2019.

[23] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," May 2020.

[24] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[25] L. Babai, "Group, graphs, algorithms: the graph isomorphism problem," in *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*. World Scientific, 2018, pp. 3319–3336.

An Efficient Compute-in-Memory based Accelerator for Point-based Point Cloud Neural Networks

Xipeng Lin, Cong Wang, Shanshi Huang, Hongwu Jiang
The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China
Email: {xlin297, cwang841}@connect.hkust-gz.edu.cn, {shanshihuang, hongwujiang}@hkust-gz.edu.cn

***Abstract*—Point-based point cloud neural networks (PNNs) have shown remarkable accuracy in various applications. However, the performance of PNNs is usually limited on resource-constrained edge devices. This paper presents Point-CIM, an efficient Compute-in-Memory (CIM) based accelerator for PNNs, with software and hardware co-optimization. A reconfigurable CIM unit, which exploits the inherent zero-bit-skipping capability of CIM array, is designed to efficiently process Multiply-Accumulate (MAC) operations on decomposed feature data with high bit sparsity. A Voxel-Morton-sorted Partitioning (VMP) method combined with Channel-wise Minimum (CM) base point selection is proposed to improve the decomposition bit sparsity further and reduce the hardware implementation overhead. Additionally, a detailed Bit-level Truncation Quantization (BTQ) method is proposed to directly compress the bit-width of the offset without incurring any additional hardware overhead. Based on this decomposition method, a Pre-decomposition (PD) data movement strategy is employed to reduce data transfers of intermediate features between the on-chip buffer and CIM array. Extensive evaluation experiments on multiple datasets show that Point-CIM achieves an average speedup of 1.69× to 9.63×, and 3.11× to 17.32× improvement in energy efficiency, compared to state-of-the-art accelerators and general-purpose processors.**

***Index Terms*—compute-in-memory, accelerator, point neural network, bit truncation**

I. Introduction

3D point cloud data captured by LIDAR sensors or RGB cameras can accurately model the shape and surface characteristics of objects and environments. Unlike 2D pixels, 3D point cloud data is inherently unstructured and contains spatial positioning information, making its data processing and analysis complex and demanding. To address the challenges, Point-based PNNs, like PointNet [1] and PointNet++ [2], have been proposed to learn embedding features from raw points directly.

PointNet++, an extension of PointNet, specializes in capturing local structures and hierarchical features of 3D point clouds. Due to its promising performance, PointNet++ has become the foundational architecture for most Point-based PNNs [3]–[5]. Although Point-based PNNs show excellent accuracy, their heavy computational demands limit their deployment in resource-constrained edge devices. To tackle this problem, various software-hardware co-optimization methods have been proposed to improve the hardware efficiency. Most previous works [6]–[8] mainly focus on optimizing the farthest point sampling (FPS) and neighbor search stages, whereas the feature learning stage, typically executed by multi-layer perceptron (MLP), receives less attention. However, this stage

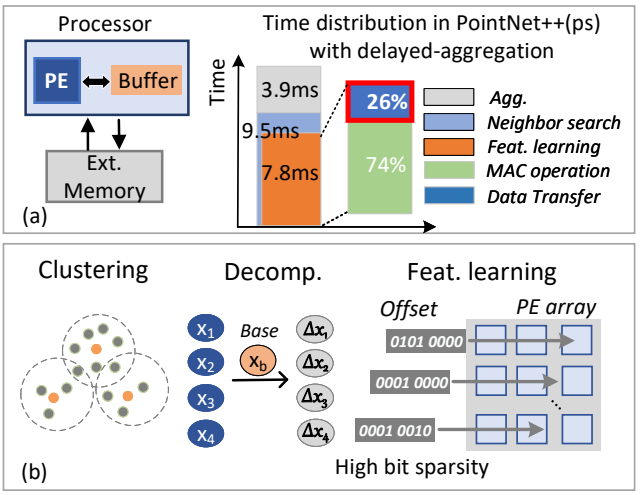


Fig. 1. (a) Time distribution in the point cloud processing task [9]; (b) overview of decomposition-based method.

takes more than half of the computation time of the PointNet++ executed on GPUs (Fig. 1(a)). Recently, several works revealed that the computation efficiency could be improved by exploiting the similarity in the features of points in closer spatial proximity [10]–[12]. As illustrated in Fig. 1(b), this insight allows for the decomposition of feature data into base and offset. The offset contains a higher proportion of zero-value bits and has a more concentrated distribution, enabling a further reduction in redundant computations of offset data. To fully convert this software optimization to real hardware performance improvement, customized processing units are designed to support zero-skipping operations and distribution statistics analysis for dynamic precision quantization on offsets [11], [12]. While previous works focus on the acceleration of MAC operations, the massive data movements also introduce substantial latency during the feature learning stage. As shown in Fig. 1(a), data transfer between the external memory and the processing units accounts for 26% of the processing time during the feature learning stage (tested on Nvidia Jetson TX2 [13]), which limits further improvement in the accelerator throughput.

Compute-in-memory (CIM) has emerged as a promising solution for data-centric applications to reduce the data movements [14]. By integrating the computation units into memory units and employing a weight stationary dataflow for MAC

operations, CIM avoids the data transfer of weight parameters, thus enhancing both the speed and energy efficiency of accelerators. Additionally, as the MAC operations in the CIM array are generally done in a digit-decomposed manner with binary inputs, zero-bits in the inputs are encoded as 0V and thus consume no energy [15]. In other words, CIM inherently supports zero-skipping operations without requiring any extra specialized designs in terms of energy consumption. Consequently, the energy efficiency of CIM-based accelerators is highly sensitive to the bit sparsity of inputs. On the other hand, the latency of the MAC operations are proportional to the bit-width of input data, which is directly affected by the quantization method. Furthermore, although the weight transfers are eliminated, there is still data movement of intermediate features on-chip. The latency and energy consumption caused by it remain significant overheads that cannot be overlooked [16].

To address the aforementioned problems and fully utilize the hardware advantages of CIM, we propose Point-CIM, an efficient CIM-based accelerator for PNNs. Software-hardware co-optimization techniques have been applied to improve the efficiency of Point-CIM. Our key contributions can be summarized as follows:

- We propose a reconfigurable CIM-based processing unit to support MAC operations on decomposed inputs with high bit sparsity. Two different modes are designed to process base and offset calculations, respectively. To further improve the hardware performance, a Pre-decomposition (PD) dataflow is employed to reduce data transfers of intermediate feature results on chip.
- We introduce a Voxel-Morton-sorted Partitioning (VMP) method combined with Channel-wise Minimum (CM) base point selection to ensure positive offset values and maintain high bit sparsity, which benefits the subsequent CIM computing scheme. Additionally, we introduce a detailed Bit-level Truncation Quantization (BTQ) method, designed to directly compress the bit-width of the offset without incurring any additional hardware overhead.
- Evaluated on various datasets, Point-CIM can achieve an average speedup ranging from $1.69\times$ to $9.63\times$, along with $3.11\times$ to $17.32\times$ improvement in energy efficiency, compared to state-of-the-art accelerators and general-purpose processors.

II. Background

A. Point-based Point Cloud Neural Networks

Point-based PNNs typically consist of several Set Abstraction (SA) layers in serial. Each SA layer is composed of FPS, neighbor search, and feature learning stages. Firstly, FPS is applied to select a subset of points that represents the overall structure of the point cloud. Next, the neighbor search groups neighboring points around each sampled point. Finally, MLP layers are performed to extract the local feature within each group during the feature learning stage.

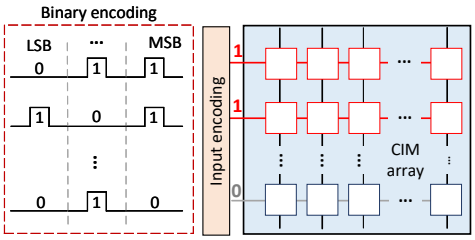


Fig. 2. The binary input encoding process for CIM.

Given an FPS-sampled point x_i and its neighboring group $\mathcal{N}(x_i)$, the final output of each SA layer is determined by

$$y_i = \max_{1 \le j \le J} \sigma(w \cdot (x_i^{(j)} - x_i))$$

where w represents the weights, $x_i^{(j)}$ denotes the j-th neighboring point in $\mathcal{N}(x_i)$, and $\sigma(\cdot)$ is the activation function (e.g. ReLU).

B. Feature Similarity in Close Spatial Proximity

Several previous works [10]–[12] have observed that points located near each other in space tend to share similar geometric features. This property can be leveraged to decompose features into two components: a base and an offset. For points within the same cluster, denoted as $x_c^k \in \mathcal{C}(x_c)$. They share a common base point x_c, with each point can be represented as:

$$x_c^k = x_c + (x_c^k - x_c) = x_c + \Delta x^k$$

where Δx^k represents the offset value of each point in this cluster. After this decomposition, the convolutional operation can be transformed into:

$$w \cdot x_c^k = w \cdot x_c + w \cdot \Delta x^k$$

which offers three advantages in theory. Firstly, since all points within the same cluster share a common base, the convolutional result of the base point $w \cdot x_c$ can be reused for all points of this cluster. Secondly, points that are close in space often show strong feature similarity, meaning that the offset Δx^k tends to have a more concentrated distribution. This allows Δx^k to be quantized with lower precision. Thirdly, the effective bits of each value are reduced greatly, which enables reduce redundant computations by skipping unnecessary zero-bit calculations.

C. CIM-based CNN accelerator

To tackle the memory wall bottleneck, researchers have introduced the in-memory computing paradigm, where computations are conducted directly within the memory array [17]–[19]. In CIM, the weight parameters are mapped into the crossbar memory array. As illustrated in Fig. 2, to perform MAC operations, the input vector is usually encoded into a binary sequence, with positive numbers represented in their original form and negative numbers in two's complement. Zero volts and power supply voltages are used to represent

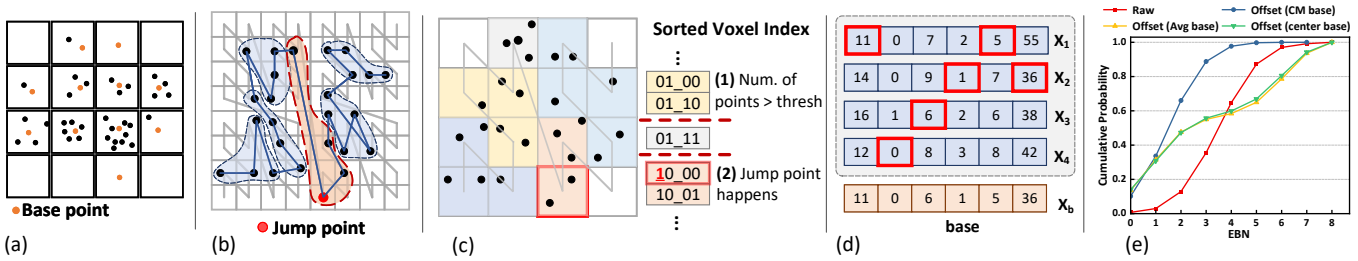


Fig. 3. (a) Uniform voxel partition method; (b) morton-sorted partition method; (c) Voxel-Morton-sorted Partition method (take a 2D space with Morton encoding using 2 MSBs as an example); (d) Channel-wise Minimum base point selection; (e) CDF of EBN for raw data and offsets in SA1 of PointNet++(m) with different base types.

logic "0" and "1", respectively. The "0" input bits, in terms of 0V, do not contribute any current, consuming no energy cost. Accordingly, the CIM paradigm can naturally support skip zero bits of the input and the energy efficiency of the CIM-based accelerators is significantly affected by the bit sparsity of input series.

III. Methodology

This section discusses the software-hardware co-design techniques adopted for Point-CIM. In this section, all models' weights and activations are originally quantized to 8 bits. Additionally, two definitions need to be clarified:

- Effective Bit Number (EBN): This refers to the number of bits equal to "1" in a fixed-point value, representing the sparsity of the value at the bit level. Lower EBN means higher bit-level sparsity.
- Effective Bit Length (EBL): This refers to the valid bit length of a fixed-point value from the highest non-zero bit to the least significant bit (LSB).

As an example, for an 8-bit fixed-point number $00101010_{(2)}$, the EBN is 3, and the EBL is 6.

A. Partitioning Method and Base Point selection

Before decomposing the features, it is necessary to partition the 3D space into clusters of spatially proximate points. GDPCA [11] partitions the 3D space into uniformly distributed, fixed-size voxels. However, the number of points within each voxel may vary greatly, leading to significant workload imbalances, as shown in Fig. 3(a). To balance the size of the clusters, MoC [12] utilizes Morton code to encode and sort the coordinates of all points. All points are partitioned into uniform regions with fixed intervals. However, as shown in Fig. 3(b), the jump points usually reflect huge changes in spatial location, which will be clustered into wrong groups with uniform partitions. Additionally, for tasks with large input sizes, the significant number of points poses challenges for sorting circuits. For example, in the S3DIS dataset [20], each input contains 4096 points. To balance the computing efficiency and accuracy performance, a Voxel-Morton-sorted Partitioning (VMP) method is proposed in this work.

Voxel-Morton-sorted Partitioning (VMP) method. As shown in Fig. 3(c). Instead of encoding the full precision

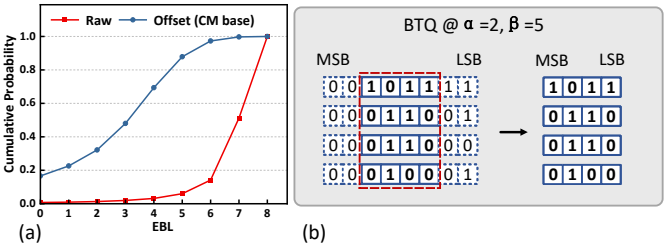


Fig. 4. (a) CDF of EBL for raw data and offsets in SA1 of PointNet++(m); (b) Bit-level Truncation Quantization method (Take $\alpha = 2$ and $\beta = 5$ for example).

of coordinates, we only encode the first 3 MSBs of each coordinate with Morton code (e.g. [101xxx, 001xxx, 010xxx] is encoded as 001_100_011). In this way, the 3D space with xyz dimension will be partitioned into uniform 512 ($8 \times 8 \times 8$) voxels with index encoded with 9 bits. Consequently, we only need to sort 512 blocks, rather than all points. The partitioning is conducted based on two principles: (1) the total number of each partition's points exceeds the workload threshold; (2) an index jump occurs. In our setting, the workload threshold is defined as the ratio of the number of points to the number of partitions. The jump point is identified when the 3rd MSB of the voxel index flips, indicating a location jump in space. As a result, points from different voxels around jump point will not be encoded to the same partition. Inside each voxel, the points are grouped by storage ordering with loose locality constraints.

Channel-wise Minimum (CM) Base Point. The base point of the decomposition can be decided in different ways. GDPCA [11] takes the point nearest to the center of each voxel (Center base), while MoC [12] takes the average of features in each group (Avg base). The former suffers from the overhead of distance calculation, while the latter may lead to nearly symmetric offsets around zero. For performing CIM operations, the inputs are encoded in 2's complement format. Thus, near-zero negative values for offsets (e.g. -1, -2) in this format contain lots of "1" bits (yellow line in Fig. 3(e)), rapidly reducing the bit sparsity of offset data and increasing the energy cost of in-memory computing. To address this issue, the base point is defined as a vector of channel-wise minimum feature values across different data points within the same partition, ensuring that all offsets are positive (Fig.

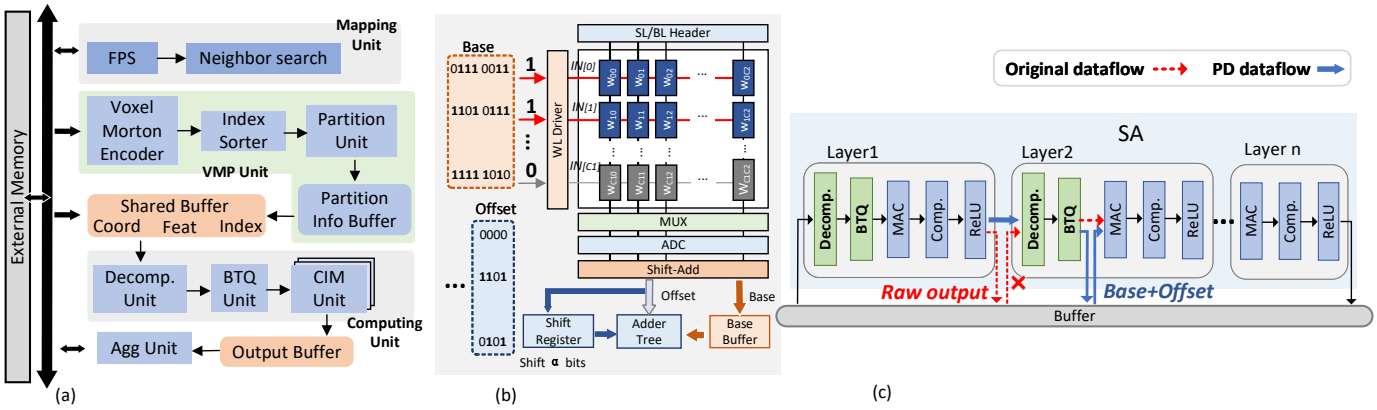


Fig. 5. (a) Architecture Overview; (b) hardware design for CIM computing unit; (c) Pre-decomposition dataflow

3(d)). Since all vectors within one partition exhibit strong similarity in value distributions along the channel axis, there is no need to worry about the base point being a zero vector. The performance of the proposed partitioning and base point selection methods are evaluated by analyzing the Cumulative Distribution Function (CDF) of EBN for bases and offsets in Fig. 3(e). It can be observed that the offsets using CM base point exhibit much higher bit sparsity, compared to raw data, as well as the offsets using the Avg and Center base, which will improve the energy efficiency of CIM computing unit.

B. Bit-level Truncation Quantization

While EBN impacts the energy efficiency of CIM operations, EBL decides the total computing cycles. As shown in Fig. 4(a), the EBL of raw data primarily ranges from 7 to 8 bits (red line) for SA1, which means 7 to 8 computing cycles are needed for bit-serial inputs. On the other hand, the EBL of offset data is considerably less, typically ranging from 4 to 6 bits. Based on this observation and DNN's tolerance for small noise, we introduce the Bit-level Truncation Quantization (BTQ). We truncate the high and low redundant bits of the input data in a real-time fashion. As shown in Fig. 4(b), for a complete 8-bit input offset from decomposition, we only keep the $\alpha_{th} \sim \beta_{th}$ bits for further computation, where α and β are configurable by the CIM macro.

IV. Hardware Design

A. Architecture Overview

As shown in Fig. 5(a), Point-CIM mainly consists of five parts: Mapping Unit, Voxel-Morton Partitioning (VMP) Unit, Shared Input Buffer, Computing Unit, and Aggregation (Agg) Unit. The Mapping Unit supports the FPS and Neighbor search operations for selecting centroid points and finding neighboring points, respectively. In the VMP unit, the Voxel-Morton Encoder interleaves only the 3 MSBs of binary representations of the three coordinates values for each point, dividing the whole space into $8 \times 8 \times 8$ voxels. The Index Sorter will sort all voxel indexes. Then, the Partition Unit will gather voxels according to two principles to generate final partitions. The shared buffer stores the index info, feature data, 3D coordinates, as well as morton-encoded indices. The main computations of feature learning are performed at the Computing Unit, which contains CIM macros for MAC operations and specific digital units for decomposition and BTQ. A global Agg Unit is employed to generate the final output from neighboring information and all points' features. The dataflow for feature learning on-chip will be discussed later.

B. Reconfigurable CIM Computing Units

The CIM macro (Fig. 5(b)) of Point-CIM can operate in two modes: Base Mode and Offset Mode. In the base mode, high-precision inputs are fed to CIM macro to generate the base partial sums. These partial sums are sent to the base buffer for temporary storage. In the offset mode, the offset inputs with a shorter bit-width are applied to the same CIM macro, allowing them to be processed with fewer computing cycles. The precision of inputs is configured by controlling the number of cycles for CIM operation. The shift-add units inside the CIM macro are designed to support the input bit-width of the base. Due to the BTQ, the scales of base and inputs are different for the same cycles. Thus, to recover the MAC results between the weights and original inputs, the offset partial sums need to be aligned with the base partial sums. This is done by a configurable shift register, where the base partial sums are shifted right by α bits (if the $\alpha - 1$ LSBs were truncated). The adder tree then combines the data from the base buffer and the shift register to produce the final output.

In our design, the CIM array is based on SRAM and is organized by tiles. Each tile has a memory size of 1024 × 1024, with each memory cell storing a single bit data. Each tile is further divided into PEs with size of 2×2, and each PE is subdivided into subarrays with size of 4×4. Each subarray has a memory size of 128×128 and is capable of conducting basic MAC operations. The 6-bit SAR-ADC is employed at array level.

C. Pre-decomposition Dataflow

For the layer-by-layer operations of the feature learning stage, the intermediate features generated after the activation

function are sent back to the on-chip buffer for temporary storage. As the chip size scales up, this data movement could lead to significant hardware overhead. To mitigate this, we propose a Pre-decomposition (PD) Dataflow, which shifts the decomposition and BTQ operation from the input of a certain layer to the output of the previous layer. As shown in Fig. 5(c), the network for feature learning consists of multiple layers of perceptrons, with the last layer followed by aggregation and max-pooling operations. In the original dataflow, the decomposition and BTQ are applied on the inputs of each layer and then fed into the MAC units for further processing. The raw outputs of each layer are sent back to the buffer for temporal storage and used as the inputs of the next layer later. In this dataflow, the raw data transmitted are in full precision. In the PD dataflow, the intermediate data between perceptron layers inside the same block are decomposed and BTQed before being sent back to the buffer. In this way, the bases are transferred in full precision and offsets in reduced precision, enabling reducing total transmitted bits. Consequently, the overhead in data transfer between buffer and CIM array can be mitigated significantly. For the last layer in the SA, as the raw data has to be applied to the Agg unit and Maxpool unit, the decomposition and BTQ for the next layer could not be applied in advance.

V. Evaluation

A. Experimental Setup

Algorithmic Setup. To evaluate our design, we select the representative Point-based model, PointNet++, and assess its performance across three datasets, as shown in Table I. All models are implemented using PyTorch and trained to reach state-of-the-art performance. Subsequently, the models were retrained using the delayed-aggregation method [9]. Finally, all models were quantized to 8-bit weights and 8-bit activations. The accuracy performance of all models is compared in Table II.

Hardware Setup. For the feature learning stage, the hardware performance for MLP-related computations is evaluated using the Neurosim framework [16], [21]. NeuroSim is an open-source simulator that can build the CIM architecture with traced inputs from real networks to evaluate the overall hardware performance. Other components, such as the Mapping Unit and Agg Unit, are directly adopted from GDPCA [11].

Baseline. To evaluate the accuracy impact and hardware performance improvement of Point-CIM, we choose the 8-bit quantized model as the accuracy baseline. Additionally, we adopt three types of platforms as hardware baselines, including CPU (Intel Xeon Gold 6226R), GPU (Tesla A100), and state-of-the-art ASICs (Mesorasi [9], PointAcc [22], GDPCA [11]).

B. Parameters Setting

The hardware design parameters are presented in Table III. Point-CIM is compared with several prior customized accelerators, which are all based on 40 nm technology node and operate at a frequency of 1 GHz. Point-CIM is implemented

TABLE I
EVALUATION BENCHMARKS

Application	Dataset	Model	Notation
Classification	ModelNet40 [23]	PointNet++ (SSG)	PointNet++(m)
Part Segmentation	ShapeNet [24]		PointNet++(ps)
Segmentation	S3DIS [20]		PointNet++(s)

TABLE II
Model Accuracy (%)

Dataset	Original Model	Delayed-Aggregation	w:8,a:8
ModelNet40	92.2	91.3	91.3
ShapeNet	85.2	84.6	84.4
S3DIS	54.3	54.6	53.7

*w:8 and a:8 mean the weight and activation are quantized with 8-bit precision.

TABLE III
Hardware Configurations

Chip	Mesorasi [9]	PointAcc [22]	GDPCA [11]	Point-CIM
Tech(nm)	40	40	40	45
Frequency(MHz)	1000	1000	1000	1000
Buffer(KB)	2048	776	2560	2048
Bandwidth (GB/s)	HBM	HBM	HBM	HBM
MACs	64×64 Systolic Array	64×64 Systolic Array	64×64 BSP Array	SRAM:4.4MB CIM Array

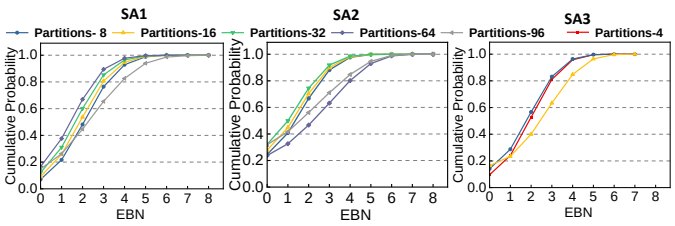


Fig. 6. CDF of EBN for offsets in SA1, SA2, and SA3 of PointNet++(m).

with 45 nm technode and operates at 1 GHz. Mesorasi and PointAcc utilize 64×64 systolic array MACs, while GDPCA introduces 64x64 BSP array MACs. Point-CIM employs an SRAM-based CIM architecture. The computational capability of the CIM architecture is evaluated based on the capacity of the SRAM, which should be able to store all weight parameters of models. In our experiment, we configured the CIM-SRAM with a capacity of 4.4 MB. All architectures utilize HBM memory with a bandwidth of 256 GB/s.

The number of partitions is a critical parameter that needs to be specified before experiments. In this section, we present the parameter setting for PointNet++(m) as an example. The same analytical process can be applied to other benchmarks to determine the optimal number of partitions. As shown in Fig. 6, when the number of partitions for SA1 is set to 64, EBN achieves the sparsest distribution. The optimal number of partitions for SA2 is 32; while for SA3, it is 8. These parameter settings align with the characteristics of PointNet++. As the depth increases, the number of points in each layer decreases, requiring fewe rpartitions.

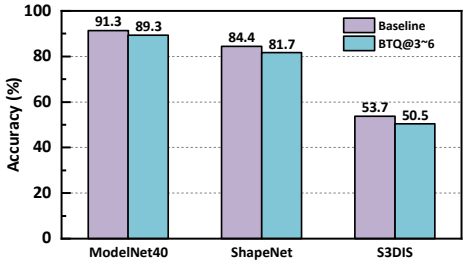


Fig. 7. The accuracy comparison between baseline models and models quantized with BTQ.

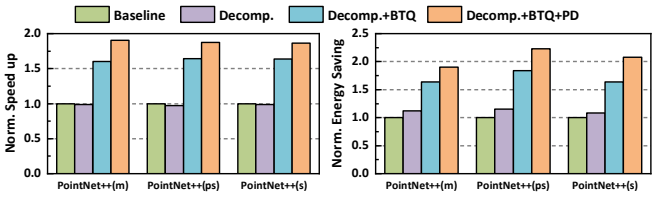


Fig. 8. The speedup and energy savings achieved by BTQ and PD dataflow are compared to the baseline model.

The configurable parameters α and β are set to be 3 and 6 according to case study, which means the 3rd to 6th bits of offsets are reserved for CIM operations.

C. Accuracy Analysis

To analyze the the impact of BTQ on inference accuracy of models, we compare the the accuracy between models quantized with BTQ and the baseline across three different datasets. As shown in Fig. 7, after BTQ, the accuracies decrease by 2.0%, 2.7%, and 3.2% across three datasets, respectively. The decreases in accuracy are within an acceptable range for our experiments.

D. Hardware Performance Analysis

Fig. 8 compares the speedup and energy savings in feature learning stage optimized with BTQ and PD dataflow against the baseline across three different models. After decomposition, Point-CIM achieves increases in energy efficiency by $1.12\times$, $1.15\times$, and $1.08\times$, respectively. This indicates that the offset values show much higher bit sparsity after decomposition and the CIM-based computing unit can handle inputs with high bit sparsity with higher energy efficiency. Furthermore, after applying BTQ, Point-CIM achieves improvements in energy efficiency by $1.64\times$, $1.84\times$, and $1.64\times$, respectively. Meanwhile, Point-CIM demonstrates speed-up enhancement of $1.61\times$, $1.64\times$, and $1.64\times$, respectively. At last, when both BTQ and PD dataflow are applied, energy savings increase by factors of $1.90\times$, $2.23\times$, and $2.07\times$, while speed-up is improved by $1.91\times$, $1.87\times$, and $1.86\times$, respectively. It suggests the effectiveness of PD dataflow in mitigating the transfer overhead of intermediate features.

Fig. 9 provides a comparative analysis of the normalized speedup and energy efficiency of Point-CIM relative to other hardware baseline on PointNet++(m). Specifically, Point-CIM

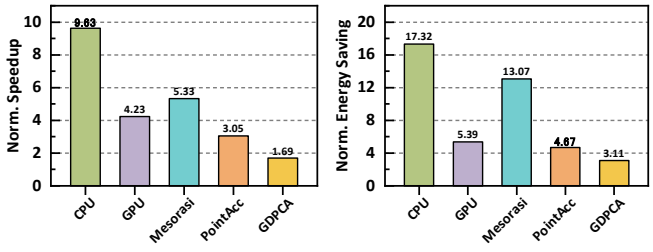


Fig. 9. The speedup and energy saving of Point-CIM over other platforms

achieves an average speedup of $9.63\times$, $4.23\times$, $5.33\times$, $3.05\times$, and $1.69\times$ over CPU, GPU, Mesorasi, PointAcc, and GDPCA, respectively. Moreover, Point-CIM shows notable improvements in energy efficiency, offering $17.32\times$, $5.39\times$, $13.07\times$, $4.67\times$, and $3.11\times$ higher energy savings compared to other platforms. These results highlight Point-CIM's superior performance in both computational speed and energy efficiency when compared to other hardware platforms.

VI. Conclusion

This paper presents Point-CIM, an efficient CIM-based accelerator for PNNs, with software and hardware co-optimization. A reconfigurable CIM unit is designed to efficiently process MAC operations on decomposed feature data with high bit sparsity. A novel VMP method combined with base point selection method is proposed to further improve the offset bit sparsity and reduce the hardware overhead. Additionally, a BTQ method is proposed to directly compress the bit-width of the offset without incurring any additional hardware overhead. To reduce data transfers of intermediate features between the on-chip buffer and CIM array, a PD data movement strategy is employed to optimize dataflow. Extensive experiments show that Point-CIM achieves an average speedup of $1.69\times$ to $9.63\times$, and $3.11\times$ to $17.32\times$ improvement in energy efficiency, compared to state-of-the-art accelerators and general-purpose processors.

VII. Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (Grant No. 62304195), Guangzhou-HKUST(GZ) Joint Funding Program (No. 2024A03J0540), Guangzhou-HKUST(GZ) Joint Funding Program (No. 2024A03J0541), Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project, Grant No.2023A03J0013)

References

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[3] A. V. Phan, M. Le Nguyen, Y. L. H. Nguyen, and L. T. Bui, "Dgcnn: A convolutional neural network over large-scale labeled graphs," *Neural Networks*, vol. 108, pp. 533–543, 2018.

[4] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 16 259–16 268.

[5] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 918–927.

[6] M. Han, L. Wang, L. Xiao, H. Zhang, C. Zhang, X. Xu, and J. Zhu, "Quickfps: Architecture and algorithm co-design for farthest point sampling in large-scale point clouds," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4011–4024, 2023.

[7] X. Yang, T. Fu, G. Dai, S. Zeng, K. Zhong, K. Hong, and Y. Wang, "An efficient accelerator for point-based and voxel-based point cloud neural networks," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[8] Y. Lian, X. Yang, K. Hong, Y. Wang, G. Dai, and N. Xu, "A point transformer accelerator with fine-grained pipelines and distribution-aware dynamic fps," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[9] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1037–1050.

[10] Z. Ying, S. Zhao, S. Bhuyan, C. S. Mishra, M. T. Kandemir, and C. R. Das, "Pushing point cloud compression to the edge," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 282–299.

[11] C. Chen, X. Zou, H. Shao, Y. Li, and K. Li, "Point cloud acceleration by exploiting geometric similarity," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1135–1147.

[12] X. Liu, Z. Song, H. Chen, X. Li, and X. Liang, "Moc: A morton-code-based fine-grained quantization for accelerating point cloud neural networks," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[13] D. Franklin, "Nvidia jetson tx2 delivers twice the intelligence to the edge," *NVIDIA Accelerated Computing—Parallel Forall*, vol. 6, 2017.

[14] J.-s. Seo, J. Saikia, J. Meng, W. He, H.-s. Suh, Y. Liao, A. Hasssan, I. Yeo *et al.*, "Digital versus analog artificial intelligence accelerators: Advances, trends, and emerging designs," *IEEE Solid-State Circuits Magazine*, vol. 14, no. 3, pp. 65–79, 2022.

[15] S. Huang, H. Jiang, and S. Yu, "Hardware-aware quantization/mapping strategies for compute-in-memory accelerators," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 3, pp. 1–23, 2023.

[16] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2306–2319, 2020.

[17] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2017, pp. 541–552.

[18] T. Yang, D. Li, Y. Han, Y. Zhao, F. Liu, X. Liang, Z. He, and L. Jiang, "Pimgcn: A reram-based pim design for graph convolutional network acceleration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 583–588.

[19] W. Li, X. Sun, S. Huang, H. Jiang, and S. Yu, "A 40-nm mlc-rram compute-in-memory macro with sparsity control, on-chip write-verify, and temperature-independent adc references," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 9, pp. 2868–2877, 2022.

[20] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1534–1543.

[21] C. Wang, Z. Chen, and S. Huang, "Micsim: A modular simulator for mixed-signal compute-in-memory based ai accelerator," *arXiv preprint arXiv:2409.14838*, 2024.

[22] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, "Pointacc: Efficient point cloud accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 449–461.

[23] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[24] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

HIVE: A High-Priority Victim Cache for Accelerating GPU Memory Accesses

Yuhan Tang, Jianmin Zhang*, Sheng Ma*, Tiejun Li*, Hanqing Li, Shengbai Luo, Jixuan Tang, Lizhou Wu
National University of Defense Technology, Changsha, China
{jmzhang, masheng, tjli}@nudt.edu.cn

***Abstract*—The victim cache was originally designed as a secondary cache to handle misses in the L1 data (L1D) cache in CPUs. However, this design is often sub-optimal for GPUs. Accessing the high-latency L1D cache and its victim cache can lead to significant latency overhead, severely degrading the performance of certain applications. We introduce *HIVE*, a high-priority victim cache designed to accelerate GPU memory accesses. HIVE handles memory requests first, before they reach the L1D cache. Our experimental results show that HIVE achieves an average performance improvement of 77.1% and 21.7% compared to the baseline and the state-of-the-art architecture, respectively.**

I. Introduction

Due to their powerful computing capacity and high energy efficiency, Graphics Processing Units (GPUs) are widely used in various applications, such as high-performance computing, machine learning and embedded systems. GPUs accelerate these applications by concurrently executing many warps, each consisting of a group of threads that execute the same instructions. For example, NVIDIA GPUs have supported up to 64 concurrent warps (2048 concurrent threads) within a single core since the Kepler architecture [1].

Unfortunately, a large number of warps can lead to severe performance degradation when GPUs execute memory-intensive applications [2]–[5]. One important reason for the performance degradation in these applications is that multiple warps sharing limited cache resources can cause severe cache contention [6], [7]. Although recent GPU architectures have larger L1D caches than previous generations, a single thread typically can access only tens of bytes in the L1D cache [8], [9].

Previous works have proposed using a victim cache to alleviate cache contention and achieve significant performance improvements in GPUs [7], [10]. In their designs, the function of the victim cache remains the same as in CPUs, acting as a secondary cache when a miss occurs in the L1D cache. However, we find that these designs are often sub-optimal for GPUs. This is due to two main reasons.

On the one hand, the L1D cache in GPUs exhibits higher miss rates, while evicted lines from the L1D cache have a high probability of being reused. This means that the number of memory requests that hit in the victim cache can approach or even exceed the number of requests that hit in the L1D cache. For example, we observe an average L1D cache miss rate of 63.7% in our evaluated GPU applications, and Oh et al. [7] also observe similar results. We also observe that most evaluated GPU applications exhibit a high probability (an average of 69.6%) of reusing evicted L1D cache lines (see Section II-B for more details). Therefore, the victim cache does not need to act as a supplement to the L1D cache.

On the other hand, GPUs use a unified architecture for the L1D cache and shared memory, or the L1D cache and texture cache, to save hardware overhead [1], [8], [9], [11]. However, this hardware architecture significantly increases the L1D hit latency. For example, the L1D hit latency is 32 cycles in the TU104 GPU and 82 cycles in the GP100 GPU [12]. We observe that some applications are susceptible to the cache access latency, and increasing cache latency can severely degrade performance (see Section II-C for more details). However, the conventional victim cache is a secondary cache for the L1D cache. This means that the victim cache has an even higher hit latency, which severely degrades the performance of latency-sensitive applications.

To this end, we propose ***HIVE***, a high-priority victim cache for accelerating GPU memory accesses. The key idea of HIVE is to prioritize using the victim cache rather than the L1D cache to handle memory requests, which means that the victim cache does not act as a supplement to the L1D cache. Doing so can improve performance because HIVE enables most memory requests to be fulfilled in the victim cache.

To implement HIVE, we insert an extra stage into the baseline pipeline, dedicated to handling memory accesses to the victim cache. This stage involves two hardware structures: the access controller and the victim cache tag array. The access controller is composed of set indexing logic and memory request queues: (1) the set indexing logic is responsible for calculating the victim cache set number based on the memory request address, and (2) the request queues buffer requests to access the victim cache, thereby avoiding the blocking of running warps from generating memory requests due to the limited number of requests that the tag array can handle simultaneously. The victim cache tag array is a traditional set-associative structure that stores tag information associated with the data array. To save hardware overhead, we use the method from the state-of-the-art architecture, Linebacker [7], reusing idle register files as the victim cache data array.

To improve the effectiveness of HIVE, we explore two optimization techniques on top of our basic design. First, we increase the effective capacity of the victim cache by employing a low-latency data compression technique. We find that a slight increase in hit latency in the high-priority victim cache due to data compression has almost no impact on performance. Second, We design a new replacement policy for the victim cache based on the LRU (Least Recently Used) algorithm, which minimizes the number of valid cache lines evicted.

To evaluate the effectiveness of HIVE, we measure the performance of various applications from different benchmark suites using the PTX mode in the AccelSim simulator [13]. Our experimental results show that HIVE achieves an average of 77.1% and 21.7% performance improvement compared to the baseline and the state-of-the-art architectures, respectively, with maximum improvements of 240.0% and 86.5%. As a trade-off, HIVE consumes 9.4% more power on average compared to the state-of-the-art architecture. However, HIVE can save an average of 17.4% in power consumption compared to the baseline architecture.

We make the following contributions:

- We propose HIVE, the first high-priority victim cache for accelerating GPU memory accesses. HIVE handles memory

*Corresponding authors.

requests first, before they reach the L1D cache. We have open-sourced our design code[1].

- We introduce two optimizations for HIVE: (1) increasing the effective capacity of the victim cache by using a low-latency data compression technique; (2) designing a cache replacement policy to minimize the number of valid cache lines evicted.
- We evaluate HIVE and show that it significantly improves performance and power efficiency, effectively outperforming the state-of-the-art across different GPU architectures.

II. BACKGROUND AND MOTIVATION

A. Baseline GPU Architecture

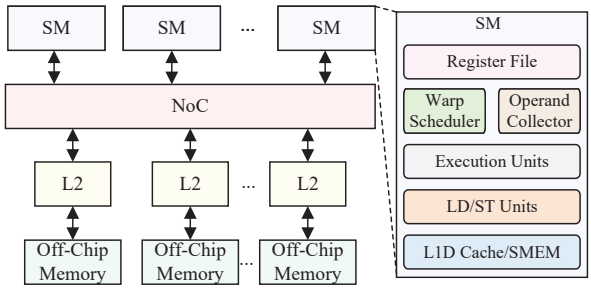


Fig. 1. Baseline GPU high-level organization.

GPU programming models allow the creation of thousands of threads that each execute the same code. NVIDIA's programming model groups threads into 32-element vectors called warps to improve efficiency. The threads in each warp execute in a SIMT (single instruction, multiple thread) fashion. In most programs, many warps map to a single GPU core, or streaming multiprocessor (SM) in NVIDIA's terminology [14].

Fig. 1 illustrates the high-level architecture of a baseline GPU. A GPU is composed of multiple SMs, in terms of the memory hierarchy, each of which contains a private register file, and a private, dynamically partitioned L1D cache and shared memory.

GPUs have very large register files to support a larger number of threads. Each active warp stores its architectural context in its own dedicated register file space. For example, NVIDIA GPUs have 256 kilobytes (KB) of register files per core across multiple generations [1], [8], [9], [11], [15]. However, this large register file is often underutilized [5], [7].

The L1D cache and shared memory (SMEM) share a unified on-chip memory space. The programmer specifies the capacity of software-managed shared memory, and the rest of the unified on-chip memory space is dedicated to the L1D cache, automatically [16]. Although recent GPU architectures have larger L1D caches than previous generations, a single thread typically can access only tens of bytes in the L1D cache [8], [9].

The GPU core accesses the memory-side shared L2 cache via the NoC in the case of an L1D cache miss. The L2 cache connects to off-chip memory, which is either HBM or GDDR [4]. Memory requests are expected to hit in the L1D cache to avoid the much longer latency of the L2 cache or off-chip memory accesses.

B. Capturing Lost Locality with Victim Cache

GPUs frequently evict cache lines that will be reused due to severe cache contention. To quantify the benefits of using a victim cache, we measure the L1D miss rates and the probability of evicted cache lines from the L1D cache being reused in various applications (details of simulations and applications can be found in Section IV).

[1]https://github.com/paper-account1/HIVE

Table I. The L1D miss rates (*Miss.*) and the probability (*Prob.*) of evicted cache lines being reused.

App.	Miss.(%)	Prob.(%)	App.	Miss.(%)	Prob.(%)
KM	99.7	75.3	FW	48.8	72.2
GM	6.3	54.1	MV	91.7	70.6
GMV	79.1	85.2	BC	8.6	77.6
BG	79.2	67.8	SN	63.2	69.4
MVT	79.1	64.4	BFS	80.9	59.3

Table I shows our results. We make two key observations. First, all evaluated applications exhibit a high probability (54.1% - 85.2%) of reusing evicted cache lines, indicating that a victim cache can be effectively used to capture lost data locality. Second, the L1D cache of most applications exhibits high miss rates (48.8% - 99.7%), except for GM and BC, indicating that most memory requests cannot be fulfilled in the L1D cache. We conclude that if a victim cache is used, the number of memory requests that hit in the victim cache can approach or even exceed the number of requests that hit in the L1D cache for most applications.

C. Motivation for a High-Priority Victim Cache

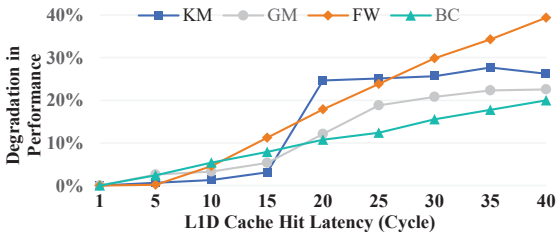


Fig. 2. Sensitivity of latency-sensitive applications to the L1D cache hit latency.

The increase in cache hit latency can significantly hurt the performance of some applications. We find that four out of ten evaluated applications experience a performance drop of over 20% when the L1D hit latency increases from 1 cycle to 40 cycles. We define these applications as latency-sensitive applications. Fig. 2 shows the sensitivity of latency-sensitive applications to the L1D cache hit latency. When the L1D cache latency increases to 40 cycles, the average performance drop of latency-sensitive applications is 27.0%.

The conventional victim cache was a secondary cache when a miss occurs in the L1D cache, which means that the hit latency of the victim cache includes the hit latency of the L1D cache. Even if memory requests hit in the victim cache, this latency overhead can severely degrade the performance of latency-sensitive applications.

Based on the above observations, it is natural to prioritize using a victim cache that does not act as a secondary cache to the L1D cache to handle memory requests. We refer to this victim cache as a high-priority victim cache. By doing so, most memory requests can be fulfilled in the victim cache, thereby accelerating overall memory accesses in GPUs.

III. DESIGN

In this section, we introduce HIVE, a high-priority victim cache designed to accelerate GPU memory accesses. HIVE handles memory requests first, before they reach the L1D cache.

A. Overall Architecture of HIVE

Fig. 3 outlines the key structures of the HIVE design. The gray parts indicate the baseline architecture, and the rest indicates the additional or modified parts required to support HIVE.

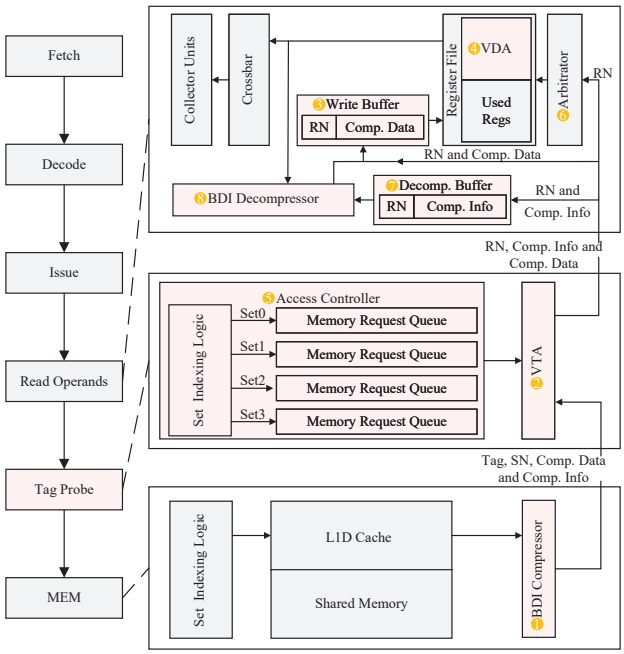


Fig. 3. Overall architecture of HIVE (*Comp. Info*: Compression Information, *Comp. Data*: Compressed Line, *RN*: Register Number, *SN*: Set Number).

To implement HIVE, we insert an extra stage, called the tag probe, into the baseline five-stage memory pipeline before the mem stage, dedicated to handling memory accesses to the victim cache. Compared to the baseline architecture, memory requests are generated in the tag probe stage, rather than in the mem stage, and these requests first arrive at the victim cache rather than the L1D cache.

The tag probe stage involves two hardware components: the Access Controller (❺) and the Victim Tag Array (VTA) (❷). Memory requests accessing the VTA first arrive at the access controller, where the set indexing logic calculates the set number based on the memory request address and then forwards the request to the corresponding memory request queue associated with the set number. A given request is dequeued as soon as the request's VTA set is ready to serve a new request. The VTA is a traditional set-associative structure that stores tag information associated with the data array. To save hardware overhead, HIVE uses the method from the state-of-the-art architecture, Linebacker [7], reusing idle register files as the victim cache data array (VDA) (❹). A warp register is 128 bytes, which is equal to the size of a physical cache line.

To improve the effectiveness of HIVE, we explore two optimization techniques: (1) cache lines evicted from the L1D cache are first compressed using the Base-Delta-Immediate (BDI) [17] Compressor (❶) and then stored in the victim cache, thus increasing the effective capacity of the victim cache; (2) we design a new replacement policy for the VTA based on the LRU algorithm, which minimizes the number of valid cache lines evicted.

B. Details of HIVE Implementation

BDI Compressor and Decompressor. HIVE exploits the BDI data compression technique [17] to increase the effective capacity of the victim cache. This is possible because a low-latency victim cache accelerates most memory accesses and leaves a "latency margin". As shown in Fig. 2, we observe that all latency-sensitive applications exhibit slight performance drops (up to 5.91%) when the increase in the hit latency is within 10 cycles, and only show significant performance drops when the hit latency exceeds 15 cycles. Therefore, we exploit a low-latency, high-compression-rate BDI algorithm from various compression algorithms to utilize this latency margin and increase the effective capacity of the victim cache.

The key idea of the BDI compression is that, for many cache lines, the values within a cache line have a low dynamic range - i.e., the differences between values stored within a cache line are small. The BDI compression algorithm chooses an 8-, 4-, or 2-byte *base*, divides the cache line into blocks of size equal to *base*, and represents each block as a *delta* (Δ) which is the difference of the value of the block from the *base* [18]. The compression logic consists of eight distinct compressor units: six units for different base sizes (8, 4 and 2 bytes) and Δ sizes (4, 2 and 1 bytes), and two units for zero and repeated value compression. Every compressor unit takes a 128-byte cache line as an input, and outputs whether or not this cache line can be compressed with this unit. If it can be, the unit outputs the compressed cache line. If multiple compression options are available for the cache line (e.g., 8-byte base 1-byte Δ and zero compression), the selection logic chooses the one with the smallest compressed cache line size. Note that all potential compressed sizes are known statically and described in Table II. All compressor units can operate in parallel.

Table II. BDI compression encoding. All sizes are in bytes.

Name	Base	Δ	Size	Enc.	Name	Base	Δ	Size	Enc.
Zeros	1	0	1	0000	Rep.Values	8	0	8	0001
Base8-Δ1	8	1	24	0010	Base8-Δ2	8	2	40	0011
Base8-Δ4	8	4	72	0100	Base4-Δ1	4	1	36	0101
Base4-Δ2	4	2	68	0110	Base2-Δ1	2	1	66	0111
NoCompr.	N/A	N/A	128	1111					

Based on prior works [18], [19], we model a 2-cycle compression latency and a 2-cycle decompression latency, with 0.192 nJ compressor access energy and 0.056 nJ decompressor access energy for BDI. For simplicity, we allow a warp register to store at most two compressed cache lines. Each compressed cache line requires 4 bits to store compression encoding and 6 bits to store the start segment, amounting to 10 bits of compression information.

VTA. The VTA is a classic set associative array with the same number of sets as the L1D cache. Each entry of the VTA maps one-to-one to a 128-byte warp register of the VDA. The baseline architecture has 2048 warp registers in total. If a tag hits in an entry at set A and way B, the warp register number (RN) associated with the compressed cache line is calculated as follows:

$$RN = OFFSET + A \times (\#WAYS) + B \quad (1)$$

OFFSET in the above equation is the value that is manually set. In this paper, we set OFFSET to 1024, allowing a maximum of 1024 idle warp registers to be reused as the VDA (i.e., RN 1024-2048). The VTA also includes a simple logic that calculates RN as shown in Equation 1.

Fig. 4 shows the entry layout in the VTA. Each entry contains two sub-entries because we allow each warp register to store at most two compressed cache lines. Each sub-entry consists of fields including the current request's tag, compression info (indicating how to decompress the compressed cache line), a valid bit (indicating if

the compressed cache line is valid), and LRU bits. Additionally, each entry has an active bit, which is set if the corresponding warp register is idle (determined at compile time).

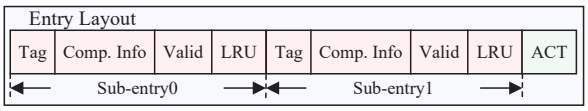


Fig. 4. VTA entry layout (*Comp. Info*: Compression Information, *ACT*: Active Bit).

We model the VTA using CACTI 6.5 [20] with a 22nm process. We observe that for a 256-way set-associative tag array with 4 sets, 1.877 ns (3 cycles) access latency and 0.041 nJ access energy are necessary for the configurations listed in Table III.

Replacement Policy. To reduce the number of valid compressed lines evicted from the victim cache, the VTA exploits a new cache replacement policy for the victim cache, as shown in Algorithm 1.

In the VTA, two sub-entries in a single entry map one-to-one to two compressed lines in a warp register. A compressed line is valid if the corresponding sub-entry in the VTA has its valid bit set. For any evicted line from the L1D cache to the victim cache, Algorithm 1 first avoids replacing an invalid sub-entry which could lead to the eviction of adjacent valid sub-entry.

For example, assume that a VTA entry stores an invalid sub-entry A and a valid sub-entry B. The sub-entry B is mapped to a compressed line of size 72 bytes. If a compressed cache line is evicted from the L1D cache and its size is 72 bytes, replacing the invalid sub-entry A will cause the valid sub-entry B to be evicted because the combined size of the two compressed cache lines will exceed 128 bytes. Algorithm 1 avoids this replacement.

Subsequently, Algorithm 1 attempts to find an entry with two invalid sub-entries and replaces one of them to store the information of the evicted compressed cache line. If no such entry is found, it finally uses the baseline LRU replacement policy.

Access Controller. The access controller contains two functional modules: the set indexing logic and the memory request queues. A memory request first arrives at the set indexing logic. To guarantee cache coherence, the VTA and the L1D cache use the same set indexing logic. The set indexing logic calculates the set number of the VTA for the memory request and forwards the request to the corresponding memory request queue.

The memory request queues buffer any request to the VTA, thereby avoiding the blocking of running warps from generating memory requests due to the limited number of requests that the tag array can handle simultaneously. A given request is dequeued as soon as the request's VTA set is ready to serve a new request. The number of memory request queues is equal to the number of sets in the VTA, as each set has a separate memory request queue. Similar to the L1D cache, the VTA can handle multiple requests per cycle from different memory request queues simultaneously.

Buffers. The write buffer for the register file (Write Buffer) (❸) holds compressed data waiting to be written into the register file, and the decompression buffer (Decomp. Buffer) (❼) holds the compression information needed for decompressing the compressed data. We evaluate the lengths of the write buffer and the decompression buffer from 1 to 20. We observe that a 1-entry write buffer and a 4-entry decompression buffer are sufficient to saturate performance.

Cache Coherence. The victim cache stores only the data evicted from the L1D cache, and the data evicted from the victim cache is directly discarded. When a store instruction is executed, HIVE searches the requested line in the victim cache. If the written line hits in the victim cache, the victim cache entry is invalidated, and the store data is sent directly to the L1D cache. The invalidated entry can be reused as a victim line later. When a cache line is evicted from the L1D cache, the evicted line is sent to both the victim cache and the next level of the memory hierarchy. The result of this cache policy is that the data in the victim cache is never dirty, and the L1D cache and the victim cache do not store data from the same memory address simultaneously.

Algorithm 1: VTA Replacement Policy

Input : *tag* denotes the tag of evicted cache line
set denotes the set number of evicted cache line
sz denotes the compressed size of evicted cache line
Output: *pos* denotes the position of the sub-entry to be replaced
way denotes the way of the sub-entry to be replaced

```
// i_0 denotes sub-entry0, and i_1 denotes sub-entry1
for i ∈ VTA[set] do
    if Valid(i_0) = false then
        if Valid(i_1) and (sz + Size(i_1) > 128B) then
            continue
        else
            return pos ← Pos(i_0), way ← Way(i)
    else if Valid(i_1) = false then
        if Valid(i_0) and (sz + Size(i_0) > 128B) then
            continue
        else
            return pos ← Pos(i_1), way ← Way(i)
if find_invalid (tag, set, sz) then
    pos, way ←invalid_replace (tag, set, sz)
else
    pos, way ←lru_replace (tag, set, sz)
return pos, way
```

C. HIVE Workflow

An evicted cache line is sent to the BDI Compressor (❶) for data compression. The compressor generates compression information during the compression, which is used to guide the decompressor to decompress the compressed data correctly. After compression, tag (*Tag*), set number (*SN*), compressed cache line (*Comp. Data*), and compression information (*Comp. Info*) are sent to the VTA (❷). The VTA stores tag and compression information for tag probing and decompression, respectively. Then, the VTA calculates the register number associated with the compressed cache line. The register number (*RN*) and the compressed cache line are sent to the Write Buffer (❸). Each buffer entry consists of a register number field and a compressed data field. When the port of the register file is free, the compressed cache line is written into the corresponding 128-byte warp register in the VDA (❹). The VDA is a set of statically unused warp registers whose numbers are determined after compile time.

A memory request from the core first arrives at the Access Controller (❺). The controller forwards the memory request to access the VTA (❷). Upon a tag miss, the request is sent to the L1D cache of the MEM stage. Upon a tag hit, if the request is a store, the store data is sent to the L1D cache, and the corresponding sub-entry in the VTA is cleared. Upon a tag hit, if the memory request is a load, the register number is sent to the Arbitrator (❻) for reading the compressed data, and the register number and the compression information are sent to the Decomp. Buffer (❼) for decompressing the compressed data. Each buffer entry consists of a register number field and a compression information field. When the compressed data from

the VDA is sent to the BDI Decompressor (❽), the decompression buffer forwards compression information to the BDI decompressor to indicate how to decompress the data correctly.

IV. Experimental Setup

We model and evaluate HIVE using AccelSim [13], a cycle-level simulator. AccelSim has two execution modes: PTX and SASS. To obtain the necessary cache data, we use the PTX mode in the AccelSim simulator. Table III shows the simulation parameters modeling our baseline. This Turing-like GPU configuration is similar to that used in a recent study [4]. We select 10 applications from six benchmark suites: Rodinia-3.1 [21], ISPASS [22], Polybench [23], Tango [24], Microbenchmark [13], and Pannotia [25]. The applications used in our study are shown in Table IV.

Table III. Baseline GPU configuration.

Parameter	Value
GPU Core Features	30 SIMT cores, 1.525GHz
Max Warps/SM	64 (2048 threads)
Warp Scheduling	4 GTO schedulers/SM
Register Files/SM	256KB, 16 banks
L1 Data Cache/SM	32/64KB 64/128-way L1 data cache, 4 banks, LRU, 128B cache line, access latency=38 cycles
L2 Cache	96KB/memory sub partition (6MB in total), access latency=198 cycles
Off-chip DRAM	RRD=3, RCD=12, RAS=28, RP=12, RC=40, CL=12, WR=10, access latency=298 cycles

Table IV. Evaluated applications.

App.	Abbr.	App.	Abbr.
kmeans [21]	KM	fw_block [25]	FW
gemm [23]	GM	spmv [25]	MV
gesummv [23]	GMV	bc [25]	BC
bicg [23]	BG	sn [24]	SN
mvt [23]	MVT	bfs [22]	BFS

We study GPU performance for four different architectures: (1) *BL*: the baseline system that models a GPU architecture with the parameters reported in Table III. (2) *LB*: the state-of-the-art architecture, Linebacker [7]. It dynamically throttles warps and utilizes idle register space as victim cache space. (3) *HIVE-Basic*: HIVE without BDI compressor and decompressor and using the baseline LRU replacement policy. (4) *HIVE-All*: the full design of HIVE. To provide a fair comparison, we require that HIVE and LB use the same maximum number of idle registers. To analyze the power consumption of HIVE, we use a modified version of AccelWattch [26] that is augmented with the VTA and BDI compressor/decompressor power models.

V. Evaluation

A. Performance

To study the effectiveness of HIVE in improving GPU performance, we measure instructions per cycle (IPC) on four architectures, namely *BL*, *LB*, *HIVE-Basic*, and *HIVE-All* (see Section IV for details of each evaluated architecture). Fig. 5 shows the results. The y-axis shows the IPC of evaluated applications (the higher the better). All results are normalized to the baseline (*BL*).

We make four key observations. First, *HIVE-All* improves GPU performance by an average of 77.1% and 21.7% over *BL* and *LB*, respectively, for all applications. Remarkably, *HIVE-All* shows the greatest performance improvement in GMV (240.0% and 86.5%) because GMV has sufficient idle registers that can be reused as victim cache space, and the compression algorithm effectively compresses the data. Additionally, although GM and BC exhibit lower L1D miss rates, they are latency-sensitive (see Section II-B and II-C), so HIVE still improves their performance compared to *BL* and *LB*.

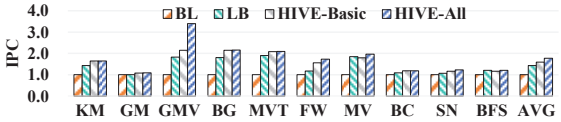


Fig. 5. Comparison of four GPU architectures' IPC (normalized to *BL*).

Second, *HIVE-Basic* improves performance by an average of 59.6% over *BL* by providing an extra on-chip cache and reducing access latency for most memory requests. Third, *HIVE-Basic* improves performance by an average of 11.2% over *LB* because it provides faster access to the victim cache. Fourth, *HIVE-All* improves performance by an average of 9.1% over *HIVE-Basic*, by utilizing data compression and exploiting a new replacement policy for the victim cache. We conclude that *HIVE-All* is highly effective at improving the performance of different applications.

B. Victim Cache Hit Rate

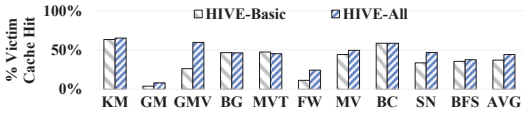


Fig. 6. Victim cache hit rates in *HIVE-Basic* and *HIVE-All*.

HIVE provides an extra cache to extend on-chip cache capacity, further reducing cache contention. We measure the victim cache hit rates for *HIVE-Basic* and *HIVE-All*. Fig. 6 shows our experimental results. We observe that the average hit rates for *HIVE-Basic* and *HIVE-All* are 37.1% and 44.2%, respectively, indicating that nearly half of the data evicted from the L1D cache can be reused in the victim cache. We also observe that *HIVE-All* improves hit rates by an average of 19.1% over *HIVE-Basic*, thanks to the BDI compression and the VTA replacement policy. We conclude that HIVE can effectively capture the lost data locality.

C. L2 Cache Traffic and Off-Chip Memory Traffic

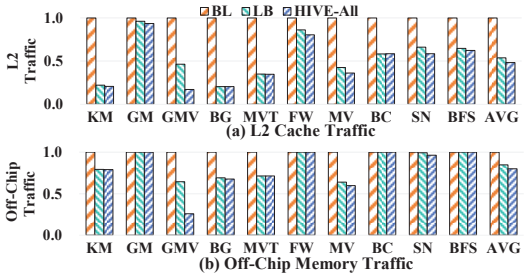


Fig. 7. L2 cache traffic and off-chip memory traffic in *BL*, *LB*, and *HIVE-All* (normalized to *BL*).

HIVE provides an extra victim cache and uses the BDI compression algorithm and the VTA replacement policy to increase the effective capacity of the victim cache, thereby effectively reducing access to the high-latency L2 cache and off-chip memory. We

compare the L2 cache traffic and the off-chip memory traffic of *HIVE-All* to those of *LB*. Fig. 7 shows our experimental results (normalized to the baseline (*BL*) architecture). *HIVE-All* reduces the L2 cache traffic and off-chip memory traffic by 51.7% and 20.0%, respectively, compared to the baseline. Compared to *LB*, *HIVE-All* reduces 12.0% more L2 cache traffic and 30.8% more off-chip memory traffic.

D. Overhead Analysis

We analyze the storage and power overheads of the additional hardware required by HIVE.

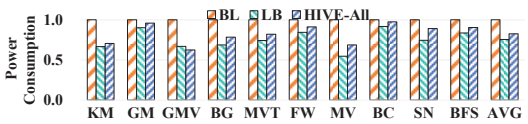


Fig. 8. Power consumption in *BL*, *LB* and *HIVE-All* (normalized to *BL*).

Storage Cost. HIVE has two main storage components: the VTA and the write buffer for the register file. The VTA is a 256-way set-associative tag array with 4 sets. Each entry in the VTA contains two sub-entries and one active bit. Each sub-entry contains 23-bit tag, 10-bit compression info, 1-bit valid, and 8-bit LRU. Consequently, the total size of the VTA is 10.625 KB. The write buffer has a 0.125 KB storage for compressed cache data. Overall, HIVE requires 10.75 KB of storage overhead, which accounts for 3.1% of the SM's storage.

Power Consumption. We measure the power consumption of *BL*, *LB*, and *HIVE-All*. Fig. 8 shows our experimental results (normalized to the baseline (*BL*) architecture). We observe that *HIVE-All* increases power consumption by an average of 9.4% compared to *LB*. However, *HIVE-All* significantly reduces power consumption by an average of 17.4% compared to the baseline. Both *HIVE-All* and *LB* reduce the L2 cache and the off-chip memory traffic compared to the baseline, but *HIVE-All* also increases register accesses and BDI compressor/decompressor accesses compared to *LB*.

Considering the experimental results in this section and the hardware cost estimation, we believe that HIVE is a cost-effective solution to improve performance on GPUs.

E. Sensitivity Study

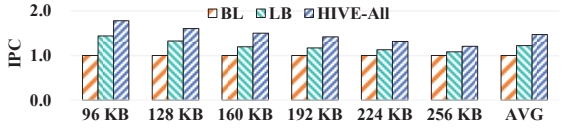


Fig. 9. Performance of varying unified memory space for the L1D cache and shared memory in *BL*, *LB*, and *HIVE-All* (all results are geometric mean values of the evaluated applications and normalized to *BL*).

Different GPU architectures have varying sizes of unified on-chip memory space for the L1D cache and shared memory [8], [15]. We explore the performance sensitivity by varying the unified on-chip memory space. We fixed the configurable shared memory capacity at 32/64 KB, with the exact size determined by the compiler. For example, if a kernel is configured with 32 KB of shared memory by the compiler in one architecture, the shared memory capacity for that kernel will still be configured as 32 KB by the compiler in other architectures. We measure the performance changes with the unified on-chip memory space for the L1D cache and shared memory set to 96 KB, 128 KB, 160 KB, 192 KB, 220 KB, and 256 KB.

Fig. 9 shows our results (normalized to the baseline (*BL*) architecture). We observe that *HIVE-All* improves performance by an average of 47.0% and 20.2% compared to the baseline and *LB*, respectively, across different unified memory space configurations. With the smallest capacity configuration (96 KB), *HIVE-All* shows 77.1% performance improvement while *LB* shows 43.6%. With the largest capacity configuration (256 KB), *HIVE-All* still outperforms *LB*. *HIVE-All* shows 20.8% performance improvement while *LB* shows 8.3%. We conclude that HIVE can be an efficient solution to improve cache efficiency across different GPU architectures.

VI. Related Work

The victim cache, proposed by [27], is placed in the refill path of an L1D cache and stores the cache lines evicted from the L1D cache in CPUs. Subsequently, there have been various studies based on the victim cache [7], [28], [29]. Differently, our proposed HIVE is more suitable for GPUs because it reduces the hit latency for most memory requests.

Data compression has already been explored in GPUs. [18], [30]–[32] explore data compression in register files [30], interconnect networks [31], and caches [18], [32]. To better utilize the latency margin provided by the high-priority victim cache, HIVE adopts a low-latency BDI compression algorithm, whose effectiveness in GPUs has been validated in [18].

The register file in GPUs is fairly large but often underutilized. To address this problem, prior works propose new techniques to utilize the register file efficiently. [2] proposes a new technique that reuses the core's on-chip memories, including register files, to extend the LLC capacity for a running memory-bound workload. [33] proposes a new compiler technique that explores TLP and registers allocation trade-off with the compile-time workload characteristic analysis.

Warp throttling techniques are introduced to reduce cache contention in GPUs [7], [34]–[36]. They require empirically setting some thresholds to decide whether to throttle. This means that for some applications, warp throttling can even hurt the performance. As explained in earlier sections, the closest work to HIVE is Linebacker [7], which simply reuses idle registers as a victim cache and employs warp throttling technology. However, Linebacker's victim cache is inherited from CPUs and is suboptimal for GPUs. In contrast, HIVE is suitable for GPUs. It reduces the hit latency for most memory requests and increases the effective capacity of the victim cache. Therefore, HIVE is expected to be more feasible and applicable to real practices.

VII. Conclusion

We propose HIVE, a high-priority victim cache for accelerating GPU memory accesses. The key idea of HIVE is to prioritize using the victim cache rather than the L1D cache to handle memory requests, meaning the victim cache does not act as a supplement to the L1D cache. Our experimental results show that HIVE can significantly improve GPU performance across various applications and different architectures.

VIII. Acknowledgment

This work is supported in part by the National Key Research and Development Program of China under Grant 2022YFB2803405, in part by the National Natural Science Foundation of China under Grant 62472435, 62172430, and 62304257, in part by the STIP of Hunan Province under Grant 2022RC3065, in part by the Foundation of PDL under Grant 2023-JKWPDL-02, and in part by the Key Laboratory of Advanced Microprocessor Chips and Systems.

REFERENCES

[1] "NVIDIA Kepler Architecture Whitepaper," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf.

[2] S. Darabi, M. Sadrosadati, N. Akbarzadeh, J. Lindegger, M. Hosseini, J. Park, J. Gómez-Luna, O. Mutlu, and H. Sarbazi-Azad, "Morpheus: Extending the last level cache capacity in gpu systems using idle gpu core resources," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

[3] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module gpus for continued performance scalability," *ACM SIGARCH Computer Architecture News*, 2017.

[4] X. Xu, L. Wang, L. Xiao, L. Liu, Y. Lv, X. Xie, M. Han, and H. Liu, "ATA-Cache: Contention mitigation for gpu shared l1 cache with aggregated tag array," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[5] N. Jing, J. Wang, F. Fan, W. Yu, L. Jiang, C. Li, and X. Liang, "Cache-emulated register file: An integrated on-chip memory architecture for high performance gpgpus," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[6] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access pattern-aware cache management for improving data utilization in gpu," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017.

[7] Y. Oh, G. Koo, M. Annavaram, and W. W. Ro, "Linebacker: Preserving victim cache lines in idle register files of gpus," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[8] "NVIDIA Ampere Architecture Whitepaper," https://images.nvidia.cn/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[9] "NVIDIA H100 Tensor Core GPU Architecture Overview," https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper.

[10] E. M. Taylor and D. W. Chang, "Studying victim caches in gpus," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018.

[11] "NVIDIA Pascal Architecture Whitepaper," https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[12] Z. Jia, M. Maggioni, J. Smith, and D. P. Scarpazza, "Dissecting the nvidia turing t4 gpu via microbenchmarking," *arXiv preprint arXiv:1903.07486*, 2019.

[13] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[14] S. Damani, M. Stephenson, R. Rangan, D. Johnson, R. Kulkami, and S. W. Keckler, "Gpu subwarp interleaving," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[15] "NVIDIA Turing Architecture Whitepaper," https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[16] S. Darabi, E. Yousefzadeh-Asl-Miandoab, N. Akbarzadeh, H. Falahati, P. Lotfi-Kamran, M. Sadrosadati, and H. Sarbazi-Azad, "OSM: Off-chip shared memory for gpus," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[17] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, 2012.

[18] A. Arunkumar, S.-Y. Lee, V. Soundararajan, and C.-J. Wu, "LATTE-CC: Latency tolerance aware adaptive cache compression management for energy efficient gpus," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[19] A. Arelakis, F. Dahlgren, and P. Stenstrom, "Hycomp: A hybrid cache compression method for selection of data-type-specific compression methods," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.

[20] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007.

[21] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*, 2009.

[22] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE international symposium on performance analysis of systems and software*, 2009.

[23] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *2012 innovative parallel computing (InPar)*, 2012.

[24] A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[25] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, 2013.

[26] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[27] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Computer Architecture News*, 1990.

[28] S. M. Khan, D. A. Jiménez, D. Burger, and B. Falsafi, "Using dead blocks as a virtual victim cache," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, 2010.

[29] M. Zhang and K. Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005.

[30] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, "Warped-compression: Enabling power efficient gpus through register compression," *ACM SIGARCH Computer Architecture News*, 2015.

[31] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, "A case for core-assisted bottleneck acceleration in gpus: enabling flexible data compression with assist warps," *ACM SIGARCH Computer Architecture News*, 2015.

[32] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," *ACM SIGARCH Computer Architecture News*, 2016.

[33] X. Xie, Y. Liang, X. Li, Y. Wu, G. Sun, T. Wang, and D. Fan, "Enabling coordinated register allocation and thread-level parallelism optimization for gpus," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.

[34] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[35] S.-Y. Lee and C.-J. Wu, "CAWS: Criticality-aware warp scheduling for gpgpu workloads," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014.

[36] B. Wang, Y. Zhu, and W. Yu, "OAWS: Memory occlusion aware warp scheduling," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, 2016.

MAGCS: Multi-Agent Guided Configuration Search for Optimization Fault Detection in Logic Synthesis

Peiyu Zou[a], Xiaochen Li[a,*], Shikai Guo[b,*], Weihong Sun[c], Yuyao Xu[c], He Jiang[a]
[a]School of Software, Dalian University of Technology, Dalian, China
[b]School of Information Science and Technology, Dalian Maritime University, Dalian, China
[c]Hi-Think Technology, Corp, Dalian, China
{zoupeiyu, xiaochen.li}@mail.dlut.edu.cn, shikai.guo@dlmu.edu.cn, {sunwh, yuyao.xu}@dhc.com.cn, jianghe@mail.dlut.edu.cn

***Abstract*—Logic synthesis tools are crucial to translate high-level descriptions into optimized gate-level netlists. However, complex optimization operations and operation configurations can cause synthesis faults. To address this, we propose MAGCS, a fault detection method using multi-agent reinforcement learning to dynamically refine optimization sequences. MAGCS consists of three components: a test program selector that applies feature extraction and cosine similarity to curate diverse test programs, an optimization selector using the A2C algorithm to adaptively adjust operations and configurations, and an optimization fault verifier performing equivalence checks to pinpoint optimization-induced faults. Using MAGCS, we identified 32 confirmed faults on Vivado and Yosys, all of which are resolved. MAGCS received recognition from the Vivado community for its significant contributions to tool improvement[1].**

***Index Terms*—Logic synthesis tools, optimization sequences, optimization faults**

I. Introduction

FPGA and ASIC are widely used in complex hardware system design such as communications and aerospace [1, 2]. In this process, logic synthesis is a critical step, which transforms high-level hardware description languages (e.g., Verilog and VHDL) into optimized gate-level netlists, to improve the performance and power efficiency of the design [3]. As shown in Fig. 1, the logic synthesis process typically consists of four main stages: Translation, Constrain, Logic Optimization, and Mapping, each involving specific optimization operations and configurations. By adjusting these operations and configurations, developers can create customized optimization sequences to achieve expression simplification, timing optimization, and efficient resource allocation. However, certain optimization sequences may produce unintended results or design errors, such as increased delays from timing optimizations and the unintended removal of critical logic during simplification. This risk is particularly concerning in safety-critical domains.

Fig. 2 is an example of circuit design code for FPGA and ASIC, which operates registers using the clock signal clk and generates a 768-bit output signal y. The logic synthesis uses Yosys with the sequence `opt_clean -purge; opt_expr -mux_bool; opt_dff -nodffe; opt_reduce -full; opt_demorgan`. In this sequence, `opt_clean` and `opt_expr` are optimization operations, while `purge` and `mux_bool` are their configurations. An error occurred when using the `opt_demorgan` operation (Fault ID: 4610[2]). As indicated in the log in Fig. 2(b), the error is caused by a zero input signal to the `$reduce_or` unit, which triggers a `std::out_of_range` exception. The error occurred at line 11: `reg79 <= (~reg67[(4'h8):(3'h5)]);` Due to an incorrect signal connection, the input size was mistakenly set to zero, leading to synthesis failure. The Yosys community resolved this issue through a fix (Pull Request ID: 4612[3]), as illustrated in Fig. 2(c). The fix involved modifying the *opt_demorgan.cc* optimization file. A conditional check was added at lines 43-44 to prevent further processing when the input signal is zero.

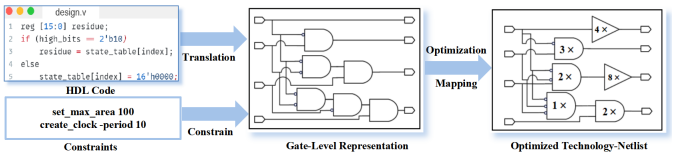


Fig. 1: Electronic Circuit Design - Logic Synthesis Flowchart.

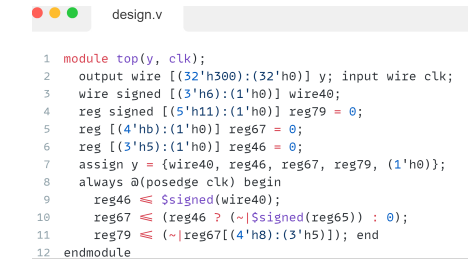
To identify faults in logic synthesis tools, several testing methods have been proposed [4, 5, 6, 7]. These methods typically rely on generators (e.g., Verismith and Vloghammer) to produce large numbers of Verilog files, which are then used as input (aka, test programs) to expose faults in synthesis tools. However, these methods often overlook the complexity introduced by user-defined optimization sequences. Methods like DeLoSo[8] address this by using heuristics to explore specific combinations of operations and parameter configurations that may trigger faults. However, DeLoSo lacks real-time adaptation based on feedback, limiting its exploration of broader optimization sequences. Additionally, DeLoSo relies on code generators to produce a single batch of test programs, which tend to exhibit similar structures within the batch, lacking sufficient diversity to comprehensively expose faults. Based on the current research landscape, fault detection in logic synthesis optimization faces two main challenges:

Challenge 1: Insufficient diversity in test programs. Fault detection relies on diverse test programs to thoroughly test the whole logic synthesis process. Limited variation among test programs restricts the ability of existing methods to reveal potential faults.

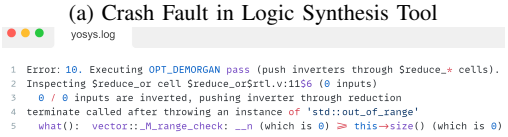
Challenge 2: Inadequate exploration of optimization sequences. Logic synthesis tools offer numerous optimization operations; for instance, Vivado has over 18 optimization operations, each with more than 10 possible operation configurations. Since only certain optimization sequences could be faulty, the vast search space of optimization sequences (e.g., over 10^{18+} for Vivado) makes exhaustive testing impractical.

To this end, we propose MAGCS, a multi-agent reinforcement learning-based method for detecting optimization faults in logic synthesis tools. To tackle the first challenge, MAGCS includes a test program selector component that uses circuit features to vectorize different test programs. MAGCS selects test programs with the greatest variability in terms of vector distances to ensure the

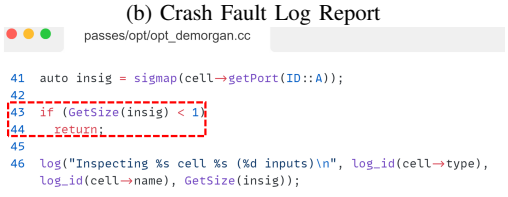
*Corresponding author (Xiaochen Li and Shikai Guo)
[1]https://support.xilinx.com/s/feed/0D54U00008Wfd2cSAB
[2]https://github.com/YosysHQ/Yosys/issues/4610
[3]https://github.com/YosysHQ/yosys/pull/4612



(a) Crash Fault in Logic Synthesis Tool



(b) Crash Fault Log Report



(c) Code Maintenance Patch

Fig. 2: Yosys Crash Fault (Fault ID: 4610)

diversity of test programs. To tackle the second challenge, MAGCS treats each optimization operation as an independent agent. MAGCS first determines the position of each optimization operation within the optimization sequence, then selects the corresponding operation configuration. Through multi-agent reinforcement learning, MAGCS dynamically adjusts the agents based on a reward function that prioritizes the fault detection ability of optimization sequences and minimizes equivalence check delays. MAGCS iteratively refines the optimization sequence to find optimal configurations for fault detection. Finally, an optimization fault verifier is used to determine if the optimized, synthesized design aligns with the intended functionality of the original design with equivalence check. If the equivalence is broken, an optimization fault could be found.

Experiments show that MAGCS outperforms baselines (i.e., DeLoSo, DynSwarm, and InitSwarm) in detecting optimization faults, with a fault detection rate 68.42% to 3100% higher than these methods. Over one month of testing on the commercial tool Vivado and the open-source tool Yosys, MAGCS identified 32 confirmed faults, demonstrating its effectiveness across different synthesis tools.

The contribution of this work includes: (1) the first multi-agent reinforcement learning-based approach for efficiently detecting optimization faults in logic synthesis tools; (2) 32 faults are found in Vivado and Yosys, all of which are confirmed and fixed; (3) the Vivado community acknowledges the value of our reported faults, recognizing our contribution to their tool improvement; (4) we have open-sourced MAGCS on Github[4].

II. RELATED WORK

Fault detection in logic synthesis tools is critical in FPGA and ASIC. Several automated testing frameworks have been proposed to generate diverse Verilog programs for fault detection. Herklotz et al. [4] developed Verismith, a tool that uses randomization techniques to generate complex Verilog programs for testing logic synthesis tools. Then, Vloghammer[5] was proposed, which can test the stability of Yosys. Ratchev et al.[6] developed VERGEN, which evaluates synthesis tool performance by generating structured random programs with specific design features. Thakur et al.[7] developed VeriGen, which uses template-based generation to test logic synthesis tools. While these approaches have been instrumental in fault detection, their primary focus is code generation; they cannot thoroughly explore faults arising from optimization sequences.

For optimization fault detection, Jiang et al. [8] proposed DeLoSo, which utilizes a heuristic method to explore optimization sequences that trigger faults in synthesis tool's optimization operations. However, DeLoSo faces limitations, such as being prone to local optima and not fully addressing test program diversity.

Overall, existing approaches lack thorough testing of optimization operations and operation configuration. This study effectively combines diverse optimization sequences with varied test programs to improve the effectiveness of fault detection.

III. MAGCS FRAMEWORK

To detect optimization faults in logic synthesis tools, we propose MAGCS, a multi-agent reinforcement learning-based fault detection method. As shown in Fig. 3, MAGCS has three core components to collaboratively select and validate optimization sequences.

A. Test Program Selector

To build a diverse and representative set of test programs, MAGCS employs a selection method based on feature extraction and similarity metrics. Given an initial set P of n_{seed} Verilog test programs, MAGCS represents each test program as a feature vector to encompass a diverse range of logic structures and timing characteristics.

Specifically, previous studies [9] have proposed five categories of features to represent a Verilog program, including

- *data processing and operations*, such as the number of arithmetic, logical, and comparison operators, which characterize data operation complexity;
- *data flow control and representation*, such as the number of assignment statements, variable declarations, and numerical representations, reflecting the synchronization and complexity of data flow;
- *structuring and modularization*, such as the number of module instantiation and declarations, describing the modular design traits of Verilog programs;
- *control flow and logic*, such as the number of conditional statements and loop control, which characterize the logical control and decision-making in Verilog programs;
- *abstraction and reuse*, such as the number of parameter definitions and scope identifiers, which capture the handling of parameterization and module reuse in Verilog programs.

We further add the sixth category of feature, namely *timing features*, which analyzes the characteristics of the timing behavior of hardware circuits, including the number of clock signals, flip-flops, edge-triggered logic, and non-blocking assignments in Verilog programs. These timing features are important for describing timing control in FPGA and ASIC designs, particularly in timing-sensitive circuits, enabling a more comprehensive characterization of Verilog program behavior.

Based on these features, MAGCS converts each test program p_i in P into a feature vector $F_i = [f_{i1}, f_{i2}, ..., f_{im}]$, where m represents the number of features. Each dimension in F_i is a quantified feature, e.g., f_{i1} counts the number of arithmetic operators in p_i, and f_{i2}

[4]https://github.com/MAGCS-method/MAGCS

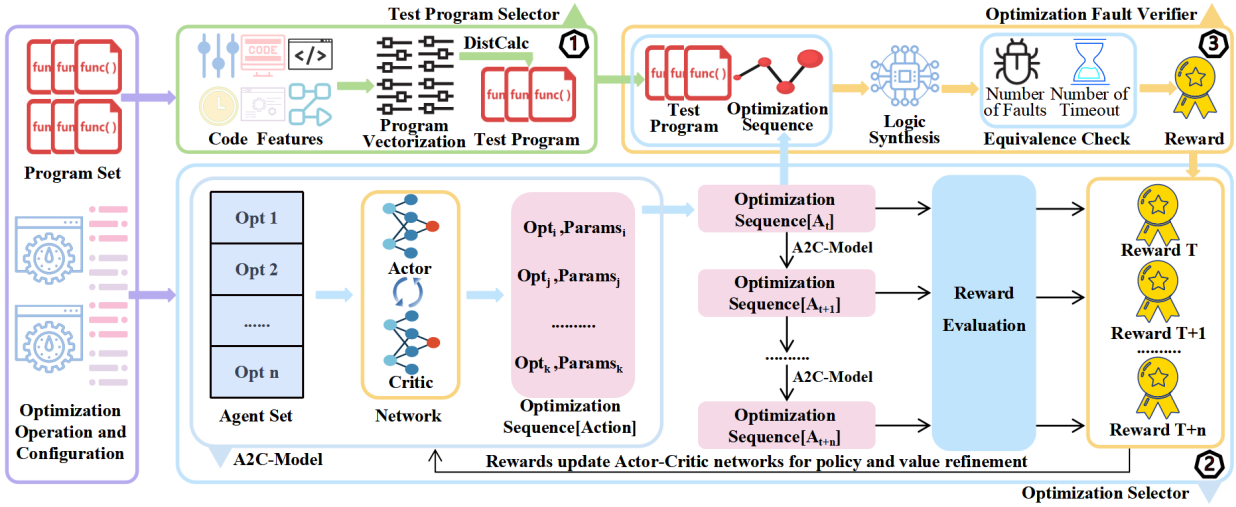


Fig. 3: The framework of MAGCS

is the number of logical operators in p_i. Once the feature vector is constructed, MAGCS normalizes each feature dimension j by:

$$\hat{f}_{ij} = \frac{f_{ij} - \min(f_{1j}, \ldots, f_{nj})}{\max(f_{1j}, \ldots, f_{nj}) - \min(f_{1j}, \ldots, f_{nj})} \quad (1)$$

where f_{ij} is the original feature value of test program p_i in dimension j, and $\min(f_{1j}, \ldots, f_{nj})$ and $\max(f_{1j}, \ldots, f_{nj})$ are the minimum and maximum values of dimension j across all n test programs.

After normalization, the feature vector of p_i is expressed as: $\hat{F}_i = [\hat{f}_{i1}, \hat{f}_{i2}, ..., \hat{f}_{im}]$. This normalization compresses the feature values of each dimension into the [0, 1] range, ensuring that all dimensions are on the same scale and avoiding bias.

Next, MAGCS uses cosine distance to measure the dissimilarity between test program features. Cosine distance reflects the angular difference between two feature vectors and is defined as: $d\left(\hat{F}_i, \hat{F}_j\right) = 1 - \frac{\hat{F}_i \hat{F}_j}{\left\|\hat{F}_i\right\| \left\|\hat{F}_j\right\|}$, where $\hat{F}_i \hat{F}_j$ is the inner product of the feature vectors, and $\left\|\hat{F}_i\right\|$ and $\left\|\hat{F}_j\right\|$ is the magnitudes of the vectors. A large $d\left(\hat{F}_i, \hat{F}_j\right)$ means two test programs are more dissimilar in the feature space.

To select a maximally diverse test program set P_{sel}, MAGCS employs a recursive greedy method. First, a test program p_1 is randomly selected from the initial set P to serve as the first program in P_{sel}. For each remaining test program $p_i \in P$, we compute the sum of its cosine distances to all already-selected programs in P_{sel}:

$$D\left(p_i\right) = \sum_{p_j \in P_{sel}} d\left(\hat{F}_i, \hat{F}_j\right) \quad (2)$$

Next, the test program p_k with the greatest dissimilarity to the current programs in P_{sel} is selected, satisfying the following condition:

$$p_k = \underset{p_i \in P / P_{sel}}{\arg\max}\ D\left(p_i\right) \quad (3)$$

Using this method, we iteratively build a diverse subset P_{sel} from P until it includes n_{sel} test programs. This method maximizes the diversity within the feature space, facilitating broader fault detection across various optimization scenarios in logic synthesis tools.

B. Optimization Selector

During the logic synthesis optimization process, the selection of optimization operations and their configurations creates a vast search space. For instance, in Yosys, commonly used optimization operations include `opt_expr`, `opt_clean`, `opt_merge`, `opt_muxtree`, and `opt_demorgan`. Each of them has various operation configuration. For example, `opt_expr` provides configurations such as `opt_expr -mux_bool` for converting Boolean expressions into multiplexers and `opt_expr -full` for applying comprehensive logic transformations. The execution order of these operations (i.e., the optimization sequence) directly affects the quality of the synthesis optimization.

To navigate this complex space, the optimization selector leverages a multi-agent reinforcement learning framework A2C [10] for optimizing the selection of operations and their configurations. In A2C, agents make decisions by selecting actions from an action space, which represents possible optimization operations and configurations, based on the current state of the system. The state space contains all relevant system information at a given time. By continuously learning from feedback rewards, agents optimize their decisions, identifying the best sequence of operations to improve fault detection. Let the state space S_t represent the optimization sequence and its corresponding operation configuration at a given time step t. S_t is composed of a set of optimization operations and their configurations:

$$S_t = \{(opt_{1,t}, params_{1,t}), \ldots (opt_{i,t}, params_{i,t}), ..., (opt_{n,t}, params_{n,t})\} \quad (4)$$

where $opt_{i,t}$ denotes the selection of the i-th optimization operation at time t, and $params_{i,t}$ represents its operation configuration. For instance, at time step t, the state might be:

$$S_t = \{(opt_expr, fine), (opt_clean, purge), (opt_merge, nomux)\} \quad (5)$$

The action space A defines the decisions that the agent can make at each time step. Each agent's actions include selecting the operation configuration of the current optimization operation as well as dynamically adjusting the order of these operations. Specifically, action $a_t \in A$ consists of two parts: $opt_{i,t+1}$, which selects the next optimization operation, and $params_{i,t+1}$, which specifies the configuration for $opt_{i,t+1}$. Hence, the action a_t is expressed as:

$$a_t = \{(opt_{i,t+1}, params_{i,t+1})\} \quad (6)$$

To guide the agent in selecting effective actions in this complex optimization space, we designed a reward function that maximizes the fault detection ability of each action a_t while minimizing unnecessary

equivalence check timeouts. Specifically, the reward function R for an action a_t is defined as:

$$R = \theta \cdot \frac{N_{fault}}{N_{fault}+1} - (1-\theta) \cdot \frac{N_{timeout}}{N_{timeout}+1} \tag{7}$$

where N_{fault} represents the number of faults detected when applying the current optimization sequence on all test programs in P_{sel}, and $N_{timeout}$ represents the number of timeouts encountered during equivalence check for the same sequence and test set. Parameter $\theta \in [0, 1]$ controls the balance between positive rewards and negative penalties. Detecting an optimization fault results in a positive reward, encouraging the agent to explore and find more faults, thereby improving the effectiveness of the optimization sequence. Timeouts during equivalence check are penalized because they delay verification of the design against the netlist and disrupt the overall testing process. By using timeouts as a negative reward, MAGCS reduces unnecessary delays to ensure test efficiency.

Our framework adopts an Actor-Critic (A2C) approach with two primary networks: a policy network (Actor) and a value network (Critic) [11]. The policy network learns to select optimization actions (operations and configurations) that maximize cumulative rewards, while the value network estimates the expected reward value of the current state to enhance decision accuracy.

At each time step t, the agent uses the policy network to select an optimization action a_t based on the current state s_t, specifying the next optimization operation and its configuration. Executing a_t generates a new state s_{t+1}, reflecting the updated sequence of optimization operations and configurations.

After each action, an immediate reward R is calculated, evaluating the optimization sequence based on detected faults and equivalence check timeouts. The critic network estimates the expected value of the state to guide further decisions. A2C iteratively refines the agent's policy, enabling effective selection of optimization operations and configurations. The objective is to maximize the cumulative discounted reward:

$$\max_{\omega} E\left[\sum_{t=0}^{T} \gamma^t R_t\right] \tag{8}$$

where ϖ is the policy network parameter, $\gamma \in [0, 1]$ is the discount factor that reduces the influence of rewards in future time steps, and R_t is the immediate reward at time step t. Based on prior studies [10, 11], we set ϖ to 0.001 and γ to 0.99.

Through iterative training, the agent gradually optimizes its strategy, making optimal decisions in the complex optimization space and maximizing the cumulative reward. The optimization selector continuously improves fault detection by dynamically adjusting the optimization operations and configurations, ensuring that each stage of testing delivers high fault detection capacity and efficiency.

C. Optimization Fault Verifier

After identifying each optimization sequence with the optimization selector, it is converted into optimization instructions applied to the design file—a Verilog or HDL circuit representation—during synthesis. In some cases, specific test programs and optimization sequences may cause unexpected synthesis terminations, as discussed in Fig. 2. This type of optimization fault indicates that certain operations or configurations within the optimization process contain issues, directly causing the synthesis to fail. Feedback on detected faults and timeouts from each sequence is then relayed back to the optimization selector, helping refine its strategy.

For optimization sequences that complete synthesis successfully, the system generates synthesis file and conducts equivalence check to ensure that the synthesized logic matches the original design's functionality. The equivalence check uses formal verification tools to convert both the original design and the synthesized files into comparable logic expressions, verifying whether their outputs match across all input conditions. Specifically, the core of the equivalence check is to ensure that the output logic of the source design L_{src} matches the synthesized design L_{syn} for all inputs x:

$$\forall x \in X, L_{src}(x) = L_{syn}(x) \tag{9}$$

Here, X represents the complete set of possible inputs. If under certain input conditions, $L_{src}(x) \neq L_{syn}(x)$, this indicates that a functional error occurred during the synthesis process.

By monitoring the synthesis process and conducting equivalence check, it is possible to determine whether the optimization sequence led to faults. If the synthesis process terminates unexpectedly or if the equivalence check fails, this signifies an issue with the optimization process. The optimization fault verifier ensures that any errors introduced during optimization are detected.

IV. EVALUATION

We evaluated the effectiveness of MAGCS by answering the following research questions (RQs): RQ1 – Effectiveness of MAGCS Compared to Baseline Methods, RQ2 – Impact of Parameter on MAGCS, and RQ3 – Effectiveness of MAGCS in Finding Real Faults.

A. Experimental Setup

Hardware Environment: Our experiments were conducted on an x86_64 computer running Ubuntu 18.04.01, equipped with an Intel Core i9-12900 CPU @ 2.80GHz with 24 cores and 128GB RAM.

Software and Test Program Generation: Our experiments employed two mainstream logic synthesis tools: the commercial tool Vivado (version 2024.1) and the open-source tool Yosys (version 0.41+126). We chose the latest versions, as they are more likely to contain recent fault fixes, which are particularly critical for developers [8]. To generate the initial test set P, we used Verismith [4] for its ability to produce diverse and complex Verilog programs essential for fault detection. Following configurations from prior studies [8], we generated an initial set of $n_{\text{seed}} = 200,000$ programs, from which a greedy method selected a diverse subset of $n_{\text{sel}} = 1,000$ programs.

Types of Errors: We identified four types of optimization faults: (1) Crash Fault: a crash during synthesis with specific optimization sequence; (2) Performance Fault: a hang or stall in the synthesis process; (3) Parsing Fault: errors in parsing the design files; and (4) Logic Fault: discrepancies between the synthesized and original design due to logic errors.

B. Baseline Methods and Testing Strategy

To assess the performance of MAGCS, we used nine methods for comparison. These methods are based on four basic methods, i.e., Default, InitSwarm, DynSwarm, and DeLoSo [8]. Each method has two steps, i.e., test program selection and optimization sequence generation. For test program selection, we employed two strategies: *Rand*, where 1,000 test programs are randomly selected from a pool of 200,000, and *Div*, where 1,000 diverse test programs are selected using the test program selector proposed in Section 3.1.

For optimization sequence generation, we have:

Default: Default optimization sequences in synthesis tools (e.g., opt-fast and opt-full in Yosys, and O1, O2, O3 in Vivado) are used.

InitSwarm: Using the default optimization operations in their fixed order, with only the operation configuration randomized.

TABLE I: Effectiveness of MAGCS in Fault Detection

Tools	Default		InitSwarm		DynSwarm		DeLoSo		MAGCS	MAGCS
	Rand	Div	Rand	Div	Rand	Div	Rand	Div	Rand	
Vivado	0	0	1	1	2	3	7	9	16	26
Yosys	0	0	0	1	1	1	2	3	3	6
Total	0	0	1	2	3	4	9	12	19	32

DynSwarm: The positions of optimization operations are randomly rearranged, and operation configurations are also randomly assigned.

DeLoSo: Genetic algorithm is used to explore optimization sequences by adjusting the positions of optimization operations and tuning their configurations, aiming to identify combinations that may trigger faults.

When combining the two steps, we have eight method variants, i.e., Default-Rand, Default-Div, InitSwarm-Rand, InitSwarm-Div, DynSwarm-Rand, DynSwarm-Div, DeLoSo-Rand, and DeLoSo-Div. For example, Default-Rand means that we use the default optimization sequence with the randomly selected 1,000 test programs to test synthesis tools. Additionally, we create a variant of MAGCS as the ninth baseline, i.e., *MAGCS-Rand*, where the test program selector in MAGCS is replace with the Rand strategy.

Each testing method was allocated a fixed runtime. During each iteration, methods like MAGCS dynamically generated new optimization sequences, while Default used predefined or randomized configurations. These optimization sequences were applied for logic synthesis. We analyzed both the synthesis process and checked for equivalence between the synthesized netlist and the original design to identify potential faults. When mismatches were detected, functional simulations were performed to gather more detailed data, enabling easier replication of the faults by the community.

For each detected fault, failed assertions and logs were analyzed to remove duplicates. The Verilog program that triggers the faults was incrementally reduced to its minimal form for reproduction. The simplified design, the triggering optimization sequence, and relevant error logs were submitted to the development team for verification as either new or known issues.

C. RQ1: Effectiveness of MAGCS and Baselines

To answer RQ1, we use MAGCS and nine baseline methods to test logic synthesis tools for over one month. This period includes Verilog test program generation, test case selection and optimization selector with each method, and fault verification. This evaluation period aligns with previous studies [8]. For the tests, we used the latest versions of Vivado (2024.1) and Yosys (0.41+126). We use two identical machines to run all methods.

As shown in TABLE I, MAGCS identified 26 faults in Vivado and 6 in Yosys, significantly outperforming other methods. In comparison, DeLoSo-Div found 9 faults in Vivado and 3 in Yosys, totaling 12. DynSwarm-Div detected 4 faults, while InitSwarm performed poorly. Default-Rand and Default-Div did not find any faults, as the default optimization sequence are typically extensively tested before release. MAGCS demonstrated superior fault detection ability across different tools by dynamically optimizing sequence combinations through multi-agent reinforcement learning.

We also used a Venn diagram to further analyze the differences between MAGCS and other baselines in fault detection. Since Default and InitSwarm-Rand did not detect any faults in the experiment, they were excluded from the Venn diagram. As shown in Fig. 4, the faults found by MAGCS is a superset of those found by other baselines. This indicates that MAGCS can both effectively select diverse test programs and explore a broader optimization configuration space, uncovering faults missed by other methods.

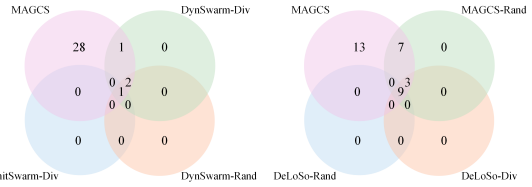


Fig. 4: Relationship of faults found by different methods.

TABLE II: Faults Detected by MAGCS with Different θ Values

Value of θ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Vivado Faults	1	2	4	3	7	8	11	13	11	9	5
Yosys Faults	0	1	1	1	2	3	4	4	3	1	1
Total	1	3	5	5	9	11	15	17	15	12	6

Conclusion: MAGCS is highly effective in detecting optimization faults in logic synthesis tools, surpassing baseline methods by triggering more optimization faults. The faults found by MAGCS form a superset of those detected by other baselines.

D. RQ2: Impact of Parameter on MAGCS

To answer RQ2, we explore a key parameter in MAGCS: the coefficient θ in the reward calculation formula (Equation 7). This coefficient balances the detection of optimization faults with the penalty for timeouts during equivalence check. In the experiment, the value of θ ranged from 0.0 to 1.0, with intervals of 0.1. Notably, when $\theta = 0.0$, the system only penalizes timeouts without incentivizing fault detection. In contrast, when $\theta = 1.0$, the system entirely ignores timeouts and focuses solely on the number of detected optimization faults.

As shown in TABLE II, we recorded the number of faults detected by MAGCS in Vivado and Yosys for varying values of θ. As θ increased from 0.0 to 0.7, the number of detected faults steadily rose. In this range, MAGCS prioritized fault detection, while allowing some timeouts, which helped reveal additional faults. However, as θ exceeded 0.7, the lack of penalty for timeouts began to interfere with testing, causing verification disruptions and fewer detected faults. At $\theta = 0.7$, MAGCS achieved the best balance between fault detection and time management, with high detection efficiency.

Conclusion: When θ is set to 0.7, MAGCS achieves the best performance in detecting optimization faults in logic synthesis tools.

TABLE III: Details of Faults Found by MAGCS

ID	Tools	Fault ID	Type	Title
1	Vivado	8TyRQBSA3	CF	Vivado Crash in HARTHOptPost::prepDsps()
2	Vivado	8gjjsGSAQ	CF	Vivado Crash in HARTLOptAbc::runNlOpt()
3	Vivado	8cMtrMSAS	CF	HARTNlOptimize::modOptimize() Error in Vivado
4	Vivado	7BrmgwSAB	CF	Crash in HARTHOptPost::optimize() During Synthesis
5	Vivado	8boSFLSA2	CF	NDup::copyModule() Causes Vivado Setback
6	Vivado	8a5wHySAI	CF	HARTSWorker::runInternal() Crashing Vivado
7	Vivado	8YNb4PSAT	CF	Vivado Obstructed by hdi::tcltasks() Crash
8	Vivado	8jW2ngSAC	CF	HARTSWorker::runJob() Causing Vivado Failure
9	Vivado	8bGuVzSAK	CF	Vivado Crash in ConstProp::cleanup()
10	Vivado	8gjjtHSAQ	CF	Stack Overflow Check Causes Vivado Issues
11	Vivado	8gjjv8SAA	CF	Vivado Crash in GDpGen::implementBinary()
12	Vivado	8jVeCmSAK	CF	Crash Due to HARTTUpdateTNInstC::updateCell()
13	Vivado	8jW2nfSAC	CF	Optimize1::optimize() Malfunction in Vivado
14	Vivado	8aRj9KSAS	CF	HSynMod::connectInputPin() Crash in Vivado
15	Vivado	8jW2xgSAC	CF	Crash in NDbC::uniquePrefixes() in Vivado
16	Vivado	7AD9ZWSA1	LF	unsigned() Function Error Due to Synthesis Parameters
17	Vivado	8X0u8WSAR	PFF	Vivado Freezes During Ubuntu Synthesis
18	Vivado	8ZY2lqSAD	PFF	Vivado Optimization Termination During Synthesis
19	Vivado	8Vzsu9SAB	PFF	Large Design Causes Vivado to Freeze
20	Vivado	8aRjBGSA0	PFF	Ubuntu Design Optimization Freezes Vivado
21	Vivado	8boBgSSAU	PFF	Vivado Stalls During Synthesis on Ubuntu
22	Vivado	8bGuWxSAK	PFF	Vivado Hangs on Specific Verilog File During Synthesis
23	Vivado	8gjjwGSAQ	PFF	Synthesis Hang Issue in Vivado
24	Vivado	8XGDKYSA5	PFF	Vivado Stalls on Specific File in Optimization
25	Vivado	8YmcY9SAJ	PFF	Vivado Optimization Causes Process Hang
26	Vivado	8XVl5vSAD	PFF	Ubuntu Synthesis Process Hangs in Vivado
27	Yosys	4610	CF	Yosys Synthesis std::out_of_range Error
28	Yosys	4486	LF	Yosys Optimization Causes Incorrect Output
29	Yosys	4491	LF	Custom Yosys Passes Cause Faulty Synthesis
30	Yosys	4478	LF	Yosys Optimization Error in PEEPOPT Pass
31	Yosys	4427	PF	Yosys Verilog Parsing Error After File Read
32	Yosys	4458	PFF	Yosys Synthesis Hash Table Overflow

Note: Crash faults (CF), Logic faults (LF), Parsing faults (PF), Performance faults (PFF).

Vivado.log

```
# An unexpected error has occurred (11)
Stack:
/lib/x86_64-linux-gnu/libc.so.6(+0x42520) [0x7fcb9d242520]
/Vivado/2024.1/lib/lnx64.o/libtcl8.6.so(TclNRRunCallbacks+0x47) [0x7fcb9d484497]
/Vivado/2024.1/lib/lnx64.o/librdi_commontasks.so(+0x25c6d1) [0x7fcb9245c6d1]]
/Vivado/2024.1/lib/lnx64.o/librdi_synth.so(HARTSWorker::runJob(HARTSWorkerConfig&)+0x3f)
/Vivado/2024.1/lib/lnx64.o/librdi_synth.so(HARTSWorker::mainLoop()+0x8b) [0x7fcb64d5bc4b]
/Vivado/2024.1/lib/lnx64.o/librdi_synth.so(HARTNDb::parallelListen(char const*)
/Vivado/2024.1/lib/lnx64.o/librdi_synth.so(+0x14a4a74) [0x7fcb642a4a74]
/Vivado/2024.1/lib/lnx64.o/librdi_synth.so(+0x13623e5) [0x7fcb641623e5]
/Vivado/2024.1/lib/lnx64.o/libtcl8.6.so(TclNRRunCallbacks+0x47) [0x7fcb9d484497]
```

Fig. 5: Crash Fault Log Report for Fault ID: 8jW2ngSAC



(a) Netlist after correct synthesis by the logic synthesis tool

(b) Erroneous netlist resulting from a logic synthesis tool fault

Fig. 6: Comparison of waveform differences

E. RQ3: Effectiveness in Finding Real Faults

To answer RQ3, we used MAGCS to detect optimization faults in Vivado and Yosys. As shown in TABLE III, we have reported a total of 32 optimization faults to the Vivado and Yosys development teams or communities, including 16 crash faults, 11 performance issues, 1 parsing fault, and 4 logic faults. All reported faults have been confirmed. We also shared the details of these fault on GitHub. Below, we provide an example for each category of faults.

Crash Fault: Vivado Fault ID: 8jW2ngSAC[5] occurred due to improper resource scheduling during complex optimization operations. As shown in the crash log in Fig. 5, the crash happened while executing a synthesis task. The Vivado thread scheduling mechanism failed to properly manage multi-task loads, particularly for high-complexity tasks like hierarchy flattening and clock-gating conversion. The log indicates that the functions $HARTSWorker :: runJob()$ and $HARTSWorker :: runInternal()$ encountered synchronization issues during parallel task execution, leading to resource contention. Additionally, the $parallelListen()$ function failed to properly schedule parallel tasks, causing thread blocking and ultimately triggering the crash.

Logic Fault: Vivado Fault ID: 7AD9ZWSA1[6] is the inconsistency of simulation waveform between the original design and the synthesized netlist. At the 159ps timestamp in the waveform, as shown in the red box in Fig. 6, a clear discrepancy is evident. The first (correctly synthesized) netlist shows a value of 802, while the second (incorrectly synthesized) netlist shows a value of 800. This significant behavioral inconsistency issue arose in the module responsible for logic optimization during synthesis. Logic optimization failed to maintain the timing consistency of the design when dealing with certain synchronous or nested logic. Although subtle, this timing discrepancy can seriously cause the overall functionality of the circuit to fail, especially in high-speed designs.

Parsing Fault: Yosys Fault ID:4427[7], as shown in Fig. 7, indicates that Yosys failed to continue execution while generating the RTLIL representation due to an assertion failure. The assertion failure occurred at $frontend/ast/ast.cc : 855$, where $node- > bits == v$ failed. In this case, Yosys encountered a bit-width mismatch when generating the RTLIL representation for the modules top and

[5]https://adaptivesupport.amd.com/s/question/0D54U00008jW2ngSAC
[6]https://support.xilinx.com/s/question/0D54U00007AD9ZWSA1
[7]https://github.com/YosysHQ/Yosys/issues/4427

yosys.log

```
Generating RTLIL representation for module \top
Generating RTLIL representation for module \module282
ERROR: Assert 'node→bits ==v' failed in frontends/ast/ast.cc:855.
```

Fig. 7: Parsing Fault Log Report for Fault ID: 4427

yosys.log

```
"terminate called after throwing an instance of 'std::length_error'
what(): hash table exceeded maximum size."
```

(a) Log Report

```
kernel/hashlib.h
@@ -188,6 +188,7 @@ inline unsigned int mkhash(const T &v) {
188 188
189 189   inline int hashtable_size(int min_size)
190 190   {
    191 +       // Primes as generated by https://oeis.org/A175953
191 192         static std::vector<int> zero_and_some_primes = {
192 193                 0, 23, 29, 37, 47, 59, 79, 101, 127, 163, 211, 269, 337, 431, 541, 677,
193 194                 853, 1069, 1361, 1709, 2137, 2677, 3347, 4201, 5261, 6577, 8231, 10289,
@@ -196,7 +197,9 @@ inline int hashtable_size(int min_size)
196 197                 897133, 1121423, 1401791, 1752239, 2190299, 2737937, 3422429, 4278037,
197 198                 5347553, 6684443, 8355563, 10444457, 13055587, 16319519, 20399411,
198 199                 25499291, 31874149, 39842687, 49803361, 62254207, 77817767, 97272239,
199     -               121590311, 151987889, 189984863, 237481091, 296851369, 371064217
    200 +               121590311, 151987889, 189984863, 237481091, 296851369, 371064217,
    201 +               463830313, 579787991, 724735009, 905918777, 1132398479, 1415498113,
    202 +               1769372713
200 203         };
```

(b) Code Maintenance Patch

Fig. 8: Yosys Performance Fault (Fault ID: 4458)

$module282$. The error occurred in the frontend Abstract Syntax Tree parsing phase, where Yosys has a signal width mismatch during the conversion from Verilog program to RTLIL.

Performance Fault: Yosys Fault ID:4458[8] was triggered during logic synthesis optimization compilation. The hash table size exceeded the system's allowable limit, triggering a $std :: length_error$ exception, as shown in Fig. 8(a). This issue arose primarily because Yosys did not handle hash table size calculations properly when processing large design files, leading to rapid hash table expansion during optimization. As shown in Fig. 8(b), Yosys developers resolved this issue[9] by adjusting the hash table size calculation logic, using a larger prime number table to allocate hash table capacity. This ensured that the hash table would function correctly when handling large designs.

Conclusion: MAGCS demonstrates excellent capabilities in detecting optimization faults in logic synthesis tools. MAGCS successfully identifies 32 optimization faults, which are all confirmed and fixed.

V. Conclusion

Optimization faults in logic synthesis tools can critically impact hardware design accuracy and performance; therefore, effective fault detection is essential. We present MAGCS, a reinforcement learning-based multi-agent method to enhance fault detection in logic synthesis optimizations. Tested on Vivado and Yosys, MAGCS demonstrated high fault detection capabilities, finding 32 confirmed faults in both tools. The Vivado community acknowledged our fault reports, and shared a note of gratitude[10]: *"Thank you for reporting a series of Vivado synthesis issues and providing the related example designs. Most of the issues can be reproduced and have been reported to AMD's developers to improve our tool."*.

VI. Acknowledgment

This work was supported by Key Research and Development Project of Liaoning Province under Grant No. 2024JH2/102400059, and in part by the National Natural Science Foundation of China under Grant No. 62202079, No. 62032004, No. 62472062.

[8]https://github.com/YosysHQ/Yosys/issues/4458
[9]https://github.com/YosysHQ/Yosys/pull/4471
[10]https://support.xilinx.com/s/feed/0D54U00008Wfd2cSAB

REFERENCES

[1] G. Yan, X. Liu, and H. Wang, "Fast fpga accelerator of graph cut algorithm with out-of-order parallel execution in folding grid architecture," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[2] G. Brilli, G. Valente, A. Capotondi, P. Burgio, T. Di Masciov, and Valente, "Fine-grained qos control via tightly-coupled bandwidth monitoring and regulation for fpga-based heterogeneous socs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[3] L. Witschen, T. Wiersema, L. Reuter, and M. Platzner, "Search space characterization for approximate logic synthesis," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 433–438.

[4] Y. Herklotz and J. Wickerson, "Finding and understanding bugs in fpga synthesis tools," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 277–287.

[5] C. Wolf, "VlogHammer https://github.com/YosysHQ/VlogHammer," 2019.

[6] B. Ratchev, M. Hutton, G. Baeckler, and B. van Antwerpen, "Verifying the correctness of fpga logic synthesis algorithms," in *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, 2003, pp. 127–135.

[7] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[8] H. Jiang, P. Zou, X. Li, z. Zhou, and S. Guo, "Deloso: Detecting logic synthesis optimization faults based on configuration diversity," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[9] "Losyte https://github.com/LoSyTe-Logic-Synthesis-Tool-Test/LoSyTe," 2024.

[10] V. Mnih, "Asynchronous methods for deep reinforcement learning," *arXiv preprint arXiv:1602.01783*, 2016.

[11] J. Lu, J. Yang, S. Li, Y. Li, W. Jiang, and J. Dai, "A2c-drl: Dynamic scheduling for stochastic edge-cloud environments using a2c and deep reinforcement learning," *IEEE Internet of Things Journal*, 2024.

Differentiable Net-Moving and Local Congestion Mitigation for Routability-Driven Global Placement

Wenchao Li[1], Hongxi Wu[1], Duanxiang Liu[2], Xingquan Li[3], Wenxing Zhu[1*]
[1]Fuzhou University, China; [2]Shanghai UniVista Industrial Software Group Co., Ltd., China; [3]Pengcheng Laboratory, China

***Abstract*—Routability-driven global placement is a major challenge in modern VLSI physical design, for which mitigating routing congestion is a critical approach. Cell inflation can effectively address local routing congestion and is widely adopted, but with the issue of over-inflating or moving cells back into congested areas. Minimizing the congestion within a net bounding box is effective for alleviating global routing congestion, but the bounding box may be too large and contain congestion not contributed by the net. To address the first issue, we propose a momentum-based cell inflation technique that considers historical inflation ratios for mitigating local routing congestion. Then, we construct a differentiable global congestion function, developed from Poisson's equation, and introduce virtual standard cells onto two-pin nets to accurately guide net movements for mitigating global routing congestion. Furthermore, to improve pin accessibility, we adjust placement density around power and ground rails according to the routing congestion in global placement. The proposed techniques are integrated into an electrostatic-based global placement framework. Experiments on the ISPD 2015 contest benchmarks show that our framework achieves better routability results, with an average of 40% DRVs reduction and comparable wirelength and via count, compared to the leading routability-driven placer.**

***Index Terms*—routability-driven global placement, eliminating congestion, cell movement.**

I. Introduction

Ideally, placement and routing should be performed simultaneously in VLSI physical design. However, this is highly challenging due to their different problem nature and high computational complexity. Therefore, most studies follow the placement-then-routing flow and investigate ways to balance placement and routing to improve overall design efficiency [1, 2, 3, 4, 5]. With the rapid growth of design complexity and advances in process technology, routability has become a big issue in modern circuit design. Separation of placement and routing often leads to a large number of design rule violations after detailed routing. Therefore, routability-driven placement becomes critical. Optimizing a placement by considering routability can effectively address the routability and reduce the complexity in the later routing stage [3, 6, 7]. This work focuses on routability-driven global placement problem.

During global placement, the routability issue is complicated by diverse sources of routing congestion. As noted in [2], congestion can be divided into local routing congestion and global routing congestion. Fig. 1(a) shows both types of congestion. Local routing congestion occurs when there are too many cells in a given placement region, this congestion can be alleviated by relocating cells from the congested placement region. Global congestion occurs when too many nets traverse a G-cell, this congestion cannot be alleviated by moving cells out of the congested placement area, since there may be no cells in the congested region. Mitigating both types of congestion is a critical issue in routability-driven global placement.

Many strategies have been proposed to consider routability during the global placement stage. Cell inflation is an effective approach for mitigating local routing congestion. This approach usually utilizes a global router to create a routing congestion map and then applies cell inflation to artificially increase the density of congested areas, encouraging cell displacement from these regions [3, 5, 8]. One approach of cell inflation [3, 5] only considers the present congestion, when cells are moved to an uncongested area, these cells are not inflated, causing the cells to return to the previously congested areas inadvertently, and these areas may become congested again [9]. Another approach is inflating cells monotonically increasingly by considering historical inflation ratios [8]. This technique may lead to over-inflation even when cells have been moved away from the congested area. Therefore, to maximize the efficacy of cell inflation, it is crucial to design a dynamic cell inflation scheme by incorporating historical inflation ratios.

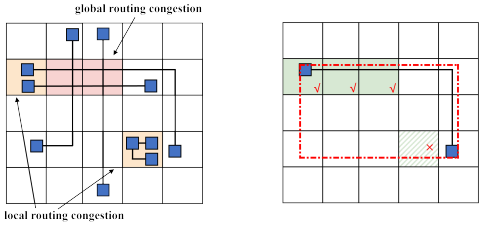


Fig. 1: (a) Local routing congestion and global routing congestion. (b) A net from (a); the red dashed box represents the bounding box (BB) of the net, the green area indicates routing congestion, and the congestion in the lower right corner is unrelated to the net.

To alleviate global routing congestion, it is an effective approach that adds a congestion penalty based on the bounding box (BB) of a net to the objective function of the global placement mathematical model. Within the deep learning framework, [4] leverages RUDY [10], PinRUDY and MacroRegion features, which are calculated based on BB, for forward propagation to generate a congestion map, followed by backpropagation to obtain routability-driven cell gradients. Nevertheless, RUDY treats all regions within the BB equally, overlooking the specific congestion situation. [6] introduces a congestion penalty for G-cell edges, minimizing the usage of G-cell edges based on nets' locations. Specifically, the congestion penalty involves every edge of G-cells surrounding the BB of a net, and the achieved effect is a reduction in the cells' distance within the net rather than direct congestion mitigation. [2] gives a BB-based congestion-aware net penalty model to reduce the net's overlap with congested areas. However, the BB may be too large and contain congestion that is not contributed by the net. For instance, as Fig. 1(b) illustrates, congestion within a net's BB may not be caused by the net itself, the lower right congested region stems from excess cell clustering, but [2] penalizes the net for this unrelated congestion.

In addition to local and global routing congestion, pin accessibility is another critical issue closely tied to routability. Existing methods

*Corresponding author: Wenxing Zhu, email: wxzhu@fzu.edu.cn.

[11, 12, 13] primarily optimize pin accessibility at the detailed placement stage, using local perturbations such as cell shifting, cell flipping, cell swapping, and cell replacement. However, pin accessibility has not been addressed at the global placement stage, which would be helpful for routability.

To address the above issues, we propose an analytical routability-driven global placement framework that includes net and multi-pin cell moving strategies, and a momentum-based cell inflation technique, which can simultaneously alleviate local and global routing congestion. Additionally, we consider optimizing pin accessibility during global placement to enhance routability further. The main contributions of this work are summarized as follows:

- We propose a differentiable congestion function developed from Poisson's equation and introduce virtual cells onto two-pin nets. The function is penalized into the global placement model for effectively guiding net movements to mitigate global routing congestion.
- A momentum-based cell inflation technique is developed to incorporate historical inflation ratios in calculating cells' current inflation ratios to mitigate local routing congestion more effectively.
- To improve pin accessibility, we adjust the placement density around power and ground rails during global placement, according to the routing congestion situation of the related region.
- We integrate the proposed techniques into an electrostatic global placement framework. Experiments on the ISPD 2015 detailed-routing-driven placement contest benchmarks demonstrate that our algorithm improves routability, achieving an average of 40% reduction in DRVs while maintaining comparable wirelength and via count, compared to the leading routability-driven placer Xplace-Route.

II. Analytical Global Placement

This section introduces the wirelength-driven analytical global placement problem, and presents our formulation of analytical routability-driven global placement problem.

A. Wirelength-Driven Global Placement

A circuit can be represented by a hypergraph $H = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the set of cells, and $E = \{e_1, e_2, \ldots, e_r\}$ is the set of nets. The placement area is divided into a uniform grid of $m \times n$ bins $\mathbb{B}$. Then the wirelength-driven global placement problem can be formulated as

$$\begin{aligned} \min_{x,y} \quad & \sum_{e \in E} WL(e; x, y) \\ \text{s.t.} \quad & D_b(x, y) \le D_b, \quad \forall b \in \mathbb{B}, \end{aligned}$$

where $WL(e; x, y)$ is net e's wirelength, $D_b(x, y)$ is the density of bin b, and D_b represents the maximum allowable density of the bin.

The wirelength function $W(e; x, y)$ is often defined as the half-perimeter wirelength (HPWL). Direct minimization of HPWL in global placement is challenging due to its non-smoothness. In this work, we adopt the weighted average (WA) wirelength model [14] to approximate HPWL, the WA model in the x-direction is:

$$WA_{e_x} = \left(\sum_{i \in e} x_i e^{\frac{x_i}{\gamma}} \Big/ \sum_{i \in e} e^{\frac{x_i}{\gamma}} \right) - \left(\sum_{i \in e} x_i e^{-\frac{x_i}{\gamma}} \Big/ \sum_{i \in e} e^{-\frac{x_i}{\gamma}} \right).$$

The WA model iterates through the horizontal and vertical coordinates of each pin i in net e, producing a smooth wirelength function to approximate the HPWL, with γ controlling the degree of smoothness.

Electrostatics-based density [15] has inspired many state-of-the-art global placement algorithms [3, 16, 17]. This approach models each cell as a charged particle interacting within an electric field, where the field strength reflects cell density. High-density region creates a repulsive force, pushing cells toward lower-density region to achieve a balanced placement. Specifically, based on Gauss's law, ePlace [15] utilizes Poisson's equation with boundary and compatibility conditions to address cell density in placement optimization:

$$\begin{cases} \nabla \cdot \nabla \psi(x, y) = -\rho(x, y), & (x, y) \in R, \\ \mathbf{n} \cdot \nabla \psi(x, y) = 0, & (x, y) \in \partial R, \\ \iint_R \rho(x, y) = \iint_R \psi(x, y) = 0, \end{cases} \tag{1}$$

where R is the rectangular placement area, ∂R denotes its boundary, $\mathbf{n}$ is the outward normal vector of the region, ρ is the charge density, and ψ stands for the electric potential. The numerical solution of Poisson's Equation (1) can be found in [15].

Then, the wirelength-driven analytical global placement problem can be formulated as [3, 15, 16]:

$$\min_{x,y} \sum_{e \in E} WA_e(x, y) + \lambda_1 D(x, y), \tag{2}$$

where $D(x, y)$ is the electrostatic-based density penalty term, calculated by: $D(x, y) = \frac{1}{2} \sum_{i \in V} A_i \psi_i$, where A_i is the area of cell i, ψ_i is the electric potential of cell i. λ_1 is the weighting factor to balance the total wirelength and density penalty. Calculation of the electrostatic-based density gradient $\nabla D(x, y)$ can be found in [15].

B. Analytical Routability-Driven Global Placement

To address routability in global placement, we first introduce the definition of routing congestion. The 3D routing region is divided into a set of global routing cells (G-cells), denoted by $G_R \in \mathbb{R}^{R_r \times R_c \times L}$, here, R_r and R_c represent the row and column numbers of the routing grid, while L denotes the number of routing layers. Based on the G-cells, a congestion map C can be defined as [8]:

$$C_{m,n} = \max \left(\frac{\sum_{l=1}^{L} Dmd_{m,n,l}}{\sum_{l=1}^{L} Cap_{m,n,l}} - 1, 0 \right), \quad \forall (m, n) \in \mathbb{R}^{R_r \times R_c}, \tag{3}$$

where $C_{m,n}$ is the congestion value of the m-th row and n-th column of congestion map C. Dmd represents the routing demand map, which represents the total of wire demand and via demand, Cap refers to the routing capacity map.

To facilitate the establishment of the congestion model, we create a mapping of G-cells onto the 2D plane, which has the same dimension as the bins division. Let $Dmd_{m,n} = \sum_{l=1}^{L} Dmd_{m,n,l}$ and $Cap_{m,n} = \sum_{l=1}^{L} Cap_{m,n,l}$ represent the total routing demand and the total routing capacity of all layers' G-cells located at the m-th row and n-th column, respectively. Hereafter, any mention of G-cell refers to the 2D mapping. For routability-driven global placement, constraints are imposed into Equation (2) to ensure that the routing demand does not exceed its given routing capacity, and the routability-driven global placement problem is:

$$\begin{aligned} \min_{x,y} \quad & \sum_{e \in E} WA_e(x, y) + \lambda_1 D(x, y) \\ \text{s.t.} \quad & Dmd_{m,n} \le Cap_{m,n}, \quad \forall (m, n) \in \mathbb{R}^{R_r \times R_c}. \end{aligned} \tag{4}$$

Meeting the routing congestion constraints of Equation (4) is challenging, as routing demands frequently exceed available resources in certain regions, making routing congestion hard to alleviate. To handle this issue, we adopt the electric field model of ePlace [15], which enables moving cells away from high-density placement areas.

Analogous to [15], we apply the electric field model for routing congestion mitigation, using Poisson's equation to obtain an analytical solution for guiding the movement of cells away from high-congestion areas. To this aim, we take ρ in Equation (1) as

$$\rho_{m,n} = \frac{Dmd_{m,n}}{Cap_{m,n}}, \quad \forall (m, n) \in \mathbb{R}^{R_r \times R_c},$$

where $Dmd_{m,n}$ and $Cap_{m,n}$ are obtained by invoking the global router of [18]. Then, we solve Poisson's Equation (1) to obtain a solution, which is the electric potential for congestion, and calculate the related electrostatic-based congestion penalty term $C(x,y)$ by: $C(x,y) = \frac{1}{2}\sum_{i\in V'} A_i\psi_i$, where V' represents the set of multi-pin cells and virtual cells (for moving nets, detailed in Subsection III-A), A_i and ψ_i denote the area and electric potential of cell i, respectively. Then similar to Equation (2), the analytical routability-driven placement model can be established:

$$\min_{x,y} \quad \sum_{e\in E} WA_e(x,y) + \lambda_1 D(x,y) + \lambda_2 C(x,y), \tag{5}$$

where λ_2 is the weight for the congestion penalty term, which will be elaborated on in Subsection III-A. **The electrostatic-based congestion gradient $\nabla C(x,y)$ is computed with the method in [15]. However, it is not used directly as that of cell density, since the aim is mainly for moving nets to mitigate global routing congestion.** Updating $\nabla C(x,y)$ is detailed in Subsection III-A.

III. The Proposed Algorithm

This section details our proposed analytical routability-driven global placement framework for problem (5). Fig. 2 outlines the overall flow of the proposed algorithm. The red boxes indicate the technical points that we contribute.

In Fig. 2, we select power and ground (PG) rails based on the positions of macro blocks to prepare for the subsequent pin accessibility optimization. Then, we use the wirelength-diven global placer Xplace [16] to obtain an initial global placement result. Next, we proceed to the routability-driven global placement stage. We adopt the GPU-accelerated 3D Z-shape routing algorithm [18] to estimate routing congestion and generate a congestion map. This map is used for adjusting the cell inflation rate through our momentum-based cell inflation technique, and then we dynamically modify the local density using our pin accessibility adjustment technique. Next, the updated placement information is incorporated into the objective function of problem (5), which is subsequently solved using the Nesterov solver proposed in [15]. The routability-driven global placement is repeated until the $C(x,y)$ no longer decreases or the given number of iterations is reached. Lastly, the routability-driven legalization and detailed placement of Xplace-route [8] is adopted for the final placement.

In the following content of this section, we first introduce the congestion gradient update for mitigating global routing congestion in Subsection III-A. Then in Subsection III-B, we present the momentum-based cell inflation technique for mitigating local routing congestion. Finally, Subsection III-C discusses the pin accessibility optimization during the global placement stage.

A. Congestion Gradient Update

This subsection presents the congestion gradient update technique for the problem (5). As described earlier, congestion is generally categorized into local routing congestion, due to an excessive number of cells within an area covered by a routing grid, and global routing congestion, caused by high wiring density within a routing grid.

After adopting the GPU-accelerated 3D Z-shape routing algorithm [18] to estimate routing congestion and generate a congestion map, we can obtain the congestion penalty and cells' congestion gradient in problem (5). However, it would be insufficient to move the cells directly according to the congestion gradient generated by the electric field model as we do for the cell density, because a high congestion value is not only due to the excessive clustering of cells (local routing congestion, handled in Subsection III-B), but may also result

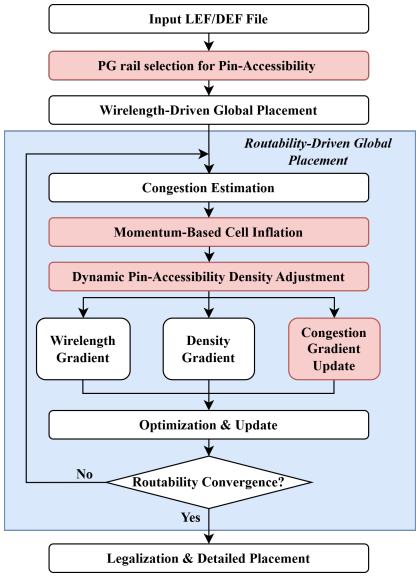


Fig. 2: Overview of our routability-driven global placement flow.

from a large number of nets covering the region (global routing congestion), as demonstrated in Fig. 1. In this subsection, we propose two strategies to mitigate global routing congestion.

Initially, We set the congestion gradient of all cells to 0, and then adjust them via the following approach. Firstly we update the congestion gradient for the two cells connected by a two-pin net, aiming to move the net. Then we update the congestion gradient for selected multi-pin cells.

1) Congestion Gradient Update for Two-Pin Net Moving: To mitigate global routing congestion, we propose a congestion gradient update algorithm for two-pin net moving. Since there is no routing information in the placement stage, the electric field force model only affects cells within the placement region, while generally ignoring net information. Therefore, to optimize routability for the nets, we introduce a virtual cell in every two-pin net, serving as a pivot point to guide the movement of the net.

For a two-pin net e with pins p_1 and p_2, as shown in Fig. 3(a), (x_1,y_1) and (x_2,y_2) are the positions of p_1 and p_2, respectively. Let c_1 and c_2 represent the cells to which these two pins belong. By connecting the pins p_1 and p_2, we get the segment $\widehat{p_1p_2}$. Then we determine the location of a virtual cell c_v with the same size as a standard cell for the two-pin net in a congested region.

Let l_x and l_y denote the width and height of a G-cell, respectively. We calculate (x_v,y_v) for c_v by the following equations:

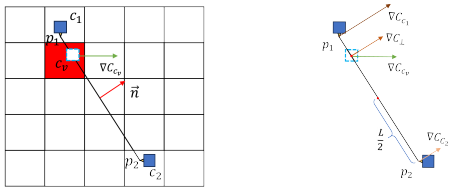
$$k = \max\left(\left\lfloor\frac{|x_1-x_2|}{l_x}\right\rfloor, \left\lfloor\frac{|y_1-y_2|}{l_y}\right\rfloor\right), \tag{6}$$

$$(x_i,y_i) = (x_1,y_1) + \frac{i}{k+1}(x_2-x_1, y_2-y_1), \quad i\in[1,k], \tag{7}$$

$$(x_v,y_v) = \arg\max_{(x_i,y_i)} C_{m,n}, \quad m = \left\lfloor\frac{x_i}{l_x}\right\rfloor, n = \left\lfloor\frac{y_i}{l_y}\right\rfloor. \tag{8}$$

In Equation (6), we approximate the number k of G-cells traversed by the segment $\widehat{p_1p_2}$ by taking the maximum value between $\left\lfloor\frac{|x_1-x_2|}{l_x}\right\rfloor$ and $\left\lfloor\frac{|y_1-y_2|}{l_y}\right\rfloor$. Next we proportionally select k candidate points (x_i,y_i) on the segment in Equation (7). Then, we substitute the candidate points into the congestion map to select the candidate point with the maximum congestion value as (x_v,y_v) in Equation (8), where $C_{m,n}$ is given in Equation (3). At (x_v,y_v), a virtual cell c_v of the same size as a standard cell is created (shown in Fig. 3(a)).

Subsequently, we utilize the virtual cell c_v to calculate the con-



(a) (b)

Fig. 3: Congestion gradient for cells on a two-pin net. (a): c_1 and c_2 are two cells connected by the net with pins p_1 and p_2. $\vec{n}$ is the unit normal direction of the segment $\widehat{p_1p_2}$, oriented to form an acute angle with ∇C_{c_v}. The red region indicates the most congested region, where a virtual cell c_v is generated with the congestion gradient ∇C_{c_v}. (b): $\nabla C_\perp$ is ∇C_{c_v}'s projection onto $\vec{n}$. Then, ∇C_{c_1} and ∇C_{c_2} are updated so that they have the same direction as $\nabla C_\perp$.

gestion gradients for the cells c_1 and c_2, ensuring their connection net can be moved away from the high congestion region. Let $\vec{n}$ be the unit normal direction of the segment $\widehat{p_1p_2}$, and oriented to the direction with an acute angle with ∇C_{c_v}. Here, ∇C_{c_v} is the congestion electric field gradient experienced by the virtual cell c_v. We project ∇C_{c_v} onto $\vec{n}$ and get $\nabla C_\perp$, as shown in Fig. 3(b). This is the most efficient direction for the connecting net to leave the congested region. The final congestion gradient for the cells c_1 and c_2 is calculated as follows:

$$d_{iv} = \sqrt{(x_v - x_i)^2 + (y_v - y_i)^2},$$
$$\nabla C_{c_i} = \frac{L}{2d_{iv}} \nabla C_\perp, \tag{9}$$

where d_{iv} is the Euclidean distance between pin p_i and virtual cell c_v. $L = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ is the segment length of $\widehat{p_1p_2}$.

In Equation (9), we use $\frac{L}{2}$ as a reference to control a cell's congestion gradient based on the distance between the cell and the virtual cell. A cell near the congested region receives a larger congestion gradient, while a cell far away from the congested region receives a smaller congestion gradient. This allows the cells to move away from highly congested routing regions more efficiently.

Algorithm 1 summarizes the pseudocode for the proposed net-driven cell congestion gradient calculation. First, we identify the maximum congestion point (x_v, y_v) along the segment connecting two pins (line 1) and create a virtual cell c_v (line 2). The congestion gradient ∇C_{c_v} of c_v is then obtained using the electric field model (line 3). Next, the straight-line distance L between the two pins is calculated, and a perpendicular vector $\vec{n}$ that forms an acute angle with the gradient ∇C_{c_v} is derived (lines 4-5). Finally, the congestion gradient of c_1 and c_2 is computed (lines 6-9).

2) Congestion Gradient Update for Multi-pin Cell: A multi-pin cell is connected to several other cells. A large number of nets are connected to multi-pin cells, and these cells often increase global routing congestion in areas where routing resources are already limited. To address this issue, we compute by Equation (1) the congestion gradient of **selected multi-pin cells**, and then update the gradient for moving these cells away from congested regions, thereby alleviating global routing congestion. The specific approach is presented as Algorithm 2.

In lines 3-6 of Algorithm 2, Algorithm 1 is executed to obtain the congestion gradient for each cell of the two-pin net. Lines 7-15 perform congestion gradient adjustments for selected multi-pin cells. Specifically, when the number of pins of a cell i is greater than the average number of pins $\bar{n}$ (line 1) and the congestion value of the G-cell under which the cell i's center position is located exceeds 0.7 (line 11), we adopt the congestion gradient of the cell by Equation (1) (line 12). Finally, the congestion gradient of all cells in net e is obtained by combining the congestion gradients produced by the two approaches (lines 16).

Algorithm 1: Cell Congestion Gradient Update for Two-Pin Net Moving

Input: Cells c_1, c_2, Pin positions p_1, p_2, Congestion map C;
Output: $CGrad^{net}$: Congestion gradient of cells c_1 and c_2.
1 $(x_v, y_v) \leftarrow$ find max congestion position by Equation (8);
2 $c_v \leftarrow$ create a virtual cell;
3 $\nabla C_{c_v} \leftarrow$ congestion gradient of c_v by Equation (1);
4 $L \leftarrow$ calculate distance between p_1 and p_2;
5 $\vec{n} \leftarrow$ the unit vector perpendicular to the segment connecting p_1 and p_2 and forming an acute angle with ∇C_{c_v};
6 **for** $i \in \{1, 2\}$ **do**
7 $\quad d_{iv} \leftarrow$ calculate distance between p_i and c_v;
8 $\quad \nabla C_\perp \leftarrow (\nabla C_{c_v} \cdot \vec{n}) \cdot \vec{n}$;
9 $\quad CGrad^{net}_{c_i} \leftarrow \frac{L}{2d_{iv}} \nabla C_\perp$;
10 **end**
11 **return** $CGrad^{net}$;

With Algorithm 2, the congestion gradient of all cells is obtained by traversing all nets of a circuit, and then the congestion gradient of each cell is the superposition of the congestion gradients returned to the cell by each of these nets.

Algorithm 2: Congestion Gradient Update for Cells on a Net

Input: Net e, the set of cells c_i of the net, pins $p_{i,j}$ of cell i, congestion map C;
Output: $CGrad$: Congestion gradient of cells c_i in net e.
1 $\bar{n} \leftarrow$ average number of pins of all cells;
2 $k \leftarrow$ number of pins in the net e;
3 $CGrad^{net} \leftarrow 0$;
4 **if** $k = 2$ **then**
5 $\quad$ Use Algorithm 1 to obtain $CGrad^{net}$;
6 **end**
7 $CGrad^{cell} \leftarrow 0$;
8 **foreach** c_i **do**
9 $\quad n \leftarrow$ the number of pins of cell c_i;
10 $\quad C_{c_i} \leftarrow$ the congestion value of the G-cell under which the cell c_i's center position is located;
11 $\quad$ **if** $n > \bar{n}$ *and* $C_{c_i} > 0.7$ **then**
12 $\quad\quad CGrad^{cell}_{c_i} \leftarrow$ congestion gradient of c_i by Equation (1);
13 $\quad$ **end**
14 $\quad CGrad^{cell} \leftarrow CGrad^{cell}_{c_i}$;
15 **end**
16 $CGrad \leftarrow CGrad^{net} + CGrad^{cell}$;
17 **return** $CGrad$;

3) The Weight of Congestion Penalty Term: In Section II-B, we established the analytical routability-driven global placement model (5), in which λ_2 is the congestion penalty weight. Here, it is set as:

$$\lambda_2 = \frac{2N_C}{N} \cdot \frac{||\nabla W(x,y)||_1}{||\nabla C(x,y)||_1}. \tag{10}$$

In Equation (10), λ_2 is obtained by the ratio of the L_1 norm of the wirelength gradient $\nabla W(x,y)$ to that of the congestion gradient $\nabla C(x,y)$, multiplied by an adjustment coefficient. Specifically, N_C is the total number of cells in congested regions, and N is the total number of cells. Using Equation (10), λ_2 is adjusted based on the current overall congestion during the iteration. When the number of cells in a congested area is big, λ_2 will be large, prioritizing con-

gestion optimization. Conversely, if congestion is low, λ_2 decreases, allowing for a greater emphasis on wirelength optimization.

B. Momentum-Based Cell Inflation

This subsection presents the momentum-based cell inflation technique for mitigating local routing congestion. Most cell inflation techniques primarily adjust cell's inflation rate based on the congestion information from the current iteration [3, 5]. They encounter the issue of moving a cell back into the original congested area when the cell is transited from the inflated state to the original state or over-inflated during the iteration. To address this issue, we incorporate the momentum technique into calculating the inflation rate. Our approach considers the historical inflation rate of a cell similar to NTUplace4dr [9] and a cell deflation mechanism to utilize available resources better. Once the inflation rate is determined, the cell size is proportionally inflated during density calculation, increasing the density of congested region. This provides a higher gradient to the cells in congested regions, prompting them to spread out the regions.

Specifically, the core formula is as follows:

$$\begin{aligned} r_i^t &= \min\left\{\max\left(r_i^{t-1}+\Delta r_i^t, r_{\min}\right), r_{\max}\right\}, \\ \Delta r_i^t &= \alpha \Delta r_i^{t-1} + (1-\alpha) s_i^t, \\ s_i^t &= \delta_i^t C_i^t, \end{aligned} \tag{11}$$

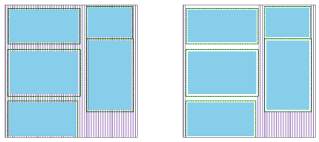
where r_i^t represents the inflation rate of cell i at the t-th ($t = 1, 2, 3, \dots$) iteration of cell inflation, and $r_i^0 = 1$. The r_{min}, $r_{\max}$, and α are user-specified values that provide an appropriate inflation rate for each cell, which are set to 0.9, 2.0, and 0.4 in this work respectively. Δr_i^t denotes the change value in the inflation rate, which is mainly affected by the inflation rate of all previous iterations, and $\Delta r_i^1 = C_i^1$. C_i^t is the congestion value of the G-cell under which cell i's center position is located at the t-th iteration of cell inflation. Moreover, in Equation (11), s_i^t is the congestion correction factor, which is determined by a decision parameter δ_i^t and the congestion value. δ_i^t is calculated by:

$$\delta_i^t = \begin{cases} -\left|\frac{C_i^{t-1}\bar{C}^t - C_i^t \bar{C}^{t-1}}{\bar{C}^{t-1}\bar{C}^t}\right|, & \text{if } C_i^t < \bar{C}^t \text{ and } C_i^{t-1} > \bar{C}^{t-1}; \\ 1, & \text{otherwise.} \end{cases} \tag{12}$$

In Equation (12), $\bar{C}^t$ is the average congestion value of all G-cells at the t-th cell inflation. If δ_i^t remains constant to 1, r_i^t will be monotonically increasing, meaning the inflation ratio of the cell will continue expanding even after it has been moved away from the congested region. This is unreasonable, so we introduce a cell deflation mechanism. When the congestion value C_i^t of cell i at the t-th iteration is lower than the average congestion value $\bar{C}^t$ of the t-th iteration, and the congestion value C_i^{t-1} of the $(t-1)$-th iteration is higher than the average value $\bar{C}^{t-1}$, we think that the cell has been moved away from a high congestion area to a low congestion area. At this time, a negative congestion correction factor is applied to the cell. We quantify the degree of cell deflation through the following approach. We calculate the ratio of $\bar{C}^t - C_i^t$ to the average congestion value $\bar{C}^t$, and then add the term $\frac{C_i^{t-1}-\bar{C}^{t-1}}{\bar{C}^{t-1}}$ to $\frac{\bar{C}^t - C_i^t}{\bar{C}^t}$ to get the absolute value $\left|\frac{C_i^{t-1}\bar{C}^t - C_i^t\bar{C}^{t-1}}{\bar{C}^{t-1}\bar{C}^t}\right|$, which represents the deflation strength of cell i, as shown in Equation (12).

C. Dynamic Pin-Accessibility Density Optimization

This subsection presents the dynamic pin-accessibility density optimization technique. Pin-accessibility is closely related to routability. Most work performs pin-accessibility optimization at detailed placement, in which local perturbations such as cell shifting, flipping, and swapping are used to optimize pin-accessibility [11, 12, 13]. These methods limit exploration space for pin-accessibility optimization. To improve solution quality, we consider pin accessibility optimization during global placement. As mentioned in Xplace-Route [8], during the placement process, cells on the M1 layer may move beneath the power grid rails on the M2 layer, making it difficult to connect the pins on these cells due to the constrained routing resources on M1. Xplace-Route [8] only adjusts cell density around PG rails before placement, while we focus on addressing pin accessibility throughout the entire global placement process. Our pin-accessibility algorithm is mainly divided into the following two steps:



(a) (b)

Fig. 4: PG rail selection for density adjustment in design matrix_mult_a [1], where the purple lines represent PG rails, the blue areas denote macro blocks and the green dashed line represents the expanded macro's bounding box. (a) presents all PG rails before PG rail selection. (b) gives the selected PG rails, the cell density of the related area will be adjusted.

1) PG rail selection for Pin-Accessibility: We find experimentally that, indiscriminately increasing the cell density of all regions covered by PG rails on the M2 layer is unreasonable. For example, the narrow region between macros are already highly congested, increasing the cell density around PG rails in the region would hinder cell spreading. Therefore, we perform a pre-processing step to identify regions where density can be adjusted effectively.

Specifically, we expand each macro's bounding box by 10%, and then project all bounding boxes and the PG rails onto the 2D plane, as shown in Fig. 4(a). These PG rails will be cut by the bounding boxes into some new PG rails. Then, we choose those PG rails whose length are at least 0.2 times the placement region's width (or height) in the horizontal (or vertical) direction, as shown in Fig. 4(b). The chosen PG rails are used for the next step of cell density adjustment.

2) Dynamic Pin-Accessibility Density Adjustment: Since multiple factors can affect congestion, it is impractical to focus only on the PG rails and ignore other factors. To reserve space for optimizing other factors and improve pin accessibility, we propose a precise density adjustment strategy concerning PG rails, aiming at enlarging density in congestion regions covered by the selected PG rails.

First, we evaluate the routing congestion on the area of the selected PG rails using the global router of [18]. Since we predefine the G-cell and bin to have the same dimension in Subsection II-B, we can map the congestion value of each G-cell to the corresponding bin at the same position. Then, **we selectively increase density only in highly congested bins**. This density adjustment dynamically adapts to congestion changes, preventing density increases across all selected PG rail areas and reserving space for further optimization.

Dynamic pin-accessibility-aware density primarily contains the original density D^{ori} and the M2-layer PG rails density D^{PG} (defined in Equation (13) and (14)). Cells located in the congested region covered by the selected PG rails will get a greater density gradient. This causes the cells at the region covered by congested PG rails to diffuse, reserving connection space for the pins in this area. The modified density D_b in bin b is: $D_b = D_b^{\text{ori}} + D_b^{PG}$, where

$$D_b^{\text{ori}} = \frac{1}{A_b}\sum_{i\in V} A_{c_i \cap b}, \qquad \forall b \in B, \tag{13}$$

$$D_b^{PG} = \frac{\eta_b(1+C_b)}{A_b}\sum_{i\in V_{PG}} A_{PG_i \cap b}, \qquad \forall b \in B. \tag{14}$$

TABLE I: Average results on the ISPD 2015 Contest Benchmarks [1]. The benchmarks removing the fence region constraints are marked by †.

Design	Xplace [16]					Xplace-Route [8]					Ours				
	DRWL/um	#DRVias	#DRVs	PT/s	RT/s	DRWL/um	#DRVias	#DRVs	PT/s	RT/s	DRWL/um	#DRVias	#DRVs	PT/s	RT/s
des_perf_1	1452606	569989	24977	5	869	1446525	566434	12768	7	657	1429056	566581	**10652**	30	568
des_perf_a†	2344941	560801	29875	5	180	2474885	578102	40701	20	200	2360122	569940	**10094**	20	3430
des_perf_b†	1817040	554677	19580	5	648	1807464	541944	1557	7	123	1783673	541130	**849**	21	89
edit_dist_a†	5670681	1006803	405858	6	748	5745638	1015369	424887	23	779	5737250	1034387	**359904**	19	801
fft_1	516059	186603	9249	3	415	515345	186516	4077	11	293	513993	185309	**3661**	26	299
fft_2	598828	187925	9334	3	117	627383	190944	1197	10	248	612371	190620	**1070**	14	328
fft_a	1090612	193181	5650	3	380	1144134	192291	925	11	296	1133734	192225	**790**	9	273
fft_b	1256346	205052	33875	3	191	1318002	217100	**14603**	12	255	1310187	216838	14710	9	263
matrix_mult_1	2708706	809962	80816	6	378	2656679	826886	15371	21	2236	2638437	820336	**10717**	25	2180
matrix_mult_2	2719259	840175	72311	6	419	2681555	860246	14422	22	2479	2676434	853057	**10651**	26	2975
matrix_mult_a	3892797	865702	34618	7	1590	3928555	849064	9380	10	2056	3938218	849468	**9304**	18	1704
matrix_mult_b†	3650328	790195	68415	6	338	3655611	783526	47964	8	374	3680673	789855	**37698**	18	323
matrix_mult_c†	3713674	813859	34226	6	1559	3681892	793568	9119	8	1205	3679849	795182	**8907**	19	686
pci_bridge32_a†	642272	148390	6553	3	631	656083	146580	4288	4	803	649379	146588	**3660**	9	1035
pci_bridge32_b†	978132	149169	2828	3	84	1008959	148146	339	4	53	998056	148052	**198**	9	38
superblue11_a†	40395036	5670612	866	33	1836	40373818	5716757	1047	49	1901	40209334	5652221	**669**	134	1670
superblue12	42759984	10825565	3276003	45	7513	43155488	10531913	22263	279	6091	49383559	12779668	**15113**	260	6796
superblue14	28028276	4330996	344	23	2555	28067336	4332033	367	43	2601	28097546	4289310	**310**	147	1511
superblue16_a†	31543711	4648757	4486	27	4163	31597720	4668672	4361	50	4303	31267452	4653129	**3387**	157	4004
superblue19	20830885	3637145	10097	18	2574	20849011	3618684	**6599**	54	1936	21126469	3647126	8280	109	1781
Avg. Ratio	1.00	1.00	5.00	0.25	1.37	1.00	0.99	1.40	0.63	1.07	1.00	1.00	1.00	1.00	1.00

* The DRV result of Xplace on superblue12 is not good, we exclude it from calculating its mean ratio to ensure a fair comparison of average ratios.
* Routing runtime may not reflect placement routability, as Innovus will end the detailed routing early if the placement has poor routability [8].

$$\eta_b = \begin{cases} 1, & \text{if } C_b > \bar{C}; \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

After adjusting the density map, we can calculate the density penalty in problem (5).

In the above Equations (13) and (14), $A_{c_i \cap b}$ represents the overlapping area between cell c_i and bin b, $A_{PG_i \cap b}$ represents the overlapping area between PG rail PG_i and bin b, and A_b represents the area of bin b. V is the set of cells, and V_{PG} is the set of selected PG rails during the PG rail selection process. C_b denotes the congestion value mapped at bin b, and $\bar{C}$ denotes the average congestion value across all bins. The PG rail density D_b^{PG} of bin b is controlled by the congestion value of bin b. As in Equation (15), if the congestion value C_b is higher than the average congestion value $\bar{C}$, the density increases with a weight of $1 + C_b$. This approach selectively increases the density in highly congested areas rather than adjusting the density in all regions where the selected PG rails cover, leaving additional space for other optimizations.

IV. Experimental Results

Our proposed framework is developed with PyTorch and CUDA, and deployed on the wirelength-driven global placer Xplace [16]. The experiments are performed on a Linux server with a 2.90GHz Intel Xeon CPU and a single Nvidia A800 GPU. The ISPD 2015 contest benchmarks [1] without fence-region constraints are used for the comprehensive evaluation and comparison.

We compare our analytical routability-driven global placement algorithm with Xplace [8] (without routability) and its routability-enhanced version Xplace-Route [8], a leading routability-driven placer. To ensure testing accuracy and maintaining the fairness of comparisons, the same version of the commercial tool Innovus [19] is used for global and detailed routing of the placement result by each compared placer. Specifically, we run each compared placer first and record the placement runtime (PT/s). Next, the placement result is input into Innovus to complete the global and detailed routing, and the post-routing results are recorded to measure routability. These results include detailed routing wirelength (DRWL/um), the number of DR vias (#DRVias), and the number of detailed routing violations (#DRVs). Moreover, we report the total routing runtime (RT/s). Table I lists the experimental results.

Table I shows that, our algorithm gains a significant improvement in routability compared to Xplace [8] and Xplace-Route [8]. Specifically, our algorithm achieves a 400% reduction in #DRVs compared to Xplace, meanwhile maintaining the same average DRWL and #DRVias. Compared to the leading routability-driven placer, Xplace-Route, our algorithm achieves a 40% reduction in #DRVs, while maintaining almost the same average DRWL and #DRVias.

TABLE II: Ablation Experiment.

Methods			DRWL/um	#DRVias	#DRVs
MCI	DC	DPA	Avg. Ratio	Avg. Ratio	Avg. Ratio
-	-	-	1.00	0.99	1.40
✓	-	-	0.99	0.99	1.27
✓	✓	-	1.00	1.00	1.12
✓	✓	✓	1.00	1.00	1.00

For the runtime, our algorithm spends $4.00\times$ and $1.59\times$ the total placement runtimes compared to Xplace and Xplace-Route, respectively. However, our algorithm's total global and detailed routing runtime is sped up by $1.37\times$ and $1.07\times$, compared to Xplace and Xplace-Route. These comparisons show that while our algorithm spends more time in placement to integrate routability techniques and reduce congestion, it ultimately reduces routing costs.

Further, we conduct ablation experiments to verify the effectiveness of our three techniques for alleviating routing congestion, with results shown in Table II. MCI refers to our algorithm with momentum-based cell inflation, DC to the algorithm with a differentiable congestion function, and DPA to the one with dynamic pin accessibility density adjustment. From the table, we observe that #DRVs decreases more as more techniques are applied, while the average DRWL and #DRVias remain nearly unchanged. This demonstrates that each technique improves routability without degrading wirelength or via count.

V. Conclusion

We have proposed an analytical framework for addressing routability in global placement. In this framework, a differentiable congestion function is developed by considering global routing congestion. Moreover, virtual cells in two-pin nets and selected multi-pin cells are introduced for guiding nets moving to mitigate global routing congestion. Furthermore, momentum-based cell inflation and dynamic pin-accessibility density optimization techniques are proposed for adjusting cell density to mitigate local routing congestion. Experimental results show that, our algorithm can effectively alleviate local and global routing congestion, improve routability significantly meanwhile maintaining comparable other metrics, compared to the leading routability-driven placer, Xplace-Route [8].

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62174033, the National Key Research and Development Program of China under Grant 2023YFA1011302, the Major Key Project of PCL (No. PCL2023A03), and the Natural Science Foundation of Fujian Province under Grants (No. 2024J09045). We thank Dr. Lixin Liu for helpful discussions and thank the authors of Xplace [8, 16] for their great contribution of the placer.

REFERENCES

[1] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. ISPD*, 2015, pp. 157–164.

[2] J.-M. Lin, C.-W. Huang, L.-C. Zane, M.-C. Tsai, C.-L. Lin, and C.-F. Tsai, "Routability-driven global placer target on removing global and local congestion for VLSI designs," in *Proc. ICCAD*, 2021, pp. 1–8.

[3] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. DAC*, 2019, pp. 1–6.

[4] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proc. DATE*, 2021, pp. 1821–1824.

[5] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, vol. 38, no. 9, pp. 1717–1730, 2018.

[6] M. k Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "Ntuplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE TCAD*, vol. 33, no. 12, pp. 1914–1927, 2014.

[7] J.-M. Lin, Y.-Y. Chang, and W.-L. Huang, "Timing-driven analytical placement according to expected cell distribution range," in *Proc. ISPD*, 2024, pp. 177–184.

[8] L. Liu, B. Fu, S. Lin, J. Liu, M. D. F. Wong, and E. F. Y. Young, "Xplace: An extremely fast and extensible placement framework," *IEEE TCAD*, vol. 43, no. 6, pp. 1872–1885, 2024.

[9] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "Ntuplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE TCAD*, vol. 37, no. 3, pp. 669–681, 2017.

[10] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proc. DATE*, 2007, pp. 1–6.

[11] S. Kim and T. kim, "Pin accessibility-driven placement optimization with accurate and comprehensive prediction model," in *Proc. DATE*, 2022, pp. 778–783.

[12] J. Seo, J. Jung, S. Kim, and Y. Shin, "Pin accessibility-driven cell layout redesign and placement optimization," in *Proc. DAC*, 2017, pp. 1–6.

[13] Y. Ding, C. Chu, and W.-K. Mak, "Pin accessibility-driven detailed placement refinement," in *Proc. ISPD*, 2017, pp. 133–140.

[14] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *Proc. DAC*, 2011, pp. 664–669.

[15] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "eplace: Electrostatics-based placement using Fast Fourier Transform and Nesterov's method," *ACM TODAES*, vol. 20, no. 2, pp. 1–34, 2015.

[16] L. Liu, B. Fu, M. D. F. Wong, and E. F. Y. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proc. DAC*, 2022, pp. 1309–1314.

[17] W. Zhu, Z. Huang, J. Chen, and Y. C, "Analytical solution of Poisson's equation and its application to VLSI global placement," in *Proc. ICCAD*, 2018, pp. 1–8.

[18] S. Lin and M. D. F. Wong, "Superfast full-scale GPU-accelerated global routing," in *Proc. ICCAD*, 2022, pp. 1–8.

[19] Cadence Design Systems, "Cadence Innovus Implementation System," n.d. [Online]. Available: https://www.cadence.com.

PICK: An SRAM-based Processing-in-Memory Accelerator for K-Nearest-Neighbor Search in Point Clouds

Chen Nie[1,2], Chao Jiang[3], Liming Xiao[3], Weifeng Zhang[3], Zhezhi He[1,2]
[1]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Intelligent Computing Research Group (ICRG), Shanghai Jiao Tong University, China [3]Lenovo Research, Beijing, China
{chen.nie, zhezhi.he}@sjtu.edu.cn {jiangchao13, xiaolm, weifengz}@lenovo.com

***Abstract*—K-nearest neighbor (kNN) search is a fundamental operation in various point cloud applications, such as autonomous driving. However, the heavy computational intensity and memory demands of kNN search pose significant challenges for efficient implementation, especially in resource-constrained scenarios. To address these challenges, we propose PICK, a processing-in-memory (PIM) architecture designed to accelerate kNN search in point cloud applications. PICK leverages bit-serial-based PIM (BS-PIM) and customized circuits to efficiently handle key operations of kNN search: distance calculation and top-k selection. The run-time off-chip access is eliminated thanks to the large on-chip memory. For distance calculation, we introduce a bit-width clipping technique to reduce the latency of bit-serial execution with negligible accuracy degradation, providing flexible trade-offs between performance and precision. Besides, we propose a filtering-and-selection strategy that realizes approximately constant time complexity for arbitrary values of k. Furthermore, a two-stage pipeline is implemented to parallelize distance calculation and top-k search, effectively hiding latency and improving throughput. According to our experiments, PICK achieves $4.17\times$ speedup and a $4.42\times$ energy saving over the state-of-the-art design.**

***Index Terms*—Processing-in-Memory, SRAM, KNN Search**

I. Introduction

Recent advancements in artificial intelligence have significantly driven the adoption of three-dimensional (3D) visual data, commonly known as point clouds. Generated by specialized sensors such as LiDAR [1], point clouds consist of sets of points representing the surfaces and properties of surrounding objects and environments. Their versatility has led to extensive use across various domains, including autonomous driving [2], robotics [3], and virtual reality [4]. These applications demand high accuracy while adhering to stringent performance constraints, such as low latency for real-time processing and minimal power consumption for edge deployments.

To meet these demands, significant research has focused on optimizing the *k-nearest neighbor* (kNN) search, a core operation in point cloud processing. KNN search identifies k closest reference points to a query point and underpins critical algorithms such as simultaneous localization and mapping (SLAM) [5], point registration [6], object recognition [7], and scene segmentation [8]. KNN search comprises two computationally intensive operators: *distance calculation* and *top-k search*, requiring extensive arithmetic computations and comparisons. Prior works [9], [10] have revealed that *kNN search can account for up to 80% of the latency in real-world point cloud applications*, primarily due to the computationally intensive nature of distance calculation and top-k search. Therefore, optimizing kNN search is crucial to improving the performance of point cloud applications, particularly in real-time and resource-constrained environments.

Recently, several kNN-specific accelerators [10]–[17] have been developed to enhance computational efficiency and reduce latency in point cloud applications. Partition-based techniques, *e.g.* kd-trees in QuickNN [11], octrees in ParallelNN [12] and SimDiff [14], hashing [15], and voxelization [16], have been explored to minimize computation and memory access. However, these methods often compromise accuracy (*e.g.*, 80% in [12]) and introduce preprocessing overhead. JUNO [13] exploits sparsity and locality to accelerate approximate neighbor search but cannot ensure full accuracy. BitNN [10] uses bit-serial computation and early termination to enhance efficiency but suffers from excessive on-chip and off-chip memory access, which accounts for over 35% of total energy consumption. Despite these advancements, challenges remain in *further reducing computation latency and memory access while preserving satisfactory accuracy*.

In this work, we propose PICK, a novel processing-in-SRAM architecture tailored to efficiently accelerate kNN search for point cloud applications. PICK leverages bit-serial-based PIM (BS-PIM) and custom computing modules to optimize the fundamental operations of kNN search, namely, distance calculation and top-k search. The *in-situ* bit-serial computing approach of BS-PIM minimizes on-chip data movement and simplifies circuit design, enabling sufficient on-chip memory to eliminate runtime off-chip memory access entirely.

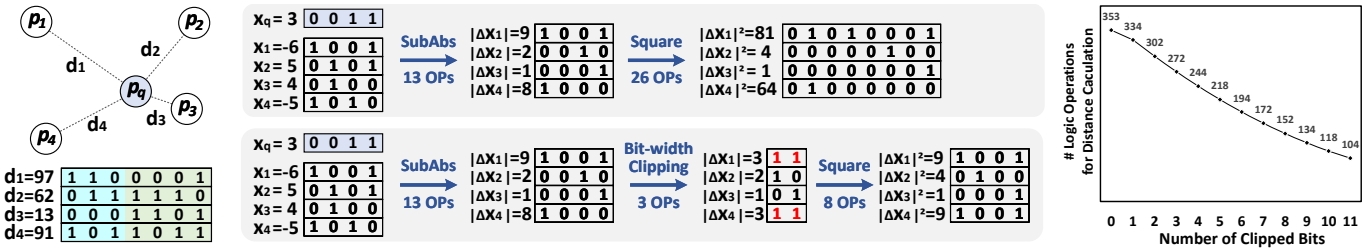
Furthermore, to address the high latency of bit-serial algorithms [18], we introduce a bit-width clipping technique that reduces latency with negligible accuracy degradation, allowing flexible performance-accuracy trade-offs for diverse use cases. For top-k search, we design an efficient filtering-and-selection strategy that achieves approximately constant time complexity for arbitrary k. Additionally, a two-stage pipeline parallelizes distance calculation and top-k search, hiding latency and improving throughput. Extensive evaluations on real-world point cloud datasets demonstrate that PICK achieves a $4.17\times$ speedup and $4.42\times$ energy savings compared to the state-of-the-art accelerator BitNN [10]. The key contributions of this work are summarized as:

- We propose PICK, the first PIM-based architecture to accelerate the kNN search in point clouds. The highly parallelized and in-situ computing approach greatly accelerates the computation and reduces both on-chip and off-chip data movements.
- We propose a bit-width clipping method to reduce the latency and energy of distance calculation, with negligible accuracy loss.
- We propose a filtering-and-selection-based search method to efficiently perform top-k search for arbitrary values of k, enabling optimal performance across a wide range of application scenarios.
- By comparison, PICK achieves $4.17\times$ speedup and $4.42\times$ energy saving over the state-of-the-art accelerator BitNN.

II. Background

Point Cloud The point cloud is an unordered collection of discrete points, represented as $\mathbb{P} = \{\boldsymbol{p}_i\}$, where $\boldsymbol{p}_i = (x_i, y_i, z_i)$ specifies the Cartesian coordinates of the i-th point in the set $\mathbb{P}$. Typically, point clouds are continuously captured by specialized devices (*e.g.*, LiDAR and 3D scanner), enabling the perception and reconstruction of surrounding environments. Modern LiDAR sensors [19] can generate

This work is partially supported by the National Natural Science Foundation of China under Grant 62102257 and National Key R&D Program of China under Grant 2022YFB4500200. Corresponding Author: Zhezhi He.



(a) Example of 1-NN (b) Distance calculation without (top) and with (bottom) bit-width clipping (c) Reduction of operations

Fig. 1: **Motivation for bit-width clipping**. (a) Example of 1-NN (kNN with $k = 1$) search between a query point $\boldsymbol{p}_q$ and four reference points $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$, where $\boldsymbol{p}_3$ is identified nearest with a distance d_3 which can be accurately represented using only 4 bits (in green). (b) Comparison of distance calculation along the x-dimension, shown without (top) and with (bottom) bit-width clipping. Bit-width clipping reduces the number of bit-serial logic operations (OPs) and memory usage, while maintaining $\boldsymbol{p}_3$ as the nearest point. (c) As more bits are clipped (initially 16-bit), the distance calculation involves fewer logic operations, leading to a reduction in computational latency.

up to 460k points per frame at a frame rate of 20Hz (*i.e.*, 50ms per frame), delivering rich spatial information that underpins a wide range of applications. Recent point cloud datasets [20]–[23] cover diverse scenarios, including objects, indoor and outdoor environments, and geographic landscapes, further broadening the utility and applicability of point cloud data.

K-nearest-neighbor (kNN) search. The kNN search is an extensively employed algorithm to identify the top-k closest reference points to a given query point, in point cloud applications. This process consists of two primary operations: *distance calculation* and *top-k search*. The distance calculation computes the squared Euclidean distance between a reference point $\boldsymbol{p}_i = (x_i, y_i, z_i)$ and a given query point $\boldsymbol{p}_\mathrm{q} = (x_\mathrm{q}, y_\mathrm{q}, z_\mathrm{q})$, which can be expressed as $d = |x_i - x_\mathrm{q}|^2 + |y_i - y_\mathrm{q}|^2 + |z_i - z_\mathrm{q}|^2$. Once the distance d_i is calculated for each reference point $\boldsymbol{p}_i$ relative to the query point $\boldsymbol{p}_\mathrm{q}$, the top-$k$ search is performed to identify the k points with the shortest distances and retrieve corresponding indices as $\{i_1, i_2, \ldots, i_k\}$.

Bit-Serial-based Processing-in-Memory. Processing-in-memory (PIM) architectures [24]–[27] have been extensively investigated to accelerate various applications, offering reduced data movements and substantial computing parallelism. Bit-serial-based PIM (BS-PIM) is an extension of PIM that incorporates bit-serial algorithms [18], aiming to support general-purpose tasks. In bit-serial algorithms, multi-bit arithmetic functions (*e.g.*, addition, multiplication) are decomposed into multiple bit-wise logic operations (*e.g.*, `AND`, `OR`, `XOR`), which are executed sequentially over multiple cycles. To realize these operations efficiently, existing BS-PIM designs [28]–[32] employ bit-line computing technique [28] that enable parallel execution of bit-wise logic operations, and further extend support to arbitrary multi-bit arithmetic functions. In essence, BS-PIM facilitates *in-situ* scalar-vector [31] and vector-vector [29] arithmetic operations, making it a versatile architecture for diverse computational tasks.

III. Motivations

A. Minimizing Data Movements by Efficient PIM architecture

Problem. Memory access poses a significant bottleneck in optimizing kNN search, prominently impacting both latency and energy consumption. For instance, prior work [10] highlights that *non-continuous DRAM access accounts for over 80% of runtime* in QuickNN [11], while the on/off-chip data movements contribute to 35% of the energy consumption in BitNN [10]. To address these challenges, we propose to leverage PIM architecture to accelerate kNN search and minimize both on-chip and off-chip data movement.

Idea. Our approach incorporates customized computing circuits alongside the BS-PIM micro-architecture design from [28]–[31] to enable efficient in-memory kNN search. The large on-chip memory capacity in our design eliminates runtime off-chip memory access, while delicately optimized execution flows significantly reduce on-chip data movement and maximize computing parallelism. Furthermore, the bit-serial execution model inherent to BS-PIM facilitates energy-efficient *in-situ* arithmetic operations, thereby substantially lowering the energy consumption associated with kNN search.

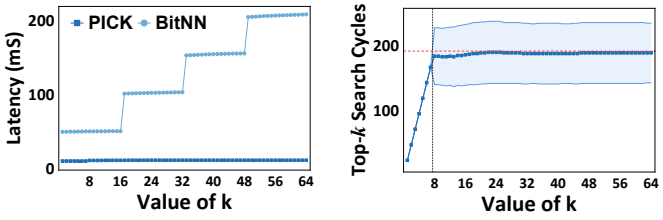
B. Optimizing Distance Calculation through Bit-width Clipping

Problem. Bit-serial algorithms support a wide range of arithmetic functions, however, at the cost of high latency. For instance, an N-bit unsigned integer multiplication requires $(N^2 + 5N - 2)$ logic operations [30], *i.e.*, consuming 334 clock cycles for 16-bit data. Despite substantial throughput achieved through extensive computing parallelism, the execution latency of BS-PIM remains considerably higher than specialized digital circuits, particularly for distance calculation in kNN search. Reducing the bit-width of point cloud coordinates (x, y, z) can only marginally improve the latency since a minimum of 14-bit is required to maintain accuracy, as in BitNN [10].

Idea. We propose *bit-width clipping* to clip the bit-width precision of multiplications in distance calculation, thereby reducing operational cost and improving latency, with minimal impact on accuracy. This method leverages the observation that the *smallest distance, indicating the nearest point, can be precisely represented with fewer bits*. For example, in the 1-NN scenario shown in Fig. 1(a), the nearest reference point $\boldsymbol{p}_3$ to the query point $\boldsymbol{p}_q$ has a distance d_3=13, which can be accurately represented using only the lower four bits. Using additional bits to accurately represent distances $\{d_1, d_2, d_4\}$ is unnecessary for identifying the nearest point but incurs more latency and energy.

Therefore, we clip the bit-width of squaring (*i.e.*, multiplication) in distance calculation to reduce operations and memory usage. The top of Fig. 1(b) shows distance calculation along the x-dimension without bit-width clipping, where $|\Delta x_i|^2$ is obtained through subtraction (*i.e.*, addition in 2's complement) and absolute value computation, followed by square operation. In contrast, the bottom portion of Fig. 1(b) applies bit-width clipping, where the bit-width of $|\Delta x_i|$ is clipped from 4-bit to 2-bit, greatly simplifying the subsequent square operation. Note that, the values of $|\Delta x_1|$ and $|\Delta x_4|$ exceed the representable range of 2-bit and are re-mapped to 3 (*i.e.*, the maximum value in 2-bit), to ensure they remain farther from the query point. Bit-width clipping preserves the accuracy of the kNN result, as $|\Delta x_i|^2$ remains the smallest value, while reducing 28% bit-wise logic operations.

In practical usage, where bit-widths are typically higher, bit-width clipping yields even greater improvements. We select an initial bit-width of 16 based on our analysis of four real-world datasets in Table I.



(a) Latency on KITTI at different k (b) Top-k search cycles at different k

Fig. 2: **Performance analysis at different k.** (a) The latency of PICK and BitNN [10] on KITTI dataset [20] for various k values (clipping width is 6-bit). (b) The cycles required by PICK for top-k search at different k. The vertical dashed line ($k = 7.5$) marks the boundary of small and large k, the shadow indicates variation.

Fig. 1(c) presents the number of operations at each clipping width, where we can achieve 56% reduction in operations by clipping 8-bit, with negligible accuracy degradation as proved in Section V.

C. Efficient Filtering-and-Selection Search for Arbitrary K

Problem. In kNN, k varies significantly w.r.t. different applications. For instance, point registration [6] often uses 1-NN, while classification [33] and segmentation [34] tasks require $k \in [3, 64]$. This variability in k poses a challenge for existing accelerators to achieve optimal performance. For example, BitNN [10] includes 16 top-k units, which support searching 16 nearest points in parallel for each iteration. When $k = 64$, four iterations are required with 4× latency as plotted in Fig. 2a, while also claimed to negatively affect accuracy [10]. Increasing the number of top-k units can mitigate latency growth, but it incurs hardware overhead and leads to underutilization for small k. In summary, *efficient support of arbitrary k is yet absent*. As the countermeasure, we introduce *filtering-and-selection-based top-k search to efficiently support arbitrary k*.

Idea. We first propose a selection-based method to implement top-k search of distances through k iterations of min-search. In each iteration, the nearest point with the smallest distance is identified, continuing this process for k times to obtain the k closest points. Notably, previously identified points are excluded from subsequent searches. This approach is efficient when k is small but suffers from linearly increasing latency as k grows, as in Fig. 2b.

Therefore, for larger values of k, we propose a filtering-based method to improve efficiency. Specifically, we introduce a threshold t to filter the distances d_i, and count the number of points within this threshold ($d_i < t$), denoted as n. By selecting an appropriate threshold t that satisfies $n = k$, the n points within the threshold correspond exactly to the top-k nearest points. However, in practice, achieving an exact match is often challenging. Thus, we employ a fast bisection algorithm to dynamically adjust the threshold t, iteratively refining it until n is close to k, such that $|n-k| < m$, where m is a variable tolerance. Once this condition is met, the selection-based method finalizes the process by including or excluding points as necessary to ensure an exact match with k. The optimal value of m and the boundary that distinguishes small and large k are determined through explorations. In KITTI dataset [20], we select $k = 7.5$ as the boundary and $m = 4$. Consequently, arbitrary top-k searches are accurately achieved with nearly constant time complexity, as shown in Fig. 2.

IV. Architecture of PICK

A. Architecture Overview

The architecture of PICK is illustrated in Fig. 3, comprising multiple components designed to efficiently perform kNN search. The

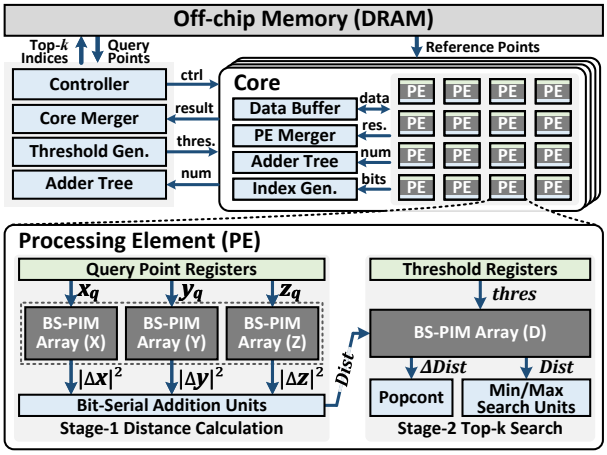


Fig. 3: **Overview of PICK architecture and dataflow.** The reference points are streamed from off-chip to the processing elements (PEs) of multiple cores for parallel computing. BS-PIM arrays cooperate with customized circuits to perform distance calculation and top-k search with a two-stage pipeline. The local search results are merged and the indices of the top-k nearest points are restored to the off-chip.

architecture features *multiple cores* dedicated to distance calculation and top-k search, a *core merger* for consolidating local search results from individual cores, a *threshold generator* and an *adder tree* for point filtering and counting (detailed in Section III-C), and a *controller* responsible for execution scheduling. Notably, the cores also function as on-chip storage for point cloud coordinates.

Core and PE. Each core integrates several key elements, including multiple processing elements (PEs), a data buffer with transposable access capabilities [28], a PE merger to aggregate search results from PEs, a local adder tree for point counting, and an index generator for retrieving the indices of searched points. At the core's foundation, the PE serves as the fundamental unit for kNN search. Each PE consists of four BS-PIM arrays dedicated to processing the x, y, and z dimensions, as well as distance computations. Additional components include registers for query points and thresholds, bit-serial-based units for arithmetic operations such as addition and min/max search, and a pop-count module for efficiently counting filtered points. This modular design ensures optimized performance and scalability for kNN search.

Dataflow of PICK, as annotated in Fig. 3, begins with point cloud data (coordinates) generated by sensors (*e.g.*, LiDAR) and stored in off-chip memory. Execution starts by streaming the reference points into the data buffers of multiple cores, where they are loaded into processing elements (PEs) in a transposed bit layout [28]. The reference points are stored in the BS-PIM arrays, while the coordinates of the query point are loaded into the PE registers.

Distance calculations for the x, y, and z dimensions are performed in parallel to compute $|\Delta x|^2$, $|\Delta y|^2$, and $|\Delta z|^2$, which are summed using bit-serial addition units and stored in the BS-PIM array dedicated to distance (Array D) for subsequent top-k search. For small values of k, the top-k nearest points are identified through k iterations of min-search, leveraging bit-serial min/max-search units, as detailed in Section III-C. For larger values of k, filtering is applied before selection, where distances are compared against a dynamical threshold, and the pop-count module counts the points within the threshold.

To optimize performance, a two-stage pipeline is employed to parallelize distance calculations and top-k search, to be described in Section IV-C. Since the number of reference points typically exceeds the capacity of a single PE, the data is distributed across multiple

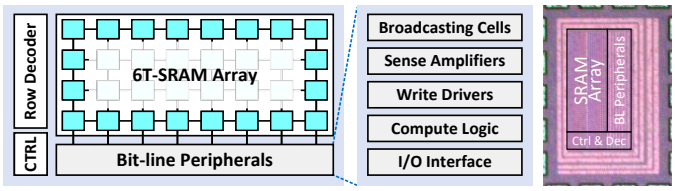


Fig. 4: **BS-PIM micro-architecture and micrograph of our demonstrated chip.** Parallel scalar/vector-vector bit-wise logic operations (AND, OR, XOR, and FA) are implemented utilizing bit-line computing technique [28] and integrated bit-line peripherals.

PEs and cores to enable parallel processing. Local search results, including values and indices, are hierarchically merged to produce the final result. Ultimately, the indices of the top-k nearest points are written back to off-chip memory.

B. Micro-architecture

1) BS-PIM Array: The micro-architecture of a single BS-PIM array, illustrated in Fig. 4, is designed to handle arithmetic operations (*e.g.*, addition, subtraction, multiplication) essential for kNN search. The circuit design of BS-PIM leverages optimizations from prior works [28]–[31], ensuring efficient and reliable performance.

At the core of the BS-PIM design is the 6T-SRAM array, which stores reference points or distances and enables parallel bit-wise logic operations (AND and NOR) through bit-line computing [28] and its associated bit-line peripherals. These peripherals are integral to both memory functionality and bit-serial computation. The broadcasting cells [31] facilitate the bit broadcasting of query points and distance thresholds, enabling concurrent scalar-vector arithmetic across all reference points and distances stored in the SRAM array.

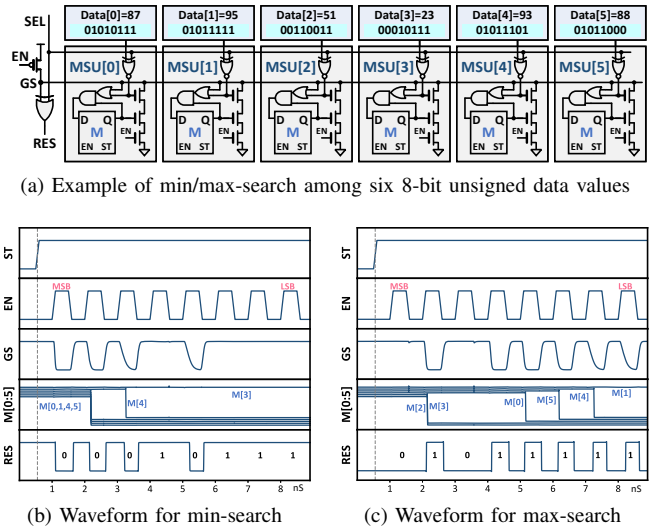
Additional components include sense amplifiers and write drivers, which support memory read and write operations. Compute logic modules handle post-processing tasks such as bit-wise addition and multiplexing, while the I/O interface manages data input and output. These elements collectively ensure seamless integration of memory and computation within the BS-PIM array.

The demonstrated BS-PIM array, as depicted in Fig. 4, can execute the described computations at a maximum frequency of 1.35 GHz, making it a highly efficient and scalable solution for kNN search.

2) Min/Max-Search Units: The micro-architecture of the min/max-search units is depicted in Fig. 5a, which implements bit-serial-based vector-wise min/max search by iteratively comparing bits from the most significant bit (MSB) to the least significant bit (LSB) and conditionally updating the latch states (M). An example with six 8-bit unsigned integer values is shown in Fig. 5 to illustrate the procedure.

In each iteration, one bit from all data items is read as a bit vector, which is subsequently processed by the search units. This work specifically addresses the min/max-search of unsigned integers, as squared Euclidean distances are inherently non-negative. In this context, minimum and maximum values are distinguished by the concentration of '0' and '1' bits near the MSB, respectively. Therefore, *min-search can be treated as a max-search with input bits inverted*.

To accommodate this, we design max-search circuits and employ XNOR gates to invert input bits based on the SEL signal, as shown in Fig. 5a. When SEL is set to low, the circuit performs min-search, identifying Data[3] as the minimum value (Fig. 5b). Conversely, setting SEL to high enables max-search, identifying Data[1] as the maximum value (Fig. 5c). In PEs, the min/max-search process follows the same procedure but is executed with higher parallelism. Additionally, the index generator, as described in Fig. 3, converts



(a) Example of min/max-search among six 8-bit unsigned data values

(b) Waveform for min-search

(c) Waveform for max-search

Fig. 5: **Illustration of bit-serial min/max search.** (a) Example of min/max-search among six 8-bit unsigned integers by iteratively comparing the bits from MSB to LSB; (b) waveforms of min-search by setting SEL low, where the output RES is Data[3]; (c) waveforms of max-search by setting SEL high, where the output RES is Data[1].

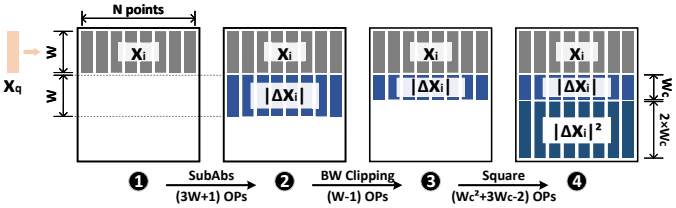


Fig. 6: **Distance calculation along the x-dimension,** encompassing three bit-serial-based functions: subtraction and absolute value (SubAbs), bit-width (BW) clipping, and square. W is the initial bit-width, W_c is the bit-width after clipping.

the latch bits (M) into an index using a priority encoder, enabling identification of the indices of the searched reference points.

C. Execution Flow

The execution flow of PICK follows the dataflow illustrated in Fig. 3, leveraging a two-stage pipeline to parallelize distance calculation and top-k search operations. The detailed implementation of these processes is outlined as follows.

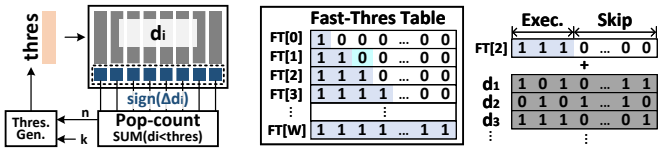
1) Distance Calculation with Bit-width Clipping: The distance calculation determines the Euclidean distances between reference points and a given query point, incorporating bit-width clipping as described in Section III-B. The coordinates $(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ of the reference points are stored as vectors in three BS-PIM arrays, utilizing a vertically aligned bit layout [28], as illustrated in Fig. 6. The query point coordinates are loaded into registers and broadcast to the BS-PIM arrays [31] for parallel computation.

Distance computation along each dimension involves three key bit-serial arithmetic functions, as shown in Fig. 6: subtraction and absolute value (SubAbs), bit-width clipping, and squaring. The implementation details of the bit-width clipping are provided in Algorithm 1. This technique introduces a complexity of $O(N)$, which simplifies the subsequent squaring operation, reducing its computational burden from $O(N^2)$. Finally, the summation of distance vectors across the x, y, and z dimensions is performed using bit-serial addition units, as depicted

Algorithm 1: Bit-serial-based Bit-width Clipping.

Input: A W-bit unsigned integer D; target bit-width W_c.

1: **for** each $i \in [W-1 : W_c+1]$ **do**
2: $\quad D[i] = OR(D[i], D[i-1])$
3: **for** each $i \in [0 : W_c]$ **do**
4: $\quad D[i] = OR(D[i], D[W_c])$
5: $D = D[0 : W_c]$



(a) Filtering-bases Search (b) Skip bit-serial addition with fast-thresholds

Fig. 7: **Illustration of filtering-based top-k search.** (a) The threshold is dynamically adjusted to make the number of points within the threshold approximates k. (b) Special thresholds (continuous '1's from MSB) are used to bypass the bit-serial additions of lower bits.

in Fig. 3. These addition units comprise digital 4-2 compressors and latches to achieve efficient accumulation of partial results.

2) Filtering-and-Selection-based Top-k Search: For small values of k, PICK performs top-k search by iteratively selecting the point with the smallest distance using min-search (Fig. 5b). To prevent previously selected points from being reconsidered in subsequent iterations, an additional bit is appended to the MSB of each reference point's distance. This bit acts as a mask, ensuring that selected points are assigned non-minimal distances and excluded from future searches.

For larger values of k, a filtering approach is employed to efficiently identify a subset of n points, where n approximates k within a predefined margin, such that $|n-k| < m$. The filtering-based top-k search process is depicted in Fig. 7a. During this process, the threshold generator produces a dynamic threshold value. The BS-PIM performs parallel bit-serial subtraction $\Delta d_i = (d_i - thres)$ and retains the sign bits. To simplify operations, negative thresholds are used, converting subtraction to addition in two's complement representation.

The pop-count unit sums the sign bits to determine the #points within the threshold ($d_i < thres$), denoted as n. Based on the comparison between n and k, the threshold generator adjusts the threshold iteratively: increasing $thres$ if $n < k$ or decreasing $thres$ if $n > k$. To accelerate threshold updates, a fast-threshold table is introduced, as shown in Fig. 7b. The table comprises threshold values consisting of continuous '1' bits starting from the MSB, enabling the system to bypass bit-serial additions for lower bits and save computational cycles. Thresholds from this table, denoted as *fast thresholds* (FT), are sequentially applied until $FT[x]$ results in $n < k$. Subsequently, thresholds are refined through binary approximation between $FT[x-1]$ and $FT[x]$, continuing until the condition $|n-k| < m$ is satisfied. Once the filtering phase concludes, the system applies either min-search (if $n < k$) or max-search (if $n > k$) to include or exclude the remaining $|n-k|$ reference points, finalizing the top-k search.

3) Pipeline Design: The pipeline of PICK is meticulously designed to optimize performance and maximize hardware utilization. Example pipeline and dataflow designs for distance calculation and top-k search are illustrated in Fig. 8. The operations are color-coded for clarity: memory read/write operations are marked in gray, while computational tasks are highlighted in green. The latency for distance calculation is constant, determined by the fixed bit-width and clipping width configurations. In contrast, the latency for the top-k search varies depending on the number of points processed, as shown in Fig. 2b.

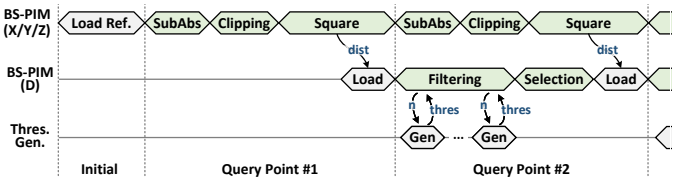


Fig. 8: **Pipeline design** of distance calculation and top-k search to hide latency and improve throughput.

TABLE I: Evaluation Benchmarks.

Dataset	Scenario	Sensor	Resolution	#Refer. Points	#Query Points
KITTI [20]	Outdoor	LiDAR	10^{-2}m	64k	64k
SONN [21]	Object	3D Scanner	10^{-4}m	64k	64k
S3DIS [22]	Indoor	3D Scanner	10^{-3}m	512k	512k
DALES [23]	Urban	LiDAR	10^{-2}m	512k	512k

V. Evaluation

A. Methodology

Benchmarks and Baselines. We construct evaluation benchmarks using four widely recognized real-world point cloud datasets, as summarized in Table I. These datasets encompass diverse application scenarios, including objects, indoor environments, outdoor settings, and urban landscapes, providing a comprehensive range of resolutions. The KITTI [20] and ScanObjectNN (SONN) [21] datasets represent smaller-scale scenarios, containing up to 64k points per frame. In contrast, S3DIS [22] and DALES [23] represent larger-scale environments, with up to 512k points per frame. Across all four datasets, the x, y, and z coordinates are encoded as 16-bit signed integers. For evaluation, we compare PICK against an RTX 3090 GPU (using PyG [35]) and prior kNN accelerators, including QuickNN [11], ParallelNN [12], and BitNN [10], which serve as our baselines.

Implementations and Specifications. We implement PICK in 28nm technology and comprehensively validate its critical functionalities. The SRAM is generated using a commercial memory compiler and integrated with customized computing circuits using Virtuoso. For the data buffers supporting bit-layout transposition, we adopt the performance metrics reported in [30]. The digital circuits are implemented in Verilog and synthesized using Design Compiler. Architectural overheads, including data paths and memory organization, are modeled using NVSim [36], while off-chip memory access is evaluated with DRAMSim3 [37]. Hardware configurations and performance specifications are detailed in Table II, where PICK is evaluated in two configurations: small (S), optimized for smaller datasets (KITTI and SONN), and large (L), tailored for larger datasets (S3DIS and DALES). The hardware specifications are input into an in-house, cycle-level simulator to assess overall performance, where a PyTorch-based arithmetic library is developed to emulate hardware behaviors and evaluate accuracy, latency, and energy.

B. Experiment Results

Accuracy Analysis is conducted across four datasets with varying values of k (ranging from 1 to 64) and clipped bit-width (ranging from 1 to 12), to demonstrate that our proposed bit-width clipping method does not introduce noticeable accuracy degradation under appropriate configurations. The results are presented in Fig. 9, where accuracy below 80% is colored in red, while accuracy exceeding 99.9999998% (6σ, considered effectively error-free) is colored in gray. As evident from the results, maintaining accuracy for larger values of k requires more bits. However, for configurations clipping 5 bits on KITTI and DALES, 6 bits on SONN, and 8 bits on S3DIS, no noticeable accuracy loss is observed, even for 64-NN. These configurations

TABLE II: Hardware Configurations and Performance Specifications

Component	Configurations	Area (mm^2)	Power (mW)
PE	(96+18)kb BS-PIM, 512 parallelism	0.024	3.503
Core	32 PEs, 32kb Buffer, 16k parallelism	0.799	114.8
PICK(S)	4 Cores, 64k parallelism, 1GHz	3.293	463.7
PICK(L)	32 Cores, 512k parallelism, 1GHz	25.78	3677

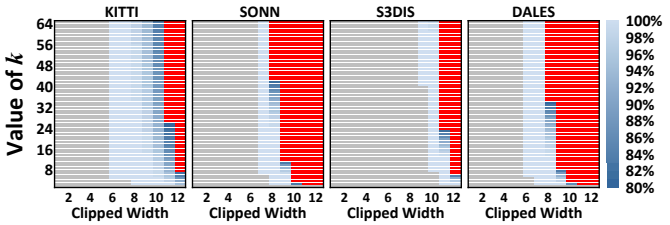
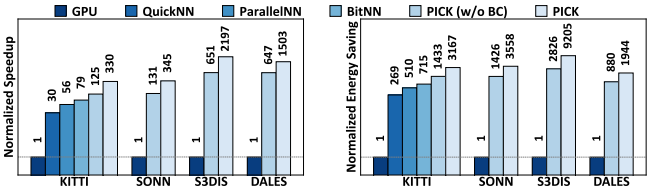
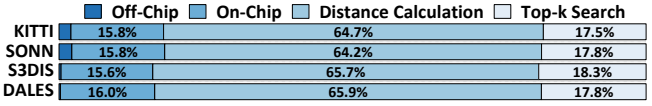


Fig. 9: **Accuracy map of PICK for varying values of k and clipped bit-width (initially 16-bit)**. Accuracy below 80% is filled in red, while accuracy exceeding 99.9999998% (6σ) is filled in gray.



(a) Normalized Speedup and Energy Saving



(b) Energy Breakdown of PICK

Fig. 10: **Normalized comparisons and energy breakdown of 5-NN search**. The bit-width clipping feature is disabled in PICK (w/o BC) to highlight the performance improvements.

achieve a significant reduction in the operational cost of distance calculation, ranging from 38% to 56%, as illustrated in Fig. 1(c).

Performance Analysis. Following prior works, we perform 5-NN searches on the four datasets to evaluate the performance of PICK, and compare it with GPU, QuickNN [11], ParallelNN [12], and BitNN [10]. The clipped bit-widths for the datasets are configured as $\{8,9,11,8\}$-bit, yielding accuracies of 99.38%, 99.94%, 99.73%, and 99.89%, respectively. To emphasize the effectiveness of our proposed bit-width clipping, we disable bit-width clipping in PICK (w/o BC). For fair comparisons, all non-GPU designs are normalized to the same power level. As shown in Fig. 10a, PICK achieves a 4.17× speedup and a 4.42× energy saving compared to the state-of-the-art design, BitNN, on the KITTI dataset. Furthermore, the bit-width clipping technique contributes to an additional 2.74× speedup and 2.54× energy savings. The energy breakdown, depicted in Fig. 10b, reveals that distance calculation and top-k search dominate the energy consumption. However, the costs of on-chip and off-chip data movements are minimized by the efficient PIM architecture and optimized execution flow, contributing to the overall efficiency.

Ablation Study. Fig. 11 presents the latency and energy scaling of PICK across different values of k, with bit-width clipping (BC), filtering-and-selection-based search (FSS), and fast-threshold (FT) individually disabled. For each value of k, the smallest clipping width is selected to maintain an accuracy above 99%. The stepwise jumps in the curves are primarily attributed to changes in the clipping width. Overall, BC demonstrates the most substantial impact on performance

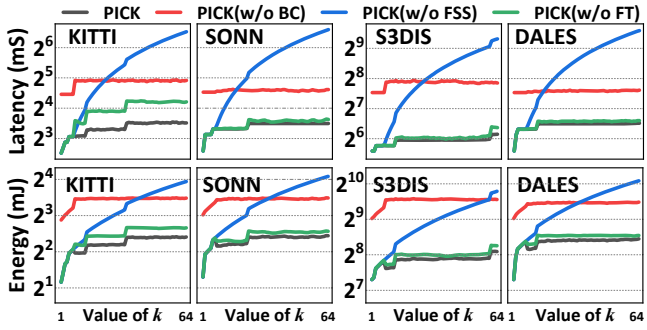


Fig. 11: **Latency and energy scaling** of PICK when disabling bit-width clipping (BC), filtering-and-selection-based search (FSS), and fast-threshold (FT), with accuracy constrained above 99%.

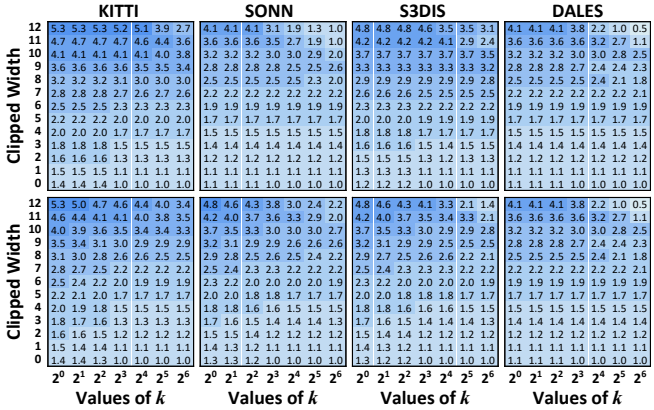


Fig. 12: **Latency (top) and energy (bottom) improvements** of PICK for varying values of k and clipped bit-width (initially 16-bit).

and energy efficiency. However, as k increases, FSS emerges as the dominant factor driving further optimizations. These results highlight the complementary contributions of the proposed techniques in scaling latency and energy efficiency of PICK across different configurations.

Data Movements Reduction. The data movement of PICK is significantly reduced, owing to its efficient architecture and optimized data flow. For example, on the KITTI dataset, BitNN [10] incurs 5.4MB of DRAM access, while PICK requires only 1.4MB for initialization and result storage, achieving over 70% reduction by fully eliminating runtime off-chip access. Additionally, on-chip access to buffers and registers is reduced from 0.4GB to 0.28GB.

Design Space Exploration. Fig. 12 illustrates the latency (top) and energy (bottom) improvements of PICK across varying values of k and clipped bit-width, using the performance of 64-NN without bit-width clipping as the baseline. PICK can achieve up to 5.29× speed and 5.25× energy saving. When combined with the accuracy map in Fig. 9, the optimal configuration can be selected based on specific task requirements. For example, applications demanding high accuracy, such as autonomous driving, may adopt lower clipped bit-widths, whereas latency-sensitive tasks, such as AR/VR applications, can benefit from higher clipped bit-widths for improved performance.

VI. Conclusion

This work proposes PICK, the first PIM architecture for kNN search in point clouds, incorporating novel bit-width clipping technique and filtering-and-selection-based search strategy to optimize the distance calculation and top-k search. Compared to existing designs, PICK achieves remarkable performance improvements.

REFERENCES

[1] Behnam Behroozpour, Phillip AM Sandborn, Ming C Wu, and Bernhard E Boser. Lidar system architectures and circuits. *IEEE Communications Magazine*, 55(10):135–142, 2017.

[2] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A Chapman, Dongpu Cao, and Jonathan Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3412–3432, 2020.

[3] Gang Wang, Wenlong Li, Cheng Jiang, Dahu Zhu, Zhongwei Li, Wei Xu, Huan Zhao, and Han Ding. Trajectory planning and optimization for robotic machining based on measured point cloud. *IEEE transactions on robotics*, 38(3):1621–1637, 2021.

[4] Daniel Garrido, Rui Rodrigues, A Augusto Sousa, Joao Jacob, and Daniel Castro Silva. Point cloud interaction and manipulation in virtual reality. In *2021 5th International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 15–20, 2021.

[5] Erik Sandström, Yue Li, Luc Van Gool, and Martin R Oswald. Point-slam: Dense neural point cloud-based slam. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18433–18444, 2023.

[6] Zhengyou Zhang. Iterative closest point (icp). In *Computer vision: a reference guide*, pages 718–720. Springer, 2021.

[7] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5565–5573, 2019.

[8] Renrui Zhang et al. Nearest neighbors meet deep neural networks for point cloud analysis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1246–1255, 2023.

[9] Ziyu Ying, Sandeepa Bhuyan, Yan Kang, Yingtian Zhang, Mahmut T Kandemir, and Chita R Das. Edgepc: Efficient deep learning analytics for point clouds on edge devices. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[10] Meng Han, Liang Wang, Limin Xiao, Hao Zhang, Tianhao Cai, Jiale Xu, Yibo Wu, Chenhao Zhang, and Xiangrong Xu. Bitnn: A bit-serial accelerator for k-nearest neighbor search in point clouds. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1278–1292. IEEE, 2024.

[11] Reid Pinkham, Shuqing Zeng, and Zhengya Zhang. Quicknn: Memory and performance optimization of kd tree based nearest neighbor search for 3d point clouds. In *2020 IEEE International symposium on high performance computer architecture (HPCA)*, pages 180–192. IEEE, 2020.

[12] Faquan Chen, Rendong Ying, Jianwei Xue, et al. Parallelnn: A parallel octree-based nearest neighbor search accelerator for 3d point clouds. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 403–414. IEEE, 2023.

[13] Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, and Yuhao Zhu. Juno: Optimizing high-dimensional approximate nearest neighbour search with sparsity-aware algorithm and ray-tracing core mapping. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 549–565, 2024.

[14] Yangfan Li, Mengquan Li, Cen Chen, Xiaofeng Zou, Hongen Shao, Fengxiao Tang, and Kenli Li. Simdiff: Point cloud acceleration by utilizing spatial similarity and differential execution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[15] Chengliang Wang, Zhetong Huang, Ao Ren, et al. An fpga-based knn seach accelerator for point cloud registration. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2024.

[16] Yunhao Hu, Hao Sun, Chunxu Guo, Qi Deng, and Yajun Ha. An energy-efficient and fast knn search accelerator for large scale point cloud map. In *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 2023.

[17] Yejin Lee, Hyunji Choi, Sunhong Min, Hyunseung Lee, Sangwon Beak, Dawoon Jeong, Jae W Lee, and Tae Jun Ham. Anna: Specialized architecture for approximate nearest neighbor search. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 169–183. IEEE, 2022.

[18] Batcher. Bit-serial parallel processing systems. *IEEE Transactions on Computers*, 100(5):377–384, 1982.

[19] RoboSense. 128-beam lidar customized for l4 autonomous vehicle. https://www.robosense.cn/en/rslidar/RS-Ruby_Plus.

[20] Andreas Geiger et al. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[21] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1588–1597, 2019.

[22] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1534–1543, 2016.

[23] Nina Varney, Vijayan K Asari, and Quinn Graehling. Dales: A large-scale aerial lidar data set for semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 186–187, 2020.

[24] Naveen Verma, Hongyang Jia, et al. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019.

[25] Donghyuk Kim, Chengshuo Yu, Shanshan Xie, Yuzong Chen, Joo-Young Kim, Bongjin Kim, Jaydeep P Kulkarni, and Tony Tae-Hyoung Kim. An overview of processing-in-memory circuits for artificial intelligence and machine learning. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(2):338–353, 2022.

[26] Hidehiro Fujiwara, Haruki Mori, Wei-Chang Zhao, Kinshuk Khare, Cheng-En Lee, Xiaochen Peng, Vineet Joshi, Chao-Kai Chuang, Shu-Huan Hsu, Takeshi Hashizume, et al. 34.4 a 3nm, 32.5 tops/w, 55.0 tops/mm 2 and 3.78 mb/mm 2 fully-digital compute-in-memory macro supporting int12× int12 with a parallel-mac architecture and foundry 6t-sram bit cell. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 67, pages 572–574. IEEE, 2024.

[27] Yiyang Yuan, Yiming Yang, Xinghua Wang, Xiaoran Li, Cailian Ma, Qirui Chen, Meini Tang, Xi Wei, Zhixian Hou, Jialiang Zhu, et al. 34.6 a 28nm 72.12 tflops/w hybrid-domain outer-product based floating-point sram computing-in-memory macro with logarithm bit-width residual adc. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 67, pages 576–578. IEEE, 2024.

[28] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *2018 ACM/IEEE 45Th annual international symposium on computer architecture (ISCA)*, pages 383–396. IEEE, 2018.

[29] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. Duality cache for data parallel acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 397–410, 2019.

[30] Jingcheng Wang, Xiaowei Wang, et al. A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing. *IEEE Journal of Solid-State Circuits*, 55(1):76–86, 2019.

[31] Chen Nie, Chenyu Tang, Jie Lin, Huan Hu, Chenyang Lv, Ting Cao, Weifeng Zhang, Li Jiang, Xiaoyao Liang, Weikang Qian, et al. Vspim: Sram processing-in-memory dnn acceleration via vector-scalar operations. *IEEE Transactions on Computers*, 2023.

[32] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. Simdram: A framework for bit-serial simd processing using dram. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 329–345, 2021.

[33] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[34] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

[35] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[36] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[37] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. Dramsim3: A cycle-accurate, thermal-capable dram simulator. *IEEE Computer Architecture Letters*, 19(2):106–109, 2020.

Expanding Logical Space Freely: A Memory-efficient Mapping Table Design for Compressional SSDs

Zixuan Huang†, Tianyu Wang*‡, Kecheng Huang†, Zelin Du†, Zili Shao†
†The Chinese University of Hong Kong, Hong Kong, China
‡Shenzhen University, Shenzhen, China
{zxhuang, kchuang21, zldu22, shao}@cse.cuhk.edu.hk, tywang@szu.edu.cn

***Abstract*—Compressional SSDs can be a mixed blessing. While they offer users expanded logical space beyond the physical capacity, they complicate the Flash Translation Layer (FTL) design by requiring a larger Logical-to-Physical (L2P) address mapping, which places a heavier burden on the limited in-SSD memory, leading to a degraded I/O performance. In this paper, we aim to reduce the memory footprint of the L2P mapping table in compressional SSDs by proposing a novel N-to-1 L2P mapping table design that consolidates multiple logical entries into a single entry. This approach eliminates the duplication of physical page numbers when a physical page contains several compressed logical pages. To accommodate the dynamic compression ratios of real-world workloads, we introduce promotion and demotion that enable mapping table entries to migrate between pages with different compression ratios. Additionally, to address the issue of partial invalidation—where some compressed logical pages within a physical page are invalid due to the N-to-1 mapping—we present a compression-aware garbage collection algorithm aimed at minimizing the number of copy operations for partially invalid physical pages. We have implemented our design in MQsim, a widely used SSD simulator, and have conducted a series of experiments to evaluate the effectiveness of the proposed techniques. The results demonstrate that our approach significantly reduces the mapping table size in compressional SSDs, leading to an improved mapping table cache hit ratio and a reduced I/O latency compared to traditional compressional SSDs.**

***Index Terms*—Compressional SSD, FTL, Cache management**

I. Introduction

The rapid growth of data volume has significantly increased the demand for high-capacity SSDs [21], [35]. However, due to the diminishing marginal effect, with Multi-Level-Cell (MLC)-based SSDs advancing to 5-bit or even 6-bit levels, the pace of capacity development begins to slow down [3], [7]. As an alternative approach, compressional SSDs emerge to offer users more available storage space than the actual physical space by compressing multiple logical pages into one physical page [2], [13]. Although compressional SSDs excel in workloads with higher compression ratios (CRs), they disrupt the traditional one-to-one Logical-to-Physical (L2P) address mapping by allowing multiple logical pages to be mapped to the same physical page. This complicates SSD firmware design.

With compressional SSDs, a larger logical space directly increases the size of the L2P mapping table, placing even more strain on limited in-SSD memory. For instance, if an 8TB compressional SSD with a 4KB page size exposes 16TB of logical space to users, the entire L2P mapping table will be doubled to 16GB which is a significant overhead for the limited in-SSD memory. Thus, existing approaches usually store the mapping table in NAND flash and cache it in the in-SSD memory. Considering that the user data is also cached in the memory, a larger mapping table inevitably compromises the cache hit ratio, thus affecting the I/O performance of compressional SSDs as more I/O operations require reading the mapping page from NAND flash before accessing corresponding data pages.

We have observed that inside compressional SSDs, a low cache hit ratio significantly prolongs access time, while the compression logic has a comparatively minor impact on overall latency [39]. This diminishes the effectiveness of existing studies that focus on optimizing compression algorithms [4], [6], [19], [22], [28], [37] or workflows [15], [17], [18], [20], [24], [30], [32], [36]. Therefore, in this paper, we aim to design a memory-efficient N-to-1 L2P mapping table for compressional SSDs. Specifically, our key idea is to eliminate redundancy in the traditional one-to-one L2P mapping table, where a Physical Page Number (PPN) can appear in multiple entries as in compressional SSDs multiple logical pages can be mapped to the same physical page. In our N-to-1 mapping table, consecutive N logical pages are consolidated into a single entry, efficiently eliminating the redundancy of physical page information. To achieve this, several challenges need to be conquered.

First, since different types of data exhibit varying CRs, a fixed parameter N in the N-to-1 mapping table cannot accommodate all scenarios effectively. Therefore, supporting diverse parameter N settings in the mapping table becomes the primary challenge. To address this, we introduce an additional marker in each Global Translation Directory (GTD) entry. The entry corresponds to the parameter N. It indicates how many consecutive logical pages can be mapped through a single second-level entry in the corresponding translation page based on the compression ratio of data. This allows us to efficiently support varying compression ratios for different workloads.

Second, as each entry represents multiple logical pages, traditional translation page allocation scheme becomes ineffective. With one translation page containing multiple entries with various CRs, logical space fragmentation becomes another critical challenge. To efficiently manage entries in a translation page, we introduce the concept of Translation Chunk inside each translation page, enabling fine-grained management of translation page space. For instance, a translation page can have eight 8-to-1 chunks, four 4-to-1 chunks, two 2-to-1 chunks, one 1-to-1 chunk, or other combinations. Building upon this, we

*Corresponding author

propose a partial translation page allocation scheme with chunk granularity, which eliminates the need to adjust the logical space range within one directory entry. This makes the mapping table more flexible and efficient.

Third, since the data stored on each logical page may change, its compression ratio may also change correspondingly, presenting another challenge for our N-to-1 mapping table design. If data that initially has a higher CR is updated to a lower CR, the original N-to-1 mapping becomes invalid as more physical pages are needed to store the original data from the N logical pages. Thus, we propose promotion and demotion for the translation page to dynamically adjust the compression ratios of the entries within the translation page. When some data is updated with a lower CR (i.e., requiring more physical pages), we reduce the CR of the corresponding translation page to allow each entry to manage fewer logical pages. In contrast, if all the entries in a translation page are updated with higher CR data, we promote the page to a higher CR to maximize space utilization in the mapping table. Thus, we can effectively keep the mapping table size relatively small, achieving better memory efficiency.

The final challenge concerns garbage collection (GC). In compressional SSDs, since each physical page contains multiple compressed logical pages, the invalidation of any single logical page results in a partial invalidation of the entire physical page, making traditional GC algorithms less effective. Thus, we develop a compression-aware GC mechanism that delays the reallocation step and retains valid pages in memory, enabling potential recompression of new updates and leading to space savings. We also take this opportunity to assess whether the corresponding translation page can be promoted at that time.

We have implemented our design in MQsim [33], a widely used SSD simulator, and evaluate the effectiveness of the proposed techniques. Results show that we can reduce the mapping table size by 50%, and achieve 2.06x speedup in IO latency compared with DFTL [11].

Our contributions are summarized as follows:

- We design a memory-efficient N-to-1 mapping table tailored for compressional SSDs, which substantially reduces the memory footprint and enhances I/O efficiency.
- A partial translation page allocation scheme is introduced to address the logical space fragmentation issue.
- We achieve dynamic workloads support by presenting promotion and demotion for translation pages, cooperating with a compression-aware GC policy for efficient physical space usage.
- The proposed technique is implemented in MQsim and evaluated using diverse real-world workloads, demonstrating its effectiveness.

II. Background and Motivation

A. Compressional SSDs

Compressional SSDs are solid-state drives incorporating compression techniques, often featuring hardware compression/decompression modules integrated directly into the SSDs [43]. This technology has gained significant attention in recent years [12], [14], [31]. In this work, we focus on one type of compressional SSDs that support built-in transparent data compression capability, in which data are compressed at a fixed granularity. For example, compressional SSDs from ScaleFlux use a per-4KB compression scheme in which every 4KB logical page is compressed individually and then multiple compressed pages are stored together in one physical page as much as possible. Figure 1 illustrates this process, where four logical pages are compressed and stored together in a single physical page. Compared with host-side compression [1], [5], [26], [38], such in-storage transparent compression completely relieves host CPU from compressing and decompressing data. To facilitate decompression and reverse mapping, the corresponding metadata can be stored in the out-of-band (OOB) area of a physical page. This metadata contains information necessary for identifying the logical page address and its size during the decompression process. Moreover, the system maintains bitmaps for every physical page, indicating valid logical pages for garbage collection.

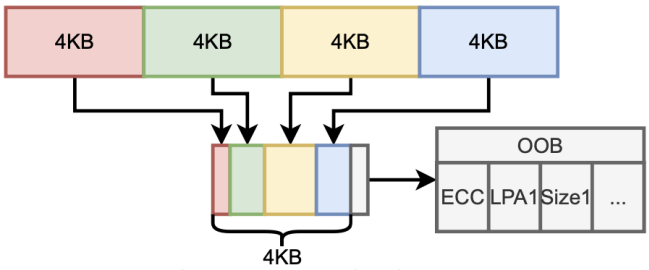


Fig. 1: Compression in SSDs.

B. Mapping Table Issues

Inside the FTL, an L2P mapping table is maintained for address translation. Constrained by limited in-SSD memory space, DFTL proposes on-demand page-level mapping schemes. The mapping table is stored in flash memory and cached in DRAM. A Global Translation Directory(GTD) is needed for indexing, which tracks the mapping of the translation pages. DFTL significantly reduces the amount of memory needed. This idea works well in traditional SSDs [16], [23], [27], [29], [42]. However, when applied to compressional SSDs, it does not address the problem of physical page number redundancy. In compressional SSDs, multiple logical pages are stored within a single physical page, resulting in consecutive entries in the mapping table that store the same physical page number. This leads to a waste of space and low cache hit ratios, which severely degrade the performance of the FTL.

C. Garbage Collection Issues

With the out-of-place update characteristic of flash memory, old physical pages become invalid once new data is written to other locations. To make room for new write operations, GC is necessary to reclaim these invalid physical pages. However, GC operations significantly affect the performance, since the basic unit of an erase operation is the block [8], [10], [34], [40], [41]. Compressional SSDs have brought new issues to garbage collection. When compressional SSDs issue an update request, it may only partially affect pages in one physical page, making

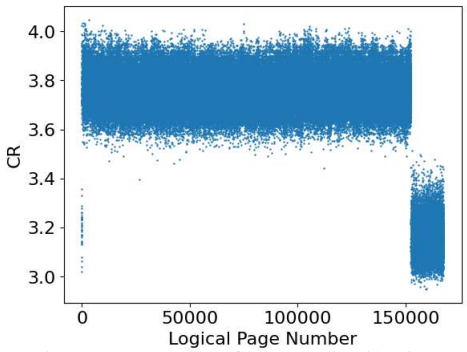


Fig. 2: B-tree log file CR distribution.

this physical page partially valid. Unfortunately, a normal GC operation only considers whether a physical page is valid or not, and it cannot free the invalid space inside these partially valid physical pages. As a result, the invalid space within these partially valid physical pages remains unutilized, impacting overall storage efficiency.

D. Motivation

Within a specific range, logical pages exhibit similar CRs. Our experiment demonstrates a spatial locality of CRs across pages. Illustrated in Figure 2, towards the beginning of the Logical Page Number (LPN) sequence, the majority of pages maintain a CR of approximately 4, while those towards the end exhibit a CR closer to 3. This observation indicates that logical pages within a defined range tend to possess comparable CRs, whereas variations exist across different sections of the page. These findings show the feasibility of consolidating several consecutive logical pages for N-to-1 mapping within a specific range. However, it also underscores the necessity of employing distinct N-to-1 mapping translations, even within a single file.

III. Overall Architecture

Figure 3 shows the overview of our mapping table design. It mainly contains two parts: the N-to-1 translation pages and the corresponding Global Translation Directory.

A translation page in our design has a similar structure to traditional translation pages, but each entry holds different flags. Each entry stores a physical page address representing N consecutive logical pages. As shown in Figure 3, the translation page stored in PPN 302 is a 2-to-1 translation page. It stores mapping entries with an CR of 2, indicating that two consecutive logical pages are compressed into one physical page. For instance, logical pages 0 and 1 are compressed and stored in physical page 405. Therefore, the first entry in this translation page stores PPN 405. To accommodate different CRs, we maintain four types of N-to-1 translation pages within the system. These types are defined by the value of N, which can be set as 1, 2, 4, 8 in this paper.

The Global Translation Directory is similar to the traditional mapping table, but with two additional flags: CR (Compression Ratio) and OF (Offset), while the logical space range of each entry remains unchanged. CR represents the compression ratio of the corresponding translation page, while OF indicates the offset of the associated Translation Chunk within the translation page. For example, in Figure 3, if the physical page size is

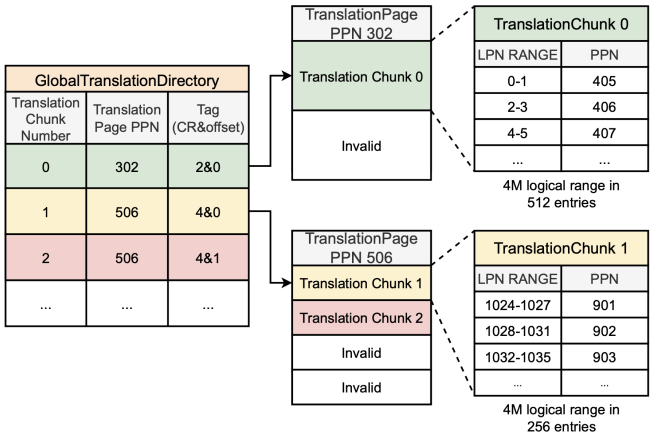


Fig. 3: Overview of our mapping table design.

4KB, each directory entry maps to a 4MB logical address. In traditional mapping tables, one directory entry corresponds to one translation page. However, with N-to-1 translation pages, each page encapsulates N*4MB of logical space. Therefore, the page can be divided into N distinct Translation Chunks, each representing 4MB of logical space. Each directory entry aligns with a Translation Chunk, facilitating efficient organization and mapping of logical addresses within the flash memory system. The Translation Chunk 0 occupies the first chunk of the Translation Page 302 with compression ratio 2, so its offset is 0 and CR is 2 in Tag. The CR and OF tags help the FTL efficiently locate the position of the associated Translation Chunk. The FTL can swiftly determine the physical page number corresponding to any given logical page number. This process ensures rapid and accurate mapping between logical and physical addresses.

IV. Detailed Techniques

In this section, we present our memory-efficient mapping table design in detail. First, to make full use of the space in N-to-1 translation pages, we propose a partial allocation scheme for managing translation pages to avoid logical space fragments. Next, we introduce demotion and promotion schemes to handle the CR inconsistencies in update requests. Additionally, the reusable allocation scheme reuses the invalid space caused by demotion and promotion. Finally, to free up more space, we develop a compression-aware GC.

A. Partial allocation

N-to-1 translation pages map larger logical spaces in one page, making the traditional full allocation strategy ineffective. Figure 4 illustrates a scenario with two write requests. The first write request targets Translation Chunk 0 with a CR of 2. If a full allocation strategy is employed, a 2-to-1 Translation Page 302 would map Translation Chunks 0 and 1. Then the second request targets Translation Chunk 3 with a CR of 4. In this case, Translation Chunks 3-6 are assigned to the 4-to-1 Translation Page 907. This causes fragmentation, as Translation Chunk 2 cannot utilize N-to-1 mapping. Regardless of its compression ratio, a full translation page is still needed, leading to inefficiency.

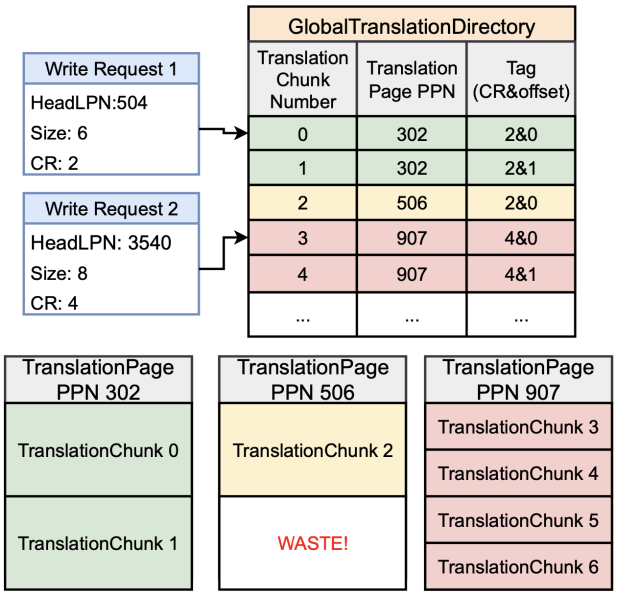


Fig. 4: Illustration of the full allocation issue.

To address this problem, a partial allocation strategy is employed. Partial allocation assigns only part of the translation page for each Translation Chunk. For instance, if a Translation Chunk requires the allocation of translation pages to store mapping entries with a CR of 2, only half of the 2-to-1 translation page is used, and the remaining half is available for subsequent allocations. The CR and OF tags in the directory enable efficient retrieval of the physical page numbers (PPNs) without any performance degradation. This approach ensures the benefits of N-to-1 mapping while minimizing wasted space and improving overall efficiency.

B. Demotion and Promotion

The update write requests may modify the CR of the same logical pages. If the new CR is lower than that of the Translation Chunk, a demotion function is invoked. On the other hand, if the new CR is higher, the system checks if the Translation Chunk can be promoted. If the Translation Chunk meets the requirements for promotion, a promotion scheme is executed.

In the first situation, new PPNs cannot be accommodated within the existing translation page. This occurs because an entry must store multiple PPNs when the new CR is lower than the previous one. For instance, in Figure 5, TranslationChunk 2 is stored in TranslationPage 506 with a CR of 4, where each entry can map 4 logical pages. If an update write request with a new CR 2 is received, LPN 2528-2531 can no longer be compressed into a single physical page. Instead, 2 PPNs are required for these 4 logical pages. In this case, a demotion scheme is invoked. This operation allocates a new chunk of an N-to-1 translation page for the lower CR (e.g., TranslationPage 302 with a CR of 2). The entry from the old translation page is copied to the new translation page. As shown in Figure 5, the updated mappings in LPN 2528-2531 are stored in two entries in new translation page, while the unchanged mapping for LPN 2532-2535 is directly copied to the new place.

In the second situation, if the new CR of the update write request is higher than the translation page CR, we check

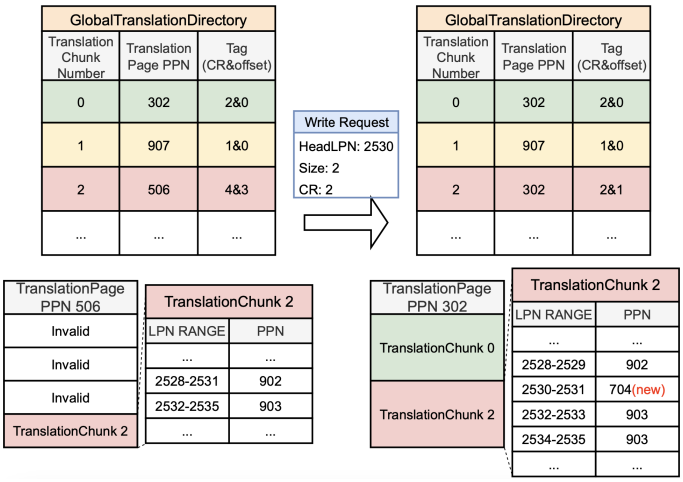


Fig. 5: Demotion with partial allocation.

whether every two entries in the Translation Chunk store the same PPN. If so, the chunk can be promoted to a higher CR. This condition may appear strict but is often met in practice. Demotion typically modifies only a small number of PPNs, leaving most unchanged. During demotion, most PPNs are copied to the new place, just like LPN 2532-2535 in Figure 5. If the modified entries LPN 2528-2531 are updated and store the same PPN with a higher CR again, the whole chunk is promotable. In this scenario, we allocate a new chunk of an N-to-1 translation page for the higher CR and move the entries from the old chunk. This approach reduces memory usage.

• **Reusable Allocation.** Partial allocation enables new translation pages to be allocated at a chunk granularity. However, during demotion and promotion schemes, moving entries to new chunks invalidates the old chunks, leading to an accumulation of invalid chunks in older translation pages. Over time, this accumulation reduces the overall memory efficiency of the system. In the worst case, our memory-efficient table needs the same number of translation pages as the traditional mapping table even if we use partial allocation. Reusing these invalid chunks helps reduce memory usage and minimizes the frequency of write-back operations. Therefore, we propose a reusable allocation scheme. We collect all the invalid chunks of different types of N-to-1 translation pages and reuse them when a new allocation request comes in. In detail, a free space list is maintained for each type of N-to-1 translation page to track free space. Invalid chunks from translation pages are added to the corresponding list during promotion and demotion. For a new allocation request, the corresponding list is checked for a reusable chunk. If the list is empty, the FTL allocates a new N-to-1 translation page and adds its chunks to the list. Furthermore, when the cache is full and the eviction is needed. We will first check if there is enough free space in memory to store all the valid and dirty parts in the evicted pages. If yes, we just allocate these valid and dirty parts to those free spaces in memory instead of writing back to NAND flash. If not, we will trigger a normal writeback operation.

C. Compression-aware GC

Partially valid data may use a whole physical page, which severely degrades the GC performance. As we can see in

Figure 5 , when a new update write request triggers a demotion operation, the data in PPN 902 becomes partially valid because the newly written data can not be compressed together. Normal GC can not release these invalid spaces by reallocating these partially valid pages to other blocks. In the worst case, one physical page may hold only one logical page with its CR is 8. As time goes on, less physical space can be released in one GC operation.

To address this issue, we propose a compression-aware GC. For each data block, we track valid physical page numbers and the valid logical page numbers for GC. Given a block to erase, we check these two numbers in this block. If the valid physical page number is low, normal GC performs well, and we execute it as well. However, if the number of valid physical pages is high and the number of valid logical pages is low, normal GC operation performs poorly, and compression-aware GC is triggered. Reallocation is delayed and the data is kept in memory. Then we directly erase the victim block to release the space immediately. The data waits for a new write request to attempt recompression of consecutive logical pages into one page. If recompression succeeds, we also check if the translation page chunk can be promoted.

V. Evaluation

We evaluate our mapping schemes on a 512GB SSD using MQSim, a state-of-the-art open-source SSD simulator. MQSim models all SSD components, including the host interface, SSD controllers, flash controllers, and flash chips. It supports multi-queue SSDs and measures end-to-end latency, making it ideal for our study. To adapt MQSim for our evaluation, we incorporated compression components and assessed overall performance in terms of end-to-end latency. In this section, we first present the setup and workloads and show the experimental results in terms of mapping table size, the cache hit ratio, and the latency. We compare our design with optimal-FTL, DFTL-base and DFTL-prefetch. In the optimal-FTL scenario, we assume the entire mapping table fits within DRAM. For DFTL-prefetch, a cache miss triggers the prefetching of the entire translation page, mitigating subsequent translation reads within the same page and effectively reducing translation page reads from cache misses. Subsequently, we evaluate our compression-aware GC scheme, and explore the performance impact with different cache sizes and large SSDs. At last, we analyze overheads of our design.

A. Experiment Setup

The parameters of the 512GB SSD are shown in Table I. In the evaluation, the mapping cache is set as large as the mapping table of a block-level FTL, which is 2MB. Other cache sizes are also considered. Furthermore, in the evaluation, we use real-world workloads to study the performance of DFTL and our design. These traces are collected from filebench [25]. The workload characteristics can be found in Table II. However, as these traces lack actual data, we utilize the SDGen scheme [9] to sample data from real applications, leveraging this information to generate CR information for the traces. In the next section, we show the performance of these FTL designs with different workloads.

TABLE I: The SSD parameters.

Parameter	Value	Parameter	Value
Channel	8	Page size	4KB
Chip	4	read latency	75us
Die	2	write latency	750us
Plane	2	erase latency	3.8ms
Block	2048	compression latency	50us
Page	256	decompression latency	2.5us

TABLE II: Workloads information.

Workloads	Number of Read Request	Average Read Length (4KB pages)	Number of Write Request	Average Write Length (4KB pages)
fileserver _ext4	141542	105.18	545689	37.83
fileserver _f2fs	346203	45.57	274453	90.78
varmail _ext4	121202	17.74	355954	10.25
varmail _f2fs	177277	17.22	253410	23.27

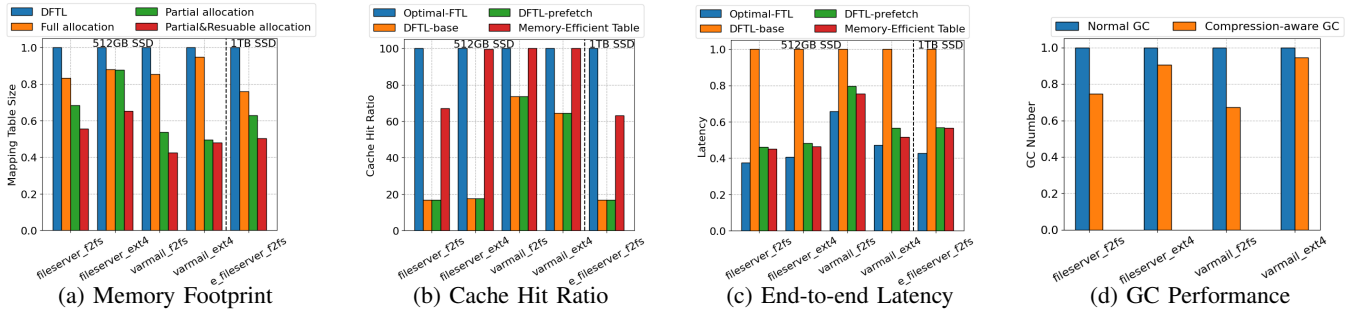
B. Overall Performance

1) Mapping Table Size: Our design significantly reduces the mapping table size compared with the DFTL, which only caches the translation page without reducing the mapping table size. By using DFTL as the baseline, we normalize other methods' table sizes for comparison. As shown in Figure 6(a), DFTL's 1-to-1 mapping corresponds to a value of 1. When implementing N-to-1 mapping with full allocation, only a 15% reduction in mapping table size is achieved. This approach leads to significant wastage of translation page space due to inefficient allocation strategies. In contrast, N-to-1 mapping with partial and reusable allocation strategies can significantly reduce the mapping table size. On average, the size of the whole N-to-1 mapping table is reduced by 50%.

2) Cache Hit Ratio: Our approach also leads to a notable increase in the cache hit ratio. Illustrated in Figure 6(b), our cache hit ratio surpasses that of DFTL by approximately 50%. In a 1-to-1 mapping scenario, a single translation page can only represent 4MB of logical space, whereas an N-to-1 mapping configuration can accommodate N*4MB of logical space. Leveraging this increased capacity, our design can store more mapping information within the cache. Therefore, we achieve a higher cache hit ratio and reduce the flash read and flash write operation numbers for mapping.

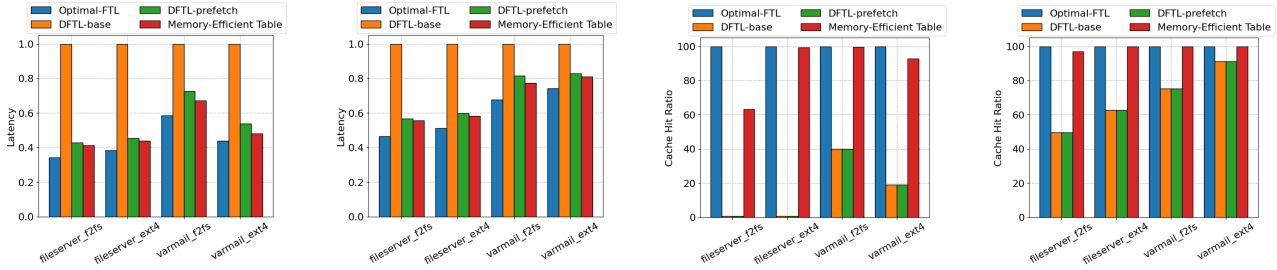
3) Latency: With the advantages of a reduced-size mapping table, our design exhibits improved latency performance. Figure 6(c) illustrates the end-to-end latency across various workloads. DFTL-base, with the poorest performance, is normalized to 1, and other methods are expressed as a fraction of its latency. The results demonstrate that our N-to-1 mapping approach delivers superior performance compared to both DFTL-base and DFTL-prefetch. Specifically, our N-to-1 mapping scheme achieves 5-10% performance improvements compared to DFTL-prefetch. Moreover, our design showcases a notable 2.06x speedup compared to DFTL-base.

4) GC Performance: To evaluate the performance of the compression-aware GC technique, we measure the number of GC operations across various workloads. As shown in Figure



(a) Memory Footprint (b) Cache Hit Ratio (c) End-to-end Latency (d) GC Performance

Fig. 6: Overall performance.



(a) Latency(256KB) (b) Latency(16MB) (c) Cache Hit Ratio(256KB) (d) Cache Hit Ratio(16MB)

Fig. 7: Performance with different cache sizes.

6(d), we establish the value of normal GC as 1, with the value of our design derived by dividing our GC numbers by the normal GC numbers. Our compression-aware GC approach results in approximately 22% fewer GC operations compared to normal GC. It shows that our compression-aware GC can free more physical space and enhance the performance.

C. Detailed Comparison

1) Cache size: This section explores the impact of cache size. We investigate two cache sizes: 256KB and 16MB. At 256KB, DFTL's cache hit ratio is low but stabilizes at 16MB. Figure 7 presents the performance of our design across cache sizes. With very small cache sizes, our design achieves a 2.2x latency speedup over DFTL-base. Our cache hit ratio exceeds DFTL's for small caches due to our memory-efficient N-to-1 mapping. Even with larger caches, our design maintains an average 1.6x speedup. The result shows our N-to-1 mapping scheme can work well even if the cache size is very small.

2) Large workloads in Large SSD: As the capacities of SSDs continue to expand, understanding the performance of our design in the context of larger SSDs becomes crucial. With a 1TB SSD environment, we scaled up the fileserver_f2fs workloads by doubling both the logical address and the request sizes. As Figure 6 shows, within a 1TB SSD setup, our cache hit ratio remains significantly superior to that of DFTL. Notably, our design achieves a 1.76x speedup compared to DFTL-base, showcasing the efficiency and scalability of our approach in handling larger SSD capacities.

D. Overhead Analysis

1) Global Translation Directory size: As above-mentioned, the logical space size represented by a single GTD entry unchanged. Therefore, when a compressional SSD exposes more logical space to the user, the size of the GTD increases proportionally. It is worth noting that the GTD size is relatively small in comparison. One GTD entry can represent 4MB of logical space with a 4KB page size. Given a total logical space of 512GB, there are 131072 entries in the Global Translation Directory. 4B are needed for each entry, so the entire GTD would only require 512KB, whereas the mapping table size would amount to 512MB if utilizing a 1-to-1 mapping strategy. It shows that even if the Global Translation Directory size doubles in compressional SSDs, it is still very small.

2) Tag size: In our N-to-1 mapping design, given a total logical space of 512GB, one byte is enough to store one Tag. Therefore, our design requires 128KB space to store the Tag information, which is negligible compared with the mapping table.

VI. Conclusion

In this paper, we propose a memory-efficient N-to-1 L2P mapping table design by reducing the size of the mapping table and enhancing latency performance. In our design, we have devised the demotion and promotion schemes, along with the partial and reusable allocation methods to fully utilize the N-to-1 translation page. Furthermore, we propose a compression-aware GC strategy to minimize the number of copy operations for partial-invalid physical pages. Our design has been implemented in MQsim, and the experimental results demonstrate significant improvements in the mapping table cache hit ratio and a reduction in I/O latency compared to traditional compressional SSDs.

Acknowledgment

The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 14202123, GRF 14200224), and the Innovation Capability Support Program of Shaanxi, China (No. 2023-CX-TD-08).

REFERENCES

[1] MySQL Compression. 2024 .https://dev.mysql.com/doc/refman/8.4/en/innodb-compression-internals.html.

[2] Scaleflux. https://scaleflux.com/.

[3] solidigmPLC. https://news.solidigm.com/en-WW/217006-solidigm-demonstrates-world-s-first-penta-level-cell-ssd-at-flash-memory-summit.

[4] M. Bartík, S. Ubik, and P. Kubalik. LZ4 compression algorithm on FPGA. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 179–182. IEEE, 2015.

[5] P. Boncz, T. Neumann, and V. Leis. FSST: fast random access string compression. *Proceedings of the VLDB Endowment*, 13(12):2649–2661, 2020.

[6] C.-H. Chen, C.-T. Chen, and W.-T. Huang. The real-time compression layer for flash memory in mobile multimedia devices. *Mobile Networks and Applications*, 13:547–554, 2008.

[7] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. *ACM SIGMETRICS Performance Evaluation Review*, 37(1):181–192, 2009.

[8] N. Dayan, P. Bonnet, and S. Idreos. GeckoFTL: Scalable flash translation techniques for very large flash devices. In *Proceedings of the 2016 International Conference on Management of Data*, pages 327–342, 2016.

[9] R. Gracia-Tinedo, D. Harnik, D. Naor, D. Sotnikov, S. Toledo, and A. Zuck. SDGen: Mimicking datasets for content generation in storage benchmarks. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 317–330, 2015.

[10] J. Guo, Y. Hu, B. Mao, and S. Wu. Parallelism and garbage collection aware I/O scheduler with improved SSD performance. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1184–1193. IEEE, 2017.

[11] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices*, 44(3):229–240, 2009.

[12] J. Hao, X. Chen, Y. Qiao, Y. Zhang, and T. Zhang. Implementing flash-cached storage systems using computational storage drive with built-in transparent compression. In *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–8. IEEE, 2021.

[13] X. Hu, F. Wang, W. Li, J. Li, and H. Guan. {QZFS}:{QAT} accelerated compression in file system for application agnostic and cost efficient data storage. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 163–176, 2019.

[14] K. Huang, Z. Shen, Z. Shao, T. Zhang, and F. Chen. Breathing new life into an old tree: Resolving logging dilemma of B+-tree on modern computational storage drives. *Proceedings of the VLDB Endowment*, 17(2):134–147, 2023.

[15] C. Ji, L.-P. Chang, L. Shi, C. Gao, C. Wu, Y. Wang, and C. J. Xue. Lightweight data compression for mobile flash storage. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–18, 2017.

[16] S. Jiang, L. Zhang, X. Yuan, H. Hu, and Y. Chen. S-FTL: An efficient address translation for flash memory by exploiting spatial locality. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12. IEEE, 2011.

[17] M. Kang, W. Lee, J. Kim, and S. Kim. PR-SSD: Maximizing partial read potential by exploiting compression and channel-level parallelism. *IEEE Transactions on Computers*, 72(3):772–785, 2022.

[18] D. Kim, Y. Won, J. Cha, S. Yoon, J. Choi, and S. Kang. Exploiting compression-induced internal fragmentation for power-off recovery in SSD. *IEEE Transactions on Computers*, 65(6):1720–1733, 2015.

[19] S. Lee, J. Park, K. Fleming, J. Kim, et al. Improving performance and lifetime of solid-state drives using hardware-accelerated compression. *IEEE Transactions on consumer electronics*, 57(4):1732–1739, 2011.

[20] Q. Li, L. Shi, R. Pan, C. Ji, X. Li, and C. J. Xue. Selective compression scheme for read performance improvement on flash devices. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 43–50. IEEE, 2018.

[21] Q. Li, Q. Xiang, Y. Wang, H. Song, R. Wen, W. Yao, Y. Dong, S. Zhao, S. Huang, Z. Zhu, et al. More than capacity: Performance-oriented evolution of pangu in alibaba. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 331–346, 2023.

[22] W. Liu, F. Mei, C. Wang, M. O'Neill, and E. E. Swartzlander. Data compression device based on modified LZ4 algorithm. *IEEE Transactions on Consumer Electronics*, 64(1):110–117, 2018.

[23] D. Ma, J. Feng, and G. Li. LazyFTL: A page-level flash translation layer optimized for NAND flash memory. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1–12, 2011.

[24] B. Mao, S. Wu, H. Jiang, Y. Yang, and Z. Xi. EDC: Improving the performance and space efficiency of flash-based storage systems with elastic data compression. *IEEE Transactions on Parallel and Distributed Systems*, 29(6):1261–1274, 2018.

[25] R. McDougall and J. Mauro. Filebench. *URL: http://www. nfsv4bat. org/Documents/nasconf/2004/filebench. pdf (Cited on page 56.)*, 2005.

[26] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.

[27] Y. Pan, H. Chen, J. Zhao, and Y. Xu. HCFTL: A locality-aware flash translation layer for efficient address translation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2477–2489, 2021.

[28] Y. Pan, Y. Li, H. Zhang, and Y. Xu. Lifetime-aware FTL to improve the lifetime and performance of solid-state drives. *Future Generation Computer Systems*, 93:58–67, 2019.

[29] Y. Pan, M. Lin, Z. Wu, H. Zhang, and Z. Xu. Caching-aware garbage collection to improve performance and lifetime for NAND flash SSDs. *IEEE Transactions on Consumer Electronics*, 67(2):141–148, 2021.

[30] Y. Park and J.-S. Kim. zFTL: Power-efficient data compression support for NAND flash-based consumer electronics devices. *IEEE transactions on consumer electronics*, 57(3):1148–1156, 2011.

[31] Y. Qiao, X. Chen, N. Zheng, J. Li, Y. Liu, and T. Zhang. Closing the B+-tree vs. LSM-tree write amplification gap on modern storage hardware with built-in transparent compression. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 69–82, 2022.

[32] Y. Song, Y. Huang, Y. Lv, Y. Zhang, and L. Shi. When F2FS meets compression-based SSD! In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*, pages 87–92, 2023.

[33] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu. MQSim: A framework for enabling realistic studies of modern Multi-Queue SSD devices. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 49–66, 2018.

[34] S. Wu, Y. Lin, B. Mao, and H. Jiang. GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12, 2016.

[35] Z. Yang, Q. Xu, S. Gao, C. Yang, G. Wang, Y. Zhao, F. Kong, H. Liu, W. Wang, and J. Xiao. OceanBase Paetica: A Hybrid Shared-Nothing/Shared-Everything Database for Supporting Single Machine and Distributed Cluster. *Proceedings of the VLDB Endowment*, 16(12):3728–3740, 2023.

[36] X. Yao, Q. Li, K. Lin, X. Gan, J. Zhang, C. Gao, Z. Shen, Q. Xu, C. Yang, and J. Xue. Extremely-Compressed SSDs with I/O Behavior Prediction. *ACM Transactions on Storage*, 20(4):1–38, 2024.

[37] K. S. Yim, H. Bahn, and K. Koh. A flash compression layer for smartmedia card systems. *IEEE Transactions on consumer Electronics*, 50(1):192–197, 2004.

[38] F. Zhang, W. Wan, C. Zhang, J. Zhai, Y. Chai, H. Li, and X. Du. CompressDB: Enabling efficient compressed data direct processing for various databases. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1655–1669, 2022.

[39] X. Zhang, J. Li, H. Wang, D. Xiong, J. Qu, H. Shin, J. P. Kim, and T. Zhang. Realizing transparent os/apps compression in mobile devices at zero latency overhead. *IEEE Transactions on Computers*, 66(7):1188–1199, 2017.

[40] B. Zhou, S. Wan, and C. Xie. Isolation: Inexpensively separating cold data via garbage collection to improve the lifetime and performance of nand flash ssds. *Concurrency and Computation: Practice and Experience*, 33(15):e5460, 2021.

[41] K. Zhou, S. Hu, P. Huang, and Y. Zhao. Lx-ssd: Enhancing the lifespan of nand flash-based memory via recycling invalid pages. In *Proc. 33rd Int. Conf. Massive Storage Syst. Technol.(MSST)*, pages 1–13, 2017.

[42] Y. Zhou, F. Wu, P. Huang, X. He, C. Xie, and J. Zhou. An efficient page-level FTL to optimize address translation in flash memory. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–16, 2015.

[43] A. Zuck, S. Toledo, D. Sotnikov, and D. Harnik. Compression and SSDs: Where and How? In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*, 2014.

VersaSlot: Efficient Fine-grained FPGA Sharing with Big.Little Slots and Live Migration in FPGA Cluster

Jianfeng Gu, Hao Wang, Xiaorang Guo, Martin Schulz and Michael Gerndt
Chair of Computer Architecture and Parallel Systems, Technical University of Munich
Email: {jianfeng.gu, whhao.wang, xiaorang.guo}@tum.de, {schulzm, gerndt}@in.tum.de

***Abstract*—As FPGAs gain popularity for on-demand application acceleration in data center computing, dynamic partial reconfiguration (DPR) has become an effective fine-grained sharing technique for FPGA multiplexing. However, current FPGA sharing encounters partial reconfiguration contention and task execution blocking problems introduced by the DPR, which significantly degrade application performance. In this paper, we propose VersaSlot, an efficient spatio-temporal FPGA sharing system with novel Big.Little slot architecture that can effectively resolve the contention and task blocking while improving resource utilization. For the heterogeneous Big.Little architecture, we introduce an efficient slot allocation and scheduling algorithm, along with a seamless cross-board switching and live migration mechanism, to maximize FPGA multiplexing across the cluster. We evaluate the VersaSlot system on an FPGA cluster composed of the latest Xilinx UltraScale+ FPGAs (ZCU216) and compare its performance against four existing scheduling algorithms. The results demonstrate that VersaSlot achieves up to 13.66x lower average response time than the traditional temporal FPGA multiplexing, and up to 2.19x average response time improvement over the state-of-the-art spatio-temporal sharing systems. Furthermore, VersaSlot enhances the LUT and FF resource utilization by 35% and 29% on average, respectively.**

I. Introduction

Field Programmable Gate Arrays (FPGAs) have rapidly gained popularity in edge and data center computing, offering efficient application acceleration with higher flexibility and lower power consumption [1]–[5]. Leading cloud providers [6]–[8] have integrated FPGAs into their platforms, allowing users to develop highly customized and on-demand applications with Quality of Service. Meanwhile, modern FPGAs have significantly enhanced their capabilities with advanced manufacturing processes and increased logic elements, able to handle more complex and diverse computational demands [9]–[11]. These improvements facilitate fine-grained FPGA sharing [12]–[15] in multi-user and multi-task environments.

Traditional FPGA virtualization and sharing techniques [7], [16], [17] allocate the entire FPGA to a single application and implement the time-multiplexing by performing a full fabric reconfiguration. This approach introduces significant context switch overhead and often leads to FPGA under-utilization, as applications may not require the full FPGA resources. Instead, *Dynamic Partial Reconfiguration* (DPR) [18] enables an FPGA to split the programmable fabric into multiple *slots*, which can be reconfigured independently to host arbitrary application logic at runtime. It provides the capability to perform *Partial Reconfiguration* (PR) on a slot while other slots continue running. Currently, DPR has emerged as the most efficient and flexible technique for spatio-temporal FPGA sharing and virtualization [13]–[15] in the data center.

However, current DPR-based spatio-temporal FPGA sharing still contains numerous overlooked challenges, leading to significant application performance degradation. First, in the current FPGA Processing System (PS), the PR module, *PCAP* (Processor Configuration Access Port), is limited to serial bitstream loading. This serial access causes frequent mutual blocking when multiple applications require PR concurrently. As the number of applications sharing an FPGA increases, the PR contention intensifies, resulting in highly prolonged and unpredictable application response times. Meanwhile, the mutual PR blocking further severely disrupts applications' original pipelines, leading application tasks with dependencies in various slots to wait for each other to complete, preventing further execution. Recent systems like Nimblock [15] and DML [14] overlook these problems, resulting in application performance that falls short of the expected pipeline outcomes. Moreover, these works [14], [15] used single-core task scheduling, in which PR operations further block the launching of application tasks, leading to severe task blocking problems.

Second, the uniform size of FPGA slots in previous work [14], [15] imposes strict constraints on task partitioning. For example, when optimizing application tasks using Xilinx's High-Level Synthesis (HLS) tools, resource consumption typically exhibits stepwise increases rather than linear growth, which can easily cause resource over-subscription and under-utilization within slots. While prior work [12] attempted to address the problem by dynamically adjusting slot sizes based on application resource demand, the solution needs regenerating bitstreams at runtime, incurring large runtime overhead.

To address these challenges, we propose **VersaSlot** (Versatile Slots), an efficient spatio-temporal FPGA sharing system, to achieve more effective FPGA multiplexing for applications. To mitigate PR contention among application tasks, VersaSlot introduces a novel *Big.Little* slot architecture. The architecture allows the system to bundle and load multiple tasks simultaneously to a Big slot for internal execution at runtime, thereby eliminating further PR contention with tasks from Little slots. Within the Big slot, tasks can still maintain their parallel or serial pipelines, as they do in the Little slots,

but without internal PR interference. Meanwhile, reduced PR contention also allows Little slots to load task bitstreams promptly, mitigating blocking delays and pipeline disruptions.

However, similar to ARM's big.LITTLE cores [19], the heterogeneous *Big.Little* slot design introduces increased complexity for resource allocation and scheduling. As for resource allocation, we propose an adaptive algorithm that combines primary allocation and redistribution along with binding and rebinding. The algorithm enables applications to flexibly switch between using Big and Little slots at runtime while also maximizing the utilization of available slot resources. The complementary between Big and Little slots can help mitigate under-utilization and over-subscription problems. In the context of scheduling, the VersaSlot dynamically bundles 3-in-1 tasks at runtime and schedules the corresponding tasks to available slots. The 3-in-1 task bundling can enhance the flexibility in task partitioning of applications. Importantly, VersaSlot decouples the PR from scheduling logic and distributes them across two CPU cores for asynchronous execution, thus effectively eliminating the task execution blocking problem.

For specific applications with few tasks and large batch sizes, PR occurs infrequently. By dividing Big slots into multiple Little slots, more applications can efficiently share resources, making an FPGA with only Little slots (*Only.Little*) also necessary. However, switching an FPGA between *Only.Little* and *Big.Little* configurations requires restarting the system and interrupting all task executions, which introduces a large runtime overhead. To address this, we propose a seamless cross-board switching and live migration mechanism based on system PR contention levels within a cluster, enabling the FPGA slot configuration switching with low overhead.

In a nutshell, the contributions are summarized as follows:

- We propose VersaSlot, an efficient spatio-temporal FPGA sharing system that introduces the novel *Big.Little* slot architecture. VersaSlot addresses the significant PR contention problem inherent in FPGA fine-grained sharing.
- We design an efficient slot allocation and dual-core scheduling algorithm for the *Big.Little* slot architecture to dynamically and efficiently utilize both kinds of slots and prevent the task execution blocking problem.
- We propose a seamless cross-board switching and live migration mechanism, enabling applications to transition among FPGAs with different slot configurations in a cluster for more efficient execution with low overhead.
- We evaluate the VersaSlot system on a real FPGA cluster, against four existing scheduling algorithms. VersaSlot achieves an average response time up to 13.66x lower than the traditional FPGA multiplexing, and up to 2.19x lower than the state-of-the-art system, and enhances the LUT and FF utilization by 35% and 29% on average.

II. Background

Dynamic Partial Reconfiguration (DPR), also known as Dynamic Function eXchange (DFX) from AMD/Xilinx [20], is an advanced FPGA configuration technique that enables dynamic reconfiguration of specific regions of the FPGA without interrupting the overall system operation. Instead, in traditional FPGA configuration flows, any functional modification requires reloading the entire bitstream file, leading to system downtime and a full restart, which causes large runtime overhead. The DFX process relies on the Processor Configuration Access Port (PCAP) [21] to load the partial bitstream from memory to the corresponding section of the FPGA. As it is essential to confirm the successful loading of the partial bitstream, DFX inherently prevents the PCAP from loading a new partial bitstream until the current one is fully loaded and blocks the corresponding CPU [21]. Meanwhile, to avoid unpredictable impacts on the remaining FPGA circuitry during the loading phase, a dedicated IP, DFX Decouple, is employed to decouple the reconfigurable region from the rest of the circuit. Recently, the DFX has been extensively used in FPGA virtualization and sharing in the cloud [13]–[15], [22], but limited to serial partial bitstream loading.

III. System Design

A. VersaSlot System Framework

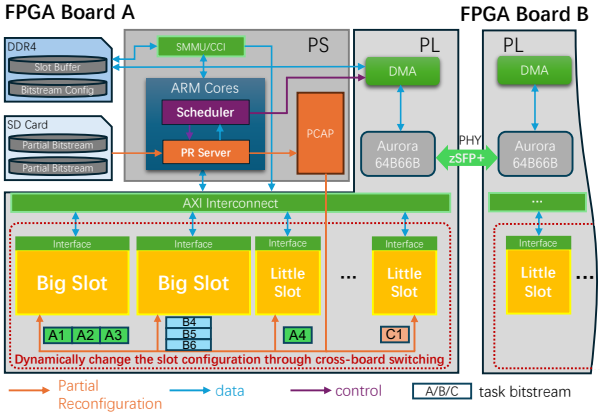


Fig. 1: VersaSlot system with PS and PL in FPGA cluster.

The VersaSlot system, aligned with the typical structure of FPGA sharing design [14], [15], consists of two main components: the embedded hypervisor within the Processing System (PS) and the FPGA fabric layout in the Programmable Logic (PL), as illustrated in Figure 1. The VersaSlot hypervisor in the PS is implemented on ARM cores via bare-metal programming to minimize the system control overhead. Its primary responsibilities include scheduling and resource allocation, managing the loading of partial bitstreams and application data, and coordinating cross-board switching. For FPGA boards without a dedicated CPU, the hypervisor can run on the host CPU and control the FPGA via the PCIe interface.

The VersaSlot hypervisor in the PS comprises two key modules: the scheduler and the PR server, both of which run as bare-metal applications on the ARM cores. At runtime, the scheduler allocates computational slot resources to each incoming application, updates the status of reconfigurable slots, and determines the next application to execute. The PR server, which handles partial reconfiguration, waits for PR requests from the scheduler. It loads pre-generated bitstreams from the SD card into memory and sends commands to the FPGA's

PCAP module to dynamically switch partial bitstreams for applications. However, the PCAP can only perform PR for one bitstream at a time and suspends the associated CPU during the process. Additionally, the PR server communicates status updates to the scheduler through the On-Chip Memory (OCM). The ARM cores utilize the System Memory Management Unit (SMMU) for address mapping and the AXI Interconnect to transfer application data to the FPGA slots for execution.

In the PL, based on the DPR technique, FPGA resources are divided into a *static region* and partial reconfigurable Big and Little slots. The static region configures interfaces for the reconfigurable slots, enabling slots to communicate with the PS and memory via the AXI bus. The region can only be programmed once at system startup, while the reconfigurable parts can be dynamically reconfigured and switched to map different application logic at runtime. Before execution, application bitstreams are prepared offline in advance. Specifically, applications are partitioned into smaller tasks suitable for Little slots by synthesis resources via automated scripts. The ***task*** represents a portion of the application and the basic execution unit for a slot. The VersaSlot system supports the dynamic ***batch*** processing for applications. The fine granularity and batched structure of tasks facilitate an application to be organized in a pipeline with dependencies across slots during execution, which can increase parallelism, thus reducing execution time and enhancing resource utilization. A ***slot*** refers to a reconfigurable region. Traditionally, the slots are designed to be uniform. In the VersaSlot system, we introduce a novel *Big.Little* slot architecture, where three consecutive small tasks can be bundled as a big task for execution in a Big slot. In our system, an FPGA consists of either 2 Big slots and 4 Little slots (*Big.Little*) or 8 Little slots (*Only.Little*), but can be extended to any Big/Little configuration. The resource capacity of each Big slot is twice that of a Little slot. To ensure slot compatibility, the automated script generates partial bitstreams for each task adaptive to each slot. These generated bitstreams are stored on the SD card. Once partial reconfiguration is complete, the scheduler allocates buffers and launches the task.

The cross-board switching module is another important component in the PL, designed for board switching between different FPGA slot configurations. It utilizes the GT transceivers (zSFP+) to connect other FPGA boards. We introduce the Aurora IP core, an efficient communication protocol to transfer tasks, application information, and data directly via DMA to another FPGA unit when switching is activated.

B. Big.Little Slots

The *Big.Little* architecture is a heterogeneous slot design for spatio-temporal FPGA sharing, coupling resource-intensive Big slots and standard-resource Little slots. Although Little slots can increase resource granularity for more applications to share an FPGA, they also bring severe mutual PR contention and task execution blocking problems due to inherent DPR limitation in serial bitstream loading and CPU blocking. For example, as shown in Figure 2, two applications (App-1 and App-2), each with 3 tasks (T1, T2, T3) and batch sizes of 3

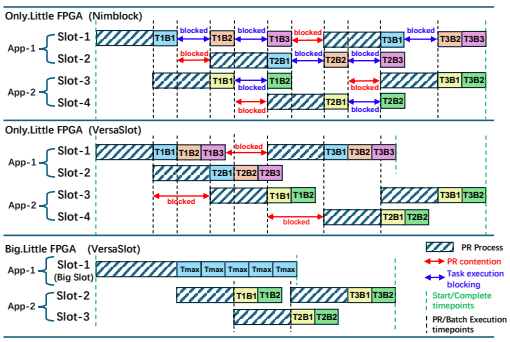


Fig. 2: Versaslot with *Big.Little* and *Only.Little* dual-core scheduling alleviates the PR contention and task execution blocking problems, thus reducing application response time.

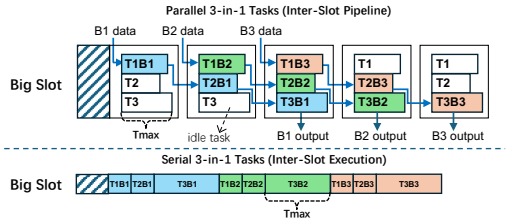


Fig. 3: Parallel and serial bundling for the 3-in-1 task.

(B1, B2, B3) and 2 (B1, B2), are processed on an FPGA with 4 Little slots, in which each application is allocated with 2 slots that forms the execution pipeline. In the Nimblock [15], after the App-1 completes the PR of its first task, the App-2 starts the PR of its first task, during which it blocks the bitstream loading of App-1's second task (T2) although App-1's first batch item of the first task (T1B1) has finished the execution. Subsequently, the PR of App-1's second task (T2) also blocks the PR of App-2's second task. Moreover, since Nimblock handles all operations and scheduling by a single CPU core, the PR process of App-2's T1 also blocks the task execution of App-1's T1B2, and the PR of App-1's T2 blocks the task execution of App-2's T1B2. As the two applications continue running, mutual PR contention and task execution blocking alternate, disrupting each application's original pipeline and significantly increasing response times. The figure shows only 4 slots with 2 applications, but as the number of shared slots and applications increases, this effect becomes more severe.

Within our uniform Little slots (*Only.Little*) FPGA design, VersaSlot decouples PR logic from the scheduler to a dedicated PR server on a separate CPU core, which effectively resolves task execution blocking. As in Figure 2, while other applications perform PR operations, the scheduler can still promptly trigger batch execution of tasks without waiting for specific PR completion points. Additionally, we pre-load tasks of the same application into the slots to further enhance pipeline efficiency. The *Only.Little* design is suited for specific applications with few tasks and large batch sizes, as PR occurs infrequently. But PR contention still persists in most workloads.

Therefore, we propose the Big slots, which can dynamically bundle and load 3 tasks (3-in-1 task) of an application simultaneously for internal execution at runtime. As shown in Figure 2, with VersaSlot's *Big.Little* slots, once App-1 and its tasks

are bundled and scheduled to execute on the Big slot, App-1 only needs to trigger tasks' batch execution without frequent PR loading, thus avoiding outside PR blocking from Little slots. Meanwhile, App-2's tasks can complete its pipeline smoothly without the PR blocking from App-1. Thus, the PR contention of applications in both Big and Little slots is alleviated. Furthermore, as in Figure 3, the tasks within Big slots can be organized in a pipeline as well. Since each batch's parallel execution time on the Big slot equals the longest of the three tasks ($T_{\max}$), serial execution is preferable when $T_{\max} \cdot (N_{\text{batch}}+2) > \sum(T_1+T_2+T_3) \cdot N_{\text{batch}}$. With this criterion, the VersaSlot system can select the optimal 3-in-1 task bitstream for execution at runtime. We set the bundling size to be 3 based on the Big slot's resource capacity to accommodate tasks and its fewer idle task cycles in pipelines than a larger size.

C. Slot Allocation and On-board Scheduling

Similar to the ARM big.LITTLE cores [19], the heterogeneous *Big.Little* slot architecture introduces greater complexity in resource allocation and scheduling. We propose an efficient slot allocation and scheduling algorithm for the architecture. When an application A_i enters the candidate list, it is first added to the waiting list C_{wait} for slot allocation. Once the application acquires slot resources R_{A_i} (the maximum number of slots to use), all of its tasks are pushed into the ready list Q_T, awaiting scheduling for execution in Big or Little slots.

1) Slot Allocation: As detailed in Algorithm 1, slot allocation is based on available slots and system status, featuring the processes of primary allocation and redistribution along with binding and rebinding. In the primary allocation, the system first allocates slots for applications in the waiting list C_{wait}. It prioritizes allocating Big slots to applications that can bundle tasks, as Big slots can significantly mitigate PR contention. After the Big slot allocation, Little slots are assigned to applications based on the optimal slot count $O^L_{A_i}$. The optimal slot allocation, derived through integer linear programming (ILP) as in [14], [15], represents the most efficient slot configuration for pipeline execution within either Little or Big slots, whose value is usually lower than the task count. The primary allocation ensures applications execute promptly along their task pipelines. When additional Little slots are left, Versaslot performs the redistribution based on applications already assigned with Little slots, prioritizing those at the front of the runnable queue to obtain the maximum needed slots. This redistribution effectively avoids slot idling and maximizes overall slot utilization. Meanwhile, to prevent situations where Big slots are idle while Little slots are overburdened, we propose a rebinding method for load balancing. For applications assigned to Little slots but not yet executing, we unbind them for new slot allocation, returning them to a waiting state. This ensures that Big slots can receive new applications for execution. But applications bound to the big slots can only complete all their tasks in the Big slots to avoid Big slot blocking caused by task dependencies.

2) On-board Scheduling: Algorithm 2 outlines the overall scheduling process, which primarily includes 3-in-1 task online bundling, task execution launching, and scheduling tasks

Algorithm 1: Slot Allocation

Input: $C_{wait} = \{A_i\}$: apps A_i waiting for slot allocation;
$S_{Big} = \{A_i\}$: the list of apps allocated with Big slots;
$S_{Little} = \{A_i\}$: the list of apps allocated with Little slots;
B_{avail}: idle Big slots; L_{avail}: idle Little slots; (number);
Q_T: the list of ready tasks waiting for scheduling;
$N_{T_{A_i}}$: the number of unfinished ready tasks of A_i;
$O_{A_i} = (O^B_{A_i}, O^L_{A_i})$: optimal Big/Little slots for A_i's pipeline
Output: $R_{A_i} = (R^B_{A_i}, R^L_{A_i})$: Big/Little slots allocated to A_i

$B_{avail} = B_{total} - \sum N_{T_{A_i}}$, $A_i \in S_{Big} \wedge T_{A_i} \in Q_T$;
`// No slot available, skip the allocation.`
if $B_{avail} \leq 0$ *and* $L_{avail} \leq 0$ **then**
  return;
`// Unbind apps with Little slots for rebinding.`
foreach $A_i \in S_{Little}$ ***and*** $B_{avail} > 0$ **do**
  if *not isAppStarted(A_i)* **then**
    $S_{Little} = S_{Little} \setminus \{A_i\}$; $C_{wait} = C_{wait} \cup \{A_i\}$;
`// Primary allocation.`
$L_{left} = L_{total} - \sum \min(R^L_{A_i}, N_{T_{A_i}}), A_i \in S_{Little} \wedge T_{A_i} \in Q_T$;
foreach $A_i \in C_{wait}$ **do**
  `// Binding, prioritize big slot allocation.`
  if $B_{avail} > 0$ ***and*** *canBundle(A_i)* **then**
    $R_{A_i} = (O^B_{A_i}, 0)$; $S_{Big} = S_{Big} \cup \{A_i\}$;
    $B_{avail} = B_{avail} - 1$; **continue**;
  `// Binding, allocation with Little slots.`
  if $L_{avail} > 0$ *and* $L_{left} > 0$ **then**
    $R_{A_i} = (0, O^L_{A_i})$; $S_{Little} = S_{Little} \cup \{A_i\}$;
    $L_{left} = L_{left} - R^L_{A_i}$;
`// Redistribution for left Little slots.`
if $L_{left} > 0$ **then**
  foreach $A_i \in S_{Little}$ ***and*** $L_{left} > 0$ **do**
    $\delta = N_{T_{A_i}} - O^L_{A_i}$, $T_{A_i} \in Q_T$;
    $R_{A_i} = (0, \min(L_{left} + R^L_{A_i}, \delta + R^L_{A_i}))$;
    $L_{left} = L_{left} - \delta$;
return $\{R_{A_i}\}$, *updated* S_{Big}, S_{Little}, C_{wait};

to their designated slots for PR. For newly added tasks in the ready list Q_T assigned with Big slots, the scheduler bundles them into serial or parallel 3-in-1 tasks at runtime and replaces them with the bundled tasks in the ready list for execution. Meanwhile, the scheduler updates task execution statuses, loads task data, and launches batch executions. In previous work [14], [15], this step was often blocked by subsequent PR processes, leading to the task execution blocking problem. However, VersaSlot employs dual-core scheduling, which decouples the PR process and scheduling logic onto two separate CPU cores. When the scheduler assigns a new task to its corresponding available slot for PR, it sends an asynchronous request to the PR server without waiting for PR completion. Meanwhile, the scheduler limits application slot usage to its allocation, ensuring efficient spatio-temporal FPGA sharing. In VersaSlot's *Only.Little* system, the scheduler follows the preemption mechanism as in [15], to prevent long-running tasks from monopolizing the FPGA. But the preemption brings more PR operations. The *Big.Little* applies preemption only in Little slots since an application cannot occupy both Big and Little slots simultaneously. Meanwhile, the inherent design of the slot redistribution in *Big.Little* also avoids monopolization.

Algorithm 2: Scheduling Algorithm

Input: $S_{Big} = \{A_i\}$, $S_{Little} = \{A_i\}$, $\{R_{A_i}\}$;
B_{avail}: idle Big slots; L_{avail}: idle Little slots; (number);
Q_T: the list of ready tasks waiting for scheduling;
$U_{A_i} = (U^B_{A_i}, U^L_{A_i})$: number of Big/Little slots used by A_i;

```
// Add new tasks to the ready task list.
foreach A_i ∈ S_Big ∪ S_Little do
    if T_{A_i} ∉ Q_T then
        Q_T.push_back(T_{A_i});
// Bundle 3-in-1 task for Big slots online.
foreach A_i ∈ S_Big and {T_{A_i}} ⊂ Q_T do
    if not isBundled({T_{A_i}}) then
        T^{bundled}_{A_i} = 3in1Bundle({T_{A_i}});
        Q_T replaces {T_{A_i}} with {T^{bundled}_{A_i}};
// Launch batch execution for running tasks.
foreach T_{A_i} ∈ Q_T do
    if isRunning(T_{A_i}) and waitBatchExec(T_{A_i}) then
        launch(T_{A_i});
    else if isFinished(T_{A_i}) then
        Q_T = Q_T \ T_{A_i}; isFinished(A_i) ⇒ S = S \ A_i;
// Schedule tasks to corresponding slots.
foreach T_{A_i} ∈ Q_T do
    if A_i ∈ S_Big and U^B_{A_i} < R^B_{A_i} and B_avail > 0 then
        PR(T_{A_i}) to an idle Big slot (Async in PR server);
        T_{A_i}.running = True;  B_avail = B_avail − 1;
    if A_i ∈ S_Little and U^L_{A_i} < R^L_{A_i} and L_avail > 0 then
        PR(T_{A_i}) to an idle Little slot (Async in PR server);
        T_{A_i}.running = True;  L_avail = L_avail − 1;
```

D. Cross-Board Switching and Live Migration

1) Live Migration: As mentioned in section III-B, *Only.Little* and *Big.Little* configurations have their own advantages for different application workloads, requiring an FPGA to switch between them to maximize performance. However, configurations related to slot size and interfaces are located in the *static region*, which can only be programmed once during system start-up. Cross-board switching enables seamless transitions between configurations without rebooting, achieved through live workload migration. When switching is triggered, the original FPGA stops executing new tasks. Applications and tasks in the ready list, along with their buffers, are transferred via DMA to a new pre-configured FPGA. Once transferred, the new FPGA resumes task execution and processes upcoming new workloads. Meanwhile, ongoing tasks on the original FPGA continue to completion to avoid bitstream reloading overhead, and the FPGA is freed afterward to prevent excess resource usage. Therefore, in a cluster, a single available FPGA can enable cross-board switching for the entire system.

2) Performance Degradation Metric D_{switch}: To determine the optimal timing for the switching and evaluate system performance degradation from PR contention, we propose an evaluation metric, D_{switch}, considering all influencing factors.

$$D_{\text{switch}} = \frac{N_{\text{blocked_tasks}}}{N_{\text{PR}}} \times \frac{N_{\text{apps}}}{N_{\text{batch}}}, \quad (0 < D_{\text{switch}} < 1) \tag{1}$$

where $N_{\text{PR}} = \sum_{A_i} N_{T_{A_i}}$, $A_i \in R_c \cup R_s$; $N_{\text{batch}} = \sum_{A_i} B_{A_i}$, $A_i \in C$. The metric D_{switch} combines the ratio $\frac{N_{\text{blocked_tasks}}}{N_{\text{PR}}}$ to evaluate current PR contention degree among tasks, and the ratio $\frac{N_{\text{apps}}}{N_{\text{batch}}}$ to assess potential conflicts from candidate applications. It is normalized from 0 to 1, where higher values indicate more severe contention and less predictable application response time. The metric is recalculated after every n updates to the application candidates queue with total completed (R_c) / running (R_s) / ready (R_r) applications. The $N_{\text{blocked_task}}$ represents the number of tasks blocked by PR contention during this period. The N_{PR} denotes the number of PR tasks $\{T_{A_i}\}$ by all completed and running applications A_i. The $N_{\text{blocked_tasks}}$ is positively correlated with N_{PR}, which indicates a higher number of PR typically results in more blocked tasks and an increasing D_{switch}. Another two influential factors, batch size B_{A_i} and the number of applications N_{apps}, are significantly factored into the future contention evaluation. The N_{batch} is the total batch size of all applications A_i in the current application candidate queue C. If each application is allocated only one slot with batch size to be one, $N_{\text{batch}} = N_{\text{apps}}$, this indicates the worst-case scenario for PR conflicts and corresponds to the maximum value of D_{switch}.

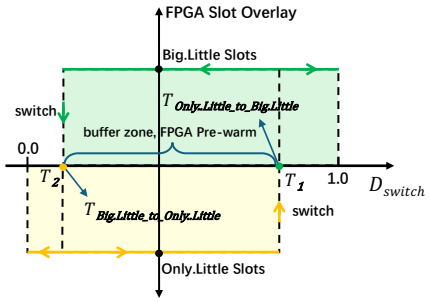


Fig. 4: Cross-board switching loop with the buffer zone.

Inspired by Schmitt Trigger [23], we propose a switch loop for seamless cross-board switching across multiple FPGAs. As shown in Figure 4, as the D_{switch} increases and reaches the threshold T_1, the system triggers the switching from the *Only.Little* slots to the *Big.Little* slots, thereby alleviating the PR contention. Similarly, when the D_{switch} decreases to the threshold T_2, it triggers the switching from the *Big.Little* FPGA back to *Only.Little* FPGA, allowing more future applications to share an FPGA. The two user-configurable thresholds adjust switching sensitivity, with a buffer zone to prevent frequent switching and ensure system stability. Meanwhile, when the D_{switch} metric enters the buffer zone, the system anticipates the direction of change, pre-configures the potential FPGA configuration, and loads task bitstreams into SD storage in a new FPGA. The pre-warming design further ensures seamless live migration with low overhead.

IV. Experiment and Evaluation

The VersaSlot system was evaluated on an FPGA cluster with two latest Xilinx UltraScale+ FPGAs (ZCU216). We used the same benchmark as previous work [15] and partitioned applications into multiple tasks based on the optimal fit between slot resources and task resource usage after synthesis. The applications include 3D Rendering (3DR, 3 tasks), LeNet (6 tasks), Image Compression (IC, 6 tasks), AlexNet (AN, 6 tasks), and Optical Flow (OF, 9 tasks). The partitioning and task bitstream generation were performed automatically

by a TCL script in Vivado 2024.1. We compared the VersaSlot system with four existing scheduling algorithms/systems, including the traditional exclusive temporal multiplexing (Baseline) [7], [16], First-come-first-served spatio-temporal sharing (FCFS), round-robin (RR) from [22], and the state-of-the-art spatio-temporal sharing system Nimblock [15]. To simulate different congestion conditions, we randomly generated application workloads (10 sequences, 20 apps/sequence) with random batch size (5-30) and arrival intervals, including Loose (5000ms), Standard (1500ms-2000ms), Stress (150ms-200ms), and Real-time (50ms) for the evaluation.

A. Response Time Reduction and Tail Latency

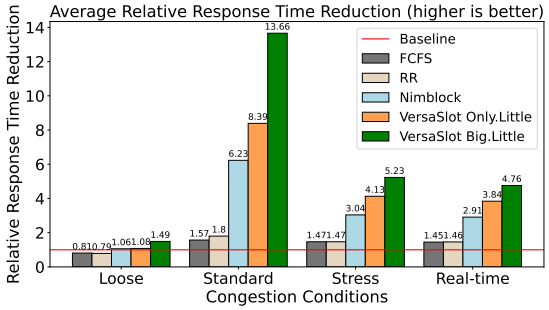


Fig. 5: Relative response time reduction under different congestion conditions, normalized to the baseline.

As shown in Figure 5, we analyze the relative response time reduction under different congestion conditions. The VersaSlot *Big.Little* design outperforms all other methods across all congestion conditions. Under standard workloads, it outperforms the baseline up to 13.66x. The FCFS and Round-robin perform slightly better than the baseline. Compared to the state-of-the-art system, Nimblock, the *Big.Little* design reduces average response time up to 2.17x in standard workloads, 1.72x under stress, and 1.63x in real-time conditions, showcasing the efficiency of *Big.Little* architecture and dual-core scheduling in fine-grained spatio-temporal FPGA sharing. Additionally, compared to the VersaSlot *Only.Little* configuration, *Big.Little* achieves average response time improvements by 63%, 27%, and 24%, respectively, verifying its effectiveness in alleviating PR contention. Meanwhile, as shown in Figure 6, we further analyze the tail latency of applications, and the *Big.Little* design consistently outperforms Nimblock in P95 and P99 metrics (95th and 99th percentiles) across all congestion conditions. Under stress workloads, *Big.Little* improves these metrics by 83% and 46%, respectively, and by 56% and 48% under real-time workloads. While *Big.Little* shows a slight increase in P99 tail latency compared to the baseline, it maintains or even improves P95 performance. Given its substantial advantage in response time, *Big.Little* obviously demonstrates superior application performance than the baseline.

B. Resource Utilization

Figure 7 shows the resource utilization improvement of 3-in-1 tasks in Big slots compared to running all tasks only in Little slots. All applications show significant LUT and FF utilization increases, with average improvements of 35% and 29%. Specifically, the right figure displays the LUT usage for

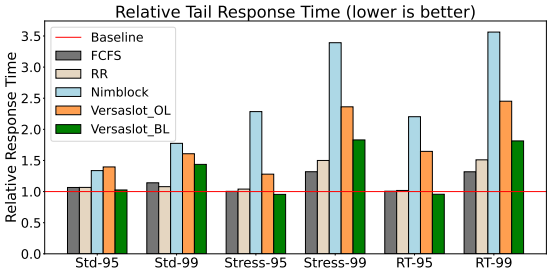


Fig. 6: Tail response time normalized to the baseline.

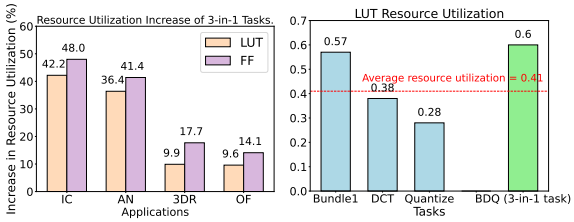


Fig. 7: Resource utilization improvement by 3-in-1 tasks.

the first three tasks of application IC and their bundled 3-in-1 task. Since task partitioning is normally based on synthesis resource usage, Bundle1's LUT utilization decreases from 0.98 in synthesis to 0.57 in implementation. But, with bundling, the 3-in-1 task improves average utilization from 0.41 to 0.6.

C. Cross-board Switching

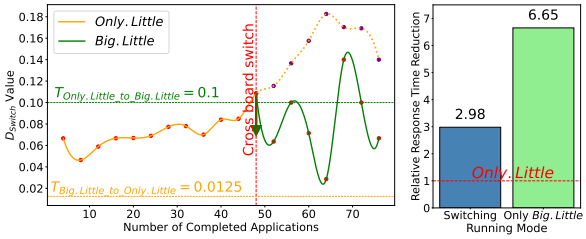


Fig. 8: D_{switch} and relative response time with switching.

To validate the effectiveness of cross-board switching, we tested three long workloads, each with 80 applications and standard arrival intervals. Figure 8 (left) shows the D_{switch} variation for one workload every 4 application updates, where the D_{switch} value triggers a switch from *Only.Little* to *Big.Little* at the threshold, enabling tasks to execute on the new Big.Little FPGA. Compared to execution solely on *Only.Little*, cross-board switching reduces the average response time up to nearly 3x with the average switching overhead of 1.13ms.

V. Conclusion

This paper presents VersaSlot, an efficient spatio-temporal FPGA sharing system with the novel *Big.Little* slot architecture to address the PR contention problem. We propose a slot allocation and dual-core scheduling algorithm and a seamless cross-board switching and live migration mechanism for the architecture to maximize FPGA multiplexing in the cluster. VersaSlot achieves up to 2.19x average response time improvement over the state-of-the-art system and enhances the LUT and FF utilization by 35% and 29% on average.

Acknowledgment

We sincerely thank Leibniz Supercomputing Centre (LRZ) and Munich Quantum Valley (MQV) for the FPGA support.

REFERENCES

[1] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[2] T. Wang, L. Gong, C. Wang, Y. Yang, Y. Gao, X. Zhou, and H. Chen, "Via: A novel vision-transformer accelerator based on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4088–4099, 2022.

[3] C. Wu, T. Geng, A. Guo, S. Bandara, P. Haghi, C. Liu, A. Li, and M. Herbordt, "Fasda: An fpga-aided, scalable and distributed accelerator for range-limited molecular dynamics," in *SC23: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–14.

[4] S. Zeng, Z. Zhu, J. Liu, H. Zhang, G. Dai, Z. Zhou, S. Li, X. Ning, Y. Xie, H. Yang, and Y. Wang, "Df-gas: a distributed fpga-as-a-service architecture towards billion-scale graph-based approximate nearest neighbor search," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 283–296.

[5] J. Zhao, Y. Zhang, J. Cheng, Y. Wu, C. Ye, H. Yu, Z. Huang, H. Jin, X. Liao, L. Gu, and H. Liu, "Sagraph: A similarity-aware hardware accelerator for temporal graph processing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[6] Alibaba Cloud, "What is FPGA as a Service? - FPGA as a Service," https://www.alibabacloud.com/help/en/fpga-as-a-service/product-overview/what-is-fpga-as-a-service, [Accessed 27-10-2024].

[7] Amazon AWS EC2, "Amazon EC2 F1 Instances," https://aws.amazon.com/ec2/instance-types/f1/, [Accessed 27-10-2024].

[8] Microsoft Azure, "Microsoft Azure Stack Edge Pro FPGA overview," https://learn.microsoft.com/en-us/azure/databox-online/azure-stack-edge-overview, [Accessed 27-10-2024].

[9] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The future of fpga acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 3, Feb. 2022.

[10] A. Damiani, G. Fiscaletti, M. Bacis, R. Brondolin, and M. D. Santambrogio, "Blastfunction: A full-stack framework bringing fpga hardware acceleration to cloud-native applications," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 2, pp. 1–27, 2022.

[11] B. Ringlein, F. Abel, D. Diamantopoulos, B. Weiss, C. Hagleitner, M. Reichenbach, and D. Fey, "A case for function-as-a-service with disaggregated fpgas," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 333–344.

[12] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, "Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 107–127.

[13] Y. Zha and J. Li, "Virtualizing fpgas in the cloud," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 845–858.

[14] A. Dhar, E. Richter, M. Yu, W. Zuo, X. Wang, N. S. Kim, and D. Chen, " DML: Dynamic Partial Reconfiguration With Scalable Task Scheduling for Multi-Applications on FPGAs ," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2577–2591, Oct. 2022.

[15] M. Mandava, P. Reckamp, and D. Chen, "Nimblock: Scheduling for fine-grained fpga sharing through virtualization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023.

[16] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 13–24.

[17] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network fpga clusters in a heterogeneous cloud data center," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 237–246.

[18] Xilinx, "Dynamic function exchange," https://docs.amd.com/v/u/2020.1-English/ug909-vivado-partial-reconfiguration, [Accessed 08-11-2024].

[19] ARM big.LITTLE, "Arm Cortex-a Series, Global Task Scheduling," https://developer.arm.com/documentation/den0013/d/big-LITTLE/Software-execution-models-in-big-LITTLE/Global-Task-Scheduling, [Accessed 08-10-2024].

[20] AMD Xilinx, "AMD Technical Information Portal - dynamic function exchange," https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Introduction, [Accessed 25-10-2024].

[21] AMD, "AMD Technical Information Portal - pcap," https://docs.amd.com/r/en-US/ug1085-zynq-ultrascale-trm/Initialize-PCAP-Interface, [Accessed 25-10-2024].

[22] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on FPGAs?" in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 991–1010. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/roscoe

[23] "Schmitt-Trigger – Wikipedia — de.wikipedia.org," https://de.wikipedia.org/wiki/Schmitt-Trigger, [Accessed 30-10-2024].

SplitSync: Bank Group-Level Split-Synchronization for High-Performance DRAM PIM

Byungkuk Yoon, Sanghyeok Han, Gyeonghwan Park, and Jae-Joon Kim
Seoul National University, Republic of Korea
[bkyoon, hansh778, gyeonghwan.park, kimjaejoon]@snu.ac.kr

***Abstract*—Processing in Memory (PIM) architectures enhance memory bandwidth by utilizing bank-level parallelism, typically implemented with a SIMD structure where all banks operate simultaneously under a single command. However, this synchronous approach requires the activation of all banks before computation, leading to activation times that exceed computation times, limiting performance gain. Recently, asynchronous execution PIM has been proposed as an alternative, allowing banks to operate asynchronously and overlap activation with processing to hide the row activation overhead. While effective at reducing row activation overhead, the independent operation requires large shared accumulators for each bank group, increasing area overhead. To address the issues, we propose bank group (BG)-level split synchronization DRAM PIM, where each bank group operates asynchronously to hide row activation overhead while operating synchronously within the bank group to eliminate the need for shared accumulators. Evaluation results show that our proposed design achieves an average throughput improvement of 1.70x and 1.06x compared to conventional PIM and asynchronous execution PIM. Furthermore, the area overhead per processing unit (PU) increases by only 1.5% compared to conventional PIM and is significantly lower than that of asynchronous execution PIM.**

I. Introduction

DRAM Processing-in-memory (PIM) has emerged as a promising approach to accelerate memory-intensive workloads by harnessing bank-level parallelism (BLP) [20, 4, 9, 18, 2, 19]. This technique leverages the inherent parallelism of DRAM banks to enhance memory bandwidth, making it well-suited for tasks like neural network acceleration, particularly General Matrix-Vector Multiplication (GEMV) [20, 13, 14]. Despite its potential, the practical performance of DRAM PIM often falls short due to the overhead associated with memory operations, such as row activation/precharge, input writing, and result reading. A critical bottleneck in achieving full BLP in DRAM PIM is the t_{FAW} (Four Activate Window) constraint, which limits the number of banks that can be activated within a specified time frame. This limitation is designed to prevent voltage drops within DRAM but introduces substantial row activation overhead when activating multiple banks, negatively impacting overall throughput.

Recent approaches have sought to mitigate this constraint. For example, DRAM PIMs in [14, 11, 12] leverage circuit-level innovations by incorporating reservoir capacitors to enable simultaneous activation of 16 banks, effectively bypassing the t_{FAW} limitation. Let us call this DRAM PIM style as ACT16-PIM in this paper. While this solution improves performance, it comes at the cost of significant area overhead due to the additional reservoir capacitors. Alternatively, asynchronous execution PIM architecture [8] activates and processes rows across banks in a staggered manner, overlapping activation and computation to reduce overhead. However, this method complicates input data distribution, requiring large shared accumulators for gathering and accumulating partial results across banks, leading to increased area and design complexity.

To address these limitations, we propose SplitSync PIM, a DRAM-PIM architecture that combines the benefits of synchronous and asynchronous operations. Within each bank group, synchronous operations enable efficient input broadcasting and computation without the need for shared accumulators, thereby minimizing area overhead. Meanwhile, asynchronous operation between bank groups allows activation and computation phases to overlap, significantly reducing row activation overhead while maintaining compliance with the t_{FAW} constraint. This design also eliminates the need for large reservoir capacitors and reduces area complexity compared to ACT16-PIM. Additionally, we introduce modifications to the I/O architecture that allow independent operations across bank groups, mitigating I/O conflicts and enabling simultaneous execution of specific operation pairs: input writing with input broadcasting, input broadcasting with another input broadcasting, and input broadcasting with result reading.

Our main contributions are as follows:

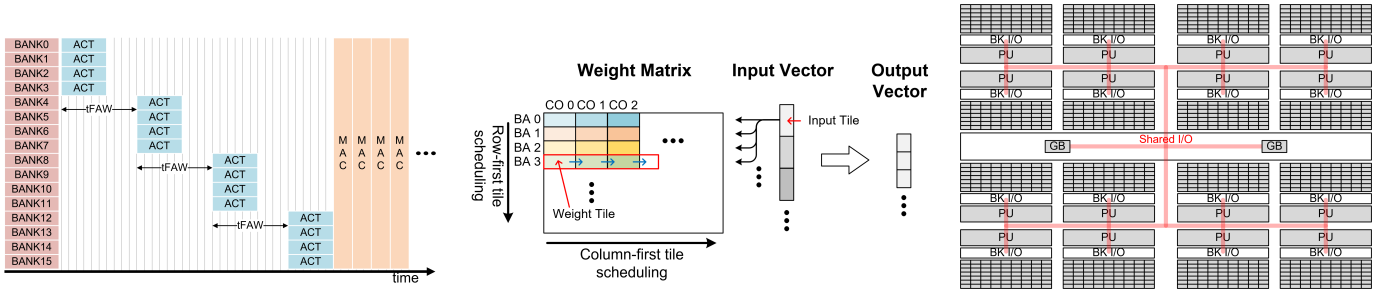
- We propose a SplitSync DRAM PIM architecture that performs synchronous computations within bank groups and asynchronous operations between bank groups. The synchronous operations facilitate implementation with minimal area overhead, while asynchronous operations between bank groups effectively reduce row activation overhead.
- To resolve I/O conflicts during asynchronous operations, we introduce new I/O architecture to support independent operations between bank groups, thereby improving overall performance.
- We compare SplitSync with previous DRAM PIM architectures using several neural network benchmarks and demonstrate that it provides higher throughput than previous works by a large margin under the same area budget.

II. Background

A. Operation and Architecture of Conventional DRAM PIM

Conventional DRAM PIM [15, 13, 14, 11, 4, 20] operations extend memory bandwidth by exploiting Bank-Level Parallelism (BLP), which accelerates memory-intensive workloads. To maximize bank-level parallelism, all DRAM banks operate concurrently within a SIMD (Single Instruction, Multiple Data) architecture. As shown in Fig. 1a, in order to access all banks simultaneously, all banks are activated before the PIM operation begins. After activation, the columns of the activated rows are accessed sequentially to perform the computation. Since the host controls the DRAM through DRAM commands, new DRAM commands need to be introduced to manage PIM operations and ensure smooth execution.

The main focus of PIM is on General Matrix-Vector Multiplication (GEMV) operations, a key component of neural network computation. GEMV is a memory-bound operation, where a large weight matrix, typically not reused, must be loaded from memory to process a relatively small input vector. Fig. 1b illustrates the tiling method used for GEMV computation in conventional PIM. The rows of the weight matrix are mapped to DRAM banks, tiled according to DRAM row size, and then stored in the corresponding banks and rows. The input vector is also tiled to match the DRAM row size and written into the global buffer (GB) in preparation for computation. GEMV performs MAC (Multiply-Accumulate) operations, sequentially multiplying the weight tiles by the input vector for each bank and accumulating



(a) Conventional PIM command timing (b) GEMV operation in conventional PIM (c) Conventional PIM architecture

Fig. 1: Conventional DRAM PIM architecture and its operation

the results in the result latch. Once the computation for one tile is finished, the process repeats for the next tile (corresponding to the next DRAM row) until all computations are completed.

The tile scheduling for GEMV computation can be classified into row-first or column-first tile scheduling, depending on how the tiles are selected during computation. In column-first tile scheduling, the tile moves to the next matrix column along the matrix row, and after completing operations on one matrix row, it moves to the next matrix row. The advantage of this scheduling is that partial sums can be continuously accumulated in the result latch to compute the final sum. However, since the input is used only within each tile, frequent input writes are required, which increases power consumption and introduces writing overhead, degrading performance. In contrast, row-first tile scheduling moves the tile along the weight columns, enabling reuse of the input vector. In this case, either new result latches are required for each tile, or the partial sums must be read out by the host. Although partial sums must be read out after each tile operation, row-first scheduling offers performance and power advantages because the partial sums are typically much smaller than the input tile. This smaller data size allows the host to handle aggregation while the PIM continues processing the next tile.

Fig. 1c shows the architecture of a conventional DRAM PIM. To perform the GEMV operation, processing units (PUs) are placed adjacent to each bank. These PUs receive weights from their respective banks and inputs from the global buffer to execute MAC operations. Since the input vector is shared across all PUs, it is broadcast via shared DRAM I/O, while the weights are delivered through local I/O from each bank. The global buffer, implemented in SRAM, is positioned on either side of the peripheral circuit area, typically sized to match the DRAM row. The results accumulated in the PUs are read out through the DRAM read path via shared I/O. Since the result of each PU is smaller than the column data width, the results from multiple PUs are aggregated and output together.

B. Challenges of DRAM PIM Operation

While DRAM PIM can boost memory bandwidth through bank-level parallelism, it often falls short of expected performance levels in practice due to significant overheads, such as input writes, row activation/precharge, and result reads required for PIM operations. In GEMV operations, where activated DRAM rows are not reused, the repeated all-bank activation and precharge for each tile make row activation/precharge overhead particularly critical.

The time required for all-bank activation is determined by two timing parameters, t_{RRD} and t_{FAW}. t_{RRD} represents the minimum time interval between activation commands, while t_{FAW} defines the minimum timing window that allows up to four activation commands. Let t_{ACT} denote the time required for all-bank activation, which can be expressed in terms of the aforementioned timing parameters:

TABLE I: Timing parameters and memory configuration for row activation overhead calculation

	Parameter	Value
Timing Parameters	t_{RRD}	9
	t_{FAW}	32
	t_{CCD}	2
	t_{RCD}	24
	t_{RP}	24
Memory Configuration	N	16
	Columns	64

$$t_{\text{ACT}} = \max(t_{\text{FAW}}, t_{\text{RRD}}) \times \left(\left\lceil \frac{N}{4} \right\rceil - 1 \right) + t_{\text{RCD}}$$

Here, t_{RCD} represents the minimum time interval between activation and read commands, and N denotes the number of banks in PIM. When the MAC operation follows read timing, the time for a single tile MAC operation, denoted as t_{MAC}, can be expressed as:

$$t_{\text{MAC}} = t_{\text{CCD}} \times (\text{columns} - 1)$$

where t_{CCD} is the minimum time interval between two read or write commands, and columns is the number of columns in a DRAM row.

In a single tile operation, the row activation/precharge overhead can be represented as:

$$\frac{t_{\text{ACT}} + t_{\text{RP}}}{t_{\text{ACT}} + t_{\text{MAC}} + t_{\text{RP}}}$$

where t_{RP} refers to the precharge time. Based on the timing parameters in Table I, this overhead is found to be 52.9%, indicating that more time is spent on activation/precharge than on computation.

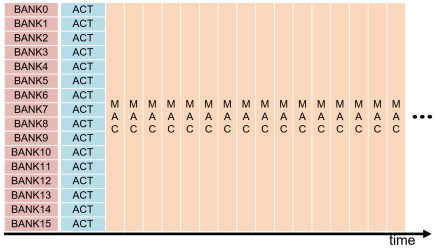
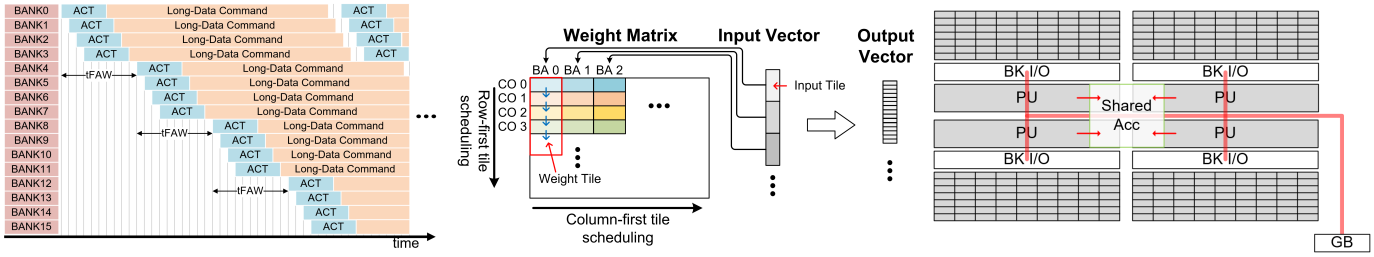


Fig. 2: ACT16-enabled DRAM PIM command timing

C. ACT16-enabled DRAM PIM

A DRAM PIM architecture has been introduced to reduce row activation overhead by eliminating the t_{FAW} constraint at the circuit level, enabling the simultaneous activation of all 16 banks (ACT16) (Fig. 2) [14, 11]. DRAM operates using various internal voltages, and certain timing parameters, such as t_{FAW}, are constrained to prevent failures due to internal voltage drops. In this design, additional reservoir capacitors are placed near the processing units to mitigate internal voltage drops caused by the simultaneous activation of all



(a) Asynchronous execution PIM command timing (b) GEMV operation in asynchronous execution PIM (c) Asynchronous execution PIM architecture

Fig. 3: Asynchronous execution DRAM PIM architecture and its operation [8]

16 banks. As a result, the ACT16 approach enables PIM operation without the t_{FAW} constraint but it incurs significant area overhead. While the exact area of the reservoir capacitors has not been provided, we estimated their area from the die photo, and estimate that the area overhead per PU is approximately 20–25%. The GEMV operation and overall architecture in this PIM design remain unchanged from those in conventional DRAM PIM.

D. Asynchronous Execution PIM

Recently, asynchronous execution PIM architecture has been proposed as another solution to reduce row activation overhead [8]. In this architecture, as shown in Fig. 3a, each bank operates asynchronously, reducing row activation overhead by overlapping row activation and operations across different banks. Since a single command controls only one bank, a long-data command is proposed to avoid command conflict. This command automatically processes operations across an entire row and is issued once at the start of the computation to each banks.

Fig. 3b illustrates the GEMV operation in this architecture. Unlike conventional PIM, small blocks of the weight matrix row are mapped to individual banks, arranged along the matrix columns. Since each bank operates at different timings, broadcasting input data simultaneously to all banks is not feasible. Instead, each bank receives and processes its designated data block independently, reusing it across the entire DRAM row with the long-data command. As operations progress across columns, accumulation cannot occur within individual banks but should take place between banks. To address this, a shared accumulator is placed at the center of each bank group, as shown in Fig. 3c. The shared accumulator receives data asynchronously transmitted from each bank, identifies the values to be added based on the address information, and performs the accumulation.

In this architecture, the operation of a single row in a single bank takes $t_{\text{RCD}} + t_{\text{MAC}} + t_{\text{RP}}$, and t_{ACT} is given by:

$$t_{\text{ACT}} = \max(t_{\text{FAW}}, 4 \times t_{\text{RRD}}) \times \left(\left\lceil \frac{N}{4} \right\rceil - 1 \right) + \left[(N-1) \mod 4 \right] \times t_{\text{RRD}}$$

Thus, the total execution time for M tiles is $(t_{\text{RCD}}+t_{\text{MAC}}+t_{\text{RP}}) \times M + t_{\text{ACT}}$, and the impact of t_{RRD} and t_{FAW} diminishes as M increases.

Despite demonstrating good performance, this architecture has several implementation challenges.

First, the size of the shared accumulator scales with the DRAM row size. According to [8], the shared accumulator incurs an area overhead of approximately 10% per PU based on a 1KB row size. For memory technologies such as GDDR6 [7], which feature row sizes of 2KB or larger, and DDR5 [3]/LPDDR5 [17], which can also support 2KB row size, the area overhead increases to 15-20%.

Second, positioning the shared accumulator centrally within a bank group presents additional challenges. Banks within the bank group share address/data buses and various control circuits. This shared infrastructure requires that bus distribution, data gathering circuits, and control circuits already occupy most of the central space in the bank group. Consequently, integrating a shared accumulator in this central area creates competition for space, making it difficult to prioritize its placement over essential DRAM operation blocks.

Third, asynchronous operations introduce potential conflicts in shared I/O during data transfers. Specifically, conflicts can occur when loading input blocks from the global buffer to each bank and when reading computation results from the shared accumulators. After a computation is completed and banks prepare to move to the next tile, those that have finished must wait for others to complete. This results in idle periods until all banks synchronize and results are read out, allowing the next tile to load. Conversely, synchronous operations ensure that all banks handle input and output simultaneously, eliminating these conflicts. As a result, asynchronous operations can introduce significant overhead, particularly with row-first tile scheduling, though column-first tile scheduling can help mitigate this issue.

III. SplitSync DRAM PIM Design

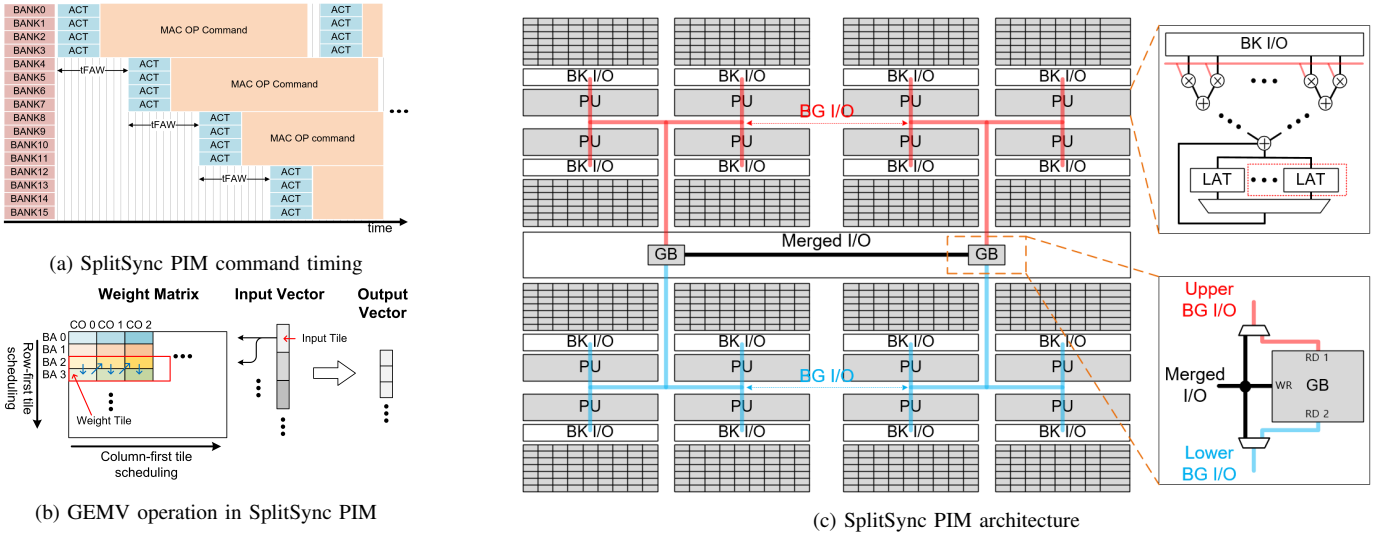
A. Bank Group-Level Split Synchronization

As discussed in the previous section, conventional DRAM PIM experiences significant performance degradation due to row activation overhead. In contrast, both ACT16-enabled PIM and asynchronous execution PIM reduce this overhead but at the cost of increased area requirements. Specifically, ACT16-enabled PIM needs substantial space for reservoir capacitors, while asynchronous execution PIM demands a large shared accumulator. To overcome the performance and area challenges, we propose a bank group-level split synchronization architecture that effectively reduces row activation overhead with minimal area overhead. This architecture operates synchronously within a bank group and asynchronously between bank groups, as illustrated in Fig. 4a.

The synchronous operation within a bank group provides two main advantages. First, since each bank group consists of four banks, all banks within the group can be activated concurrently without being constrained by t_{FAW}. Second, synchronous operations within a bank group enable the global buffer to broadcast input data across the banks. This capability allows the rows of the weight matrix to be mapped to banks in a manner similar to conventional PIM, facilitating intra-bank accumulation and removing the need for a shared accumulator. Moreover, the asynchronous operation between bank groups allows for overlapping row activations and operations across different bank groups, which helps to further reduce overhead, as previously mentioned.

B. SplitSync DRAM PIM Architecture

The SplitSync PIM architecture is shown in Fig. 4c. There are three major changes from conventional PIM designs. First, the I/O



(a) SplitSync PIM command timing

(b) GEMV operation in SplitSync PIM

(c) SplitSync PIM architecture

Fig. 4: Proposed SplitSync DRAM PIM architecture and its operation

architecture is updated from a shared I/O to a combination of bank-group I/O (BG I/O) and merged I/O. Second, the global buffer is now connected to the BG I/O, enabling input broadcasting through the BG I/O. Third, the number of result latches per PU is increased from 1 set to 4 sets, with each set consisting of 16 latches in BF16 format. Here, 1 set refers to 1 unit.

Resolving the I/O conflict Due to the different operation timings between bank groups, input broadcasting is not possible on shared I/O. To address the issue, we introduce bank group-specific I/O, called BG I/O (red and cyan lines in Fig. 4c), which is shared within each BG and supports data movement between bank groups, allowing different inputs to be broadcast to each bank group. The BG I/O is connected to the global buffer and facilitates localized broadcasting, but it does not connect directly to the DRAM interface; instead, it links through the merged I/O. The merged I/O is positioned in the peripheral circuit area and aggregates the BG I/Os (black line in Fig. 4c), enabling seamless access to banks during read and write operations. This architecture ensures that read and write operations at the bank group level do not disrupt PIM operations in other bank groups. As a result, as shown in Fig. 5, these updates to the I/O architecture enables simultaneous operations such as input writing with input broadcasting, input broadcasting with another input broadcasting, and input broadcasting with result reading.

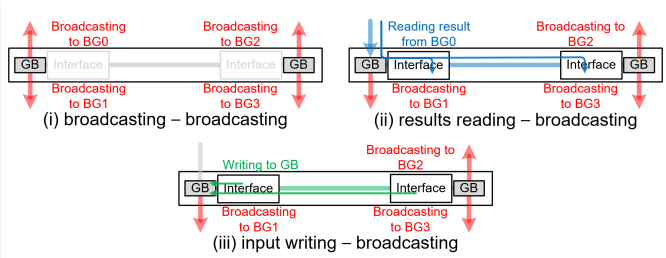


Fig. 5: Simultaneous operations in SplitSync DRAM PIM

GEMV operation in SplitSync PIM The method for tiling GEMV in SplitSync PIM is illustrated in Fig. 4b. While the GEMV tiling approach is fundamentally similar to that of conventional PIM, two key distinctions arise due to the architectural changes in SplitSync PIM.

First, the weight matrix is divided into half and mapped across the left and right bank groups, with the input vector split accordingly between the left and right GBs. This is necessary because the GB is horizontally divided, and each bank group is connected to only one of the GBs via BG I/O (Fig. 4c). Since the connections between GBs uses merged I/O, broadcasting input data from the opposite-side GB that is not connected via BG I/O can cause inefficiencies and potential I/O conflicts. To avoid these issues, the input vector can be either duplicated across both GBs or partitioned between them, with partitioning generally being the more efficient solution.

Second, more than two weight matrix rows are mapped to a single DRAM row to address the mismatch between DRAM row size and GB size. In conventional PIM architectures, even when the GB is split, it functions as a unified large GB during computation. However, in SplitSync PIM, the split GBs operate independently, reducing the GB size available to each bank group. This creates a mismatch between DRAM row size and GB size. To manage this, more result latches are required. Nonetheless, this approach is still more efficient than increasing the GB size. This mapping approach also reduces the frequency of input broadcasting, improving power efficiency. Additionally, increasing the number of result latches enables multiple matrix rows to be stored consecutively within a single DRAM row. This is particularly useful for operations involving numerous small-dimension GEMV operations, such as those found in the multi-head attention mechanism of transformer decoders [22]. In these cases, if there are insufficient result latches, the operation must pause during computation to perform intermediate readouts, which can degrade performance. In SplitSync PIM, four sets of latches are assigned to each PU to allow efficient readout at the BG level. With each PU outputting four units in BF16 format, and four banks per bank group, the total output is 256 bits per bank group, aligning with the column width. The column width is the amount of data transferred in a single read operation.

MAC operation command Since SplitSync PIM enables independent operation of each bank group, command conflicts must be considered. Theses conflicts can be managed in a manner similar to the long-data command used in asynchronous execution PIM. However, unlike the long-data command, the MAC operation command in SplitSync PIM also convey information about how many matrix rows(2, 4, or 8) are stored in each DRAM row. The MAC

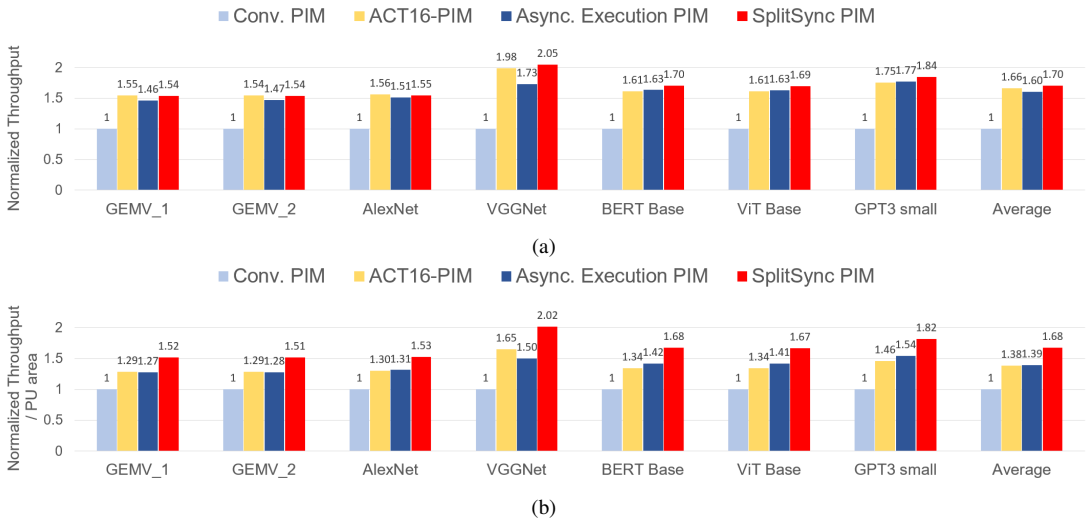


Fig. 6: Throughput comparison across benchmarks. (a) Normalized to conventional DRAM PIM (b) Normalized throughput per PU area.

operation command allows accessing either the entire row or half, which is crucial for improving resource utilization when dealing with matrices with small dimensions or tiled fragments. Once a bank group is activated, the MAC operation command is issued, enabling sequentially access to columns within the row and initiating computations.

IV. Evaluation

A. Methodology

Seven microbenchmarks are used to evaluate the performance of four PIM architectures (Table II). Specifically, two microbenchmarks are assigned to each of the GEMV and CNN models [10, 21], and three to Transformer models [5, 6, 1], focusing solely on matrix operations that PIM can handle in each model. For the CNN model, convolution opertaions are converted into General Matrix-Matrix Multiplication (GEMM), which is then executed through repeated GEMV operations.

The performance evaluation is conducted by modifying the cycle-level DRAMsim3 [16]. All simulations are based on the same GDDR6 configuration (Table II bottom). Since the performance of each PIM can vary depending on tile scheduling, the most optimal scheduling method is applied for each PIM: Conventional PIM, ACT16-enabled PIM and SplitSync PIM use row-first tile scheduling, while Asynchronous Execution PIM uses column-first tile scheduling.

TABLE II: Benchmark and GDDR6 PIM Configuration

Benchmark	**Target Layer**
GEMV1	2K × 2K GEMV
GEMV2	4K × 4K GEMV
AlexNet	5 CONV, 3 FC
VGGNet	13 CONV, 3 FC
BERT Base	12 TRANS
ViT Base	12 TRANS
GPT-3 small	12 TRANS
GDDR6 PIM Configuration	
Bank groups	4 bank groups per channel
Banks	4 banks per bank group
Column width	256 bits
Row Size	2 KB
Density	4 Gb
Clock frequency	1.5 GHz
Global buffer	2 KB

B. Performance

Fig. 6a shows a throughput comparison across seven microbenchmarks, with all results normalized to the throughput of conventional PIM. The three PIM architectures designed to reduce row activation overhead consistently outperform conventional PIM in all cases. Although the extent of performance improvement varies depending on the benchmark, SplitSync PIM shows the highest performance on average. The performance improvement is more pronounced in neural network models than GEMV benchmarks due to varying matrix dimensions within the models. The two GEMV benchmarks have matrix dimensions that align perfectly with the banks and rows during tiling, resulting in efficient use of DRAM resources. In contrast, neural network models often have matrix dimensions that do not map perfectly to DRAM rows or banks, leading to partial utilization of activated rows and higher row activation overhead. Additionally, when the matrix column size is considerably smaller than the DRAM row size, multiple matrix rows can be computed within a single DRAM row, giving an additional edge to both asynchronous execution PIM and SplitSync PIM. Overall, across the seven benchmarks, SplitSync PIM achieves an average throughput improvement of 1.70x, 1.02x, and 1.06x compared to conventional PIM, ACT16-enabled PIM, and asynchronous execution PIM, respectively. A even more important point is that SplitSync achieves higher performance compared to other designs while maintaining the smallest area overhead relative to conventional PIM. The next section provides a detailed analysis of the area overhead.

C. Area Overhead

We estimated the area overhead of SplitSync PIM using a 28nm CMOS process. While the size of the global buffer does not increase compared to conventional PIM, adding a read port results in a 62% increase in its area. However, since the global buffer accounts for less than 1% of the total PIM area, this increase is negligible. Increasing the number of result latches to 4 sets per processing unit incurs a 1.5% area overhead per PU, which is significantly smaller than the area overhead observed in ACT16-enabled PIM and asynchronous execution PIM. As described in Section II-C and II-D, ACT16-enabled PIM incurs 20-25% and asynchronous execution PIM incurs 15-20% area overhead. Fig. 6b shows the throughput results normalized by PU area for the seven benchmarks.

Although proposed architectures use different approaches, all three designs effectively address the t_{FAW} constraint, resulting in similar performance. However, the area used to resolve the t_{FAW} constraint varies across architectures, revealing notable performance differences in iso-area performance. Factoring in PU area, SplitSync PIM shows performance improvements of 1.67x, 1.21x, and 1.20x compared to conventional PIM, ACT16-enabled PIM, and asynchronous execution PIM, respectively.

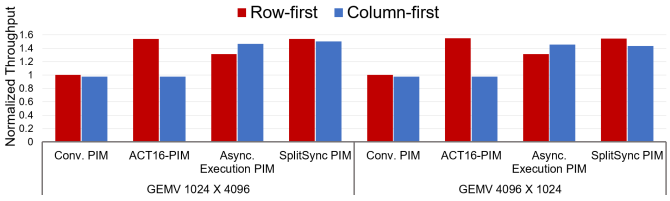


Fig. 7: Impact of tile scheduling on PIM performacne

D. Impact of Tile Scheduling

Fig. 7 shows the results of how tile scheduling affects PIM performance. We evaluate two cases of GEMV: 1K $\times$ 4K and 4K $\times$ 1K, with results normalized to the row-first tile scheduling of conventional PIM. For conventional PIM, row-first tile scheduling shows slightly better results than column-first scheduling, but the difference is minimal, with performance remaining nearly identical. In contrast, the other architectures show significant performance differences depending on the tile scheduling methods, most notably in ACT16-enabled PIM. This disparity occurs because, as row activation overhead is reduced, other overheads that were previously masked become more visible.

In asynchronous execution PIM, the performance difference is primarily due to frequent result reading in row-first tile scheduling. Unlike other architectures, asynchronous execution PIM cannot hide result reads behind precharge-activation due to I/O conflict issues and the high volume of results to be read out. As a result, it shows worse performance with row-first tile scheduling. With column-first tile scheduling, performance degrades due to the increased number of input writes, particularly affecting ACT16-enabled PIM. In conventional PIM, input writes are masked by the long activation time, but in ACT16-enabled PIM, shorter activation time prevents this masking, resulting in a greater performance impact. Conversely, SplitSync PIM and asynchronous execution PIM mitigate this issue by enabling more input reuse through additional result latches. Furthermore, in asynchronous execution PIM, only small blocks are written per tile, allowing frequent writes to be hidden during activation.

E. Impact of t_{FAW} and Row Size

Row activaion overhead is determined by the row activation time and the operation time. Row activation time is primarily influenced by t_{FAW}, while operation time depends largely on t_{CCD} and the row size. Since t_{CCD} is the inverse of the operation frequency, it can be excluded from this analysis, leaving changes in t_{FAW} and row size as the key factors determining row activation overhead. While there is a positive correlation between t_{FAW} and row size, the exact relationship varies by design. Therefore, we treat these as independent variables and analyze their impacts separately. Once the relationship between the two variables is established during the design phase, the analysis can focus on the relevant regions of the results. Fig. 8 presents these findings as a heatmap, with throughput normalized to the overall average. In conventional architectures, throughput generally decreases as row activation overhead increases

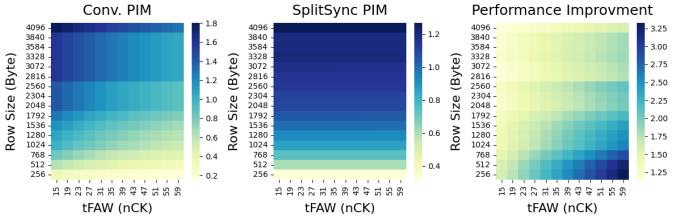


Fig. 8: Performance impact when varying t_{FAW} and row size

towards the lower right of the heatmap. In contast, SplitSync PIM less affected by t_{FAW}, with row size playing a more important role in its throughput. SplitSync PIM consistently shows a performance improvement of at lesat 1.15x across all areas. As expected, the performance gain becomes larger when row activation overhead becomes greater (moving towards the lower right of the heatmap).

F. Energy

We use the DRAMsim3 simulator to evaluate the energy consumption during PIM operations. For a 4K $\times$ 4K GEMV, ACT16-enabled PIM, asynchronous execution PIM and SplitSync PIM consume 97%, 89% and 86% of the energy compared to conventional PIM, respectively. ACT16-enabled PIM exhibits slight energy reduction over conventional PIM since energy savings come solely from reduced stand-by energy as a result of shorter processing times. In asynchronous execution PIM, input data is loaded only once per row, reducing I/O energy consumption. SplitSync PIM achieves further energy savings through shorter BG I/O paths and lower broadcasting frequency, as multiple rows are processed together, decreasing overall I/O energy usage. In both SplitSync PIM and asynchronous execution PIM, reduced I/O energy consumption and lower stand-by energy due to shorter processing times enhance energy efficiency.

V. Conclusion

In this paper, we present SplitSync PIM, a DRAM PIM architecture designed to reduces row activation overhead through bank group-level split synchronization. This approach allows each bank group to operate asynchronously to hide row activation overhead while maintaining synchronous operations within each bank group to enable input broadcasting at the bank group-level. SplitSync PIM addresses the t_{FAW} constraint without relying on the large reservoir capacitors and drivers used in ACT16-enabled PIM, and it eliminates the need for a shared accumulator in asynchronous execution PIM. Additionally, I/O conflicts occurring from asynchronous operations are resolved through a modified I/O architecture, enabling all bank groups to operate independently. Consequently, SplitSync PIM effectively reduces row activation overhead efficiently, with minimal area overhead.

VI. Acknowledgement

This work was supported in part by Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2023-00208606, NeuroHub+: Scheduler and Simulator for General In-Memory Neural Network Accelerators, RS-2022-II220266, Development of Ultra-Low Power Low-Bit Precision Mixed-mode SRAM PIM, IITP-2023-RS-2023-00256081: artificial intelligence semiconductor support program to nurture the best talents), BK21 FOUR program at Seoul National University, and ISRC at Seoul National University. The EDA tool was supported by the IC Design Education Center(IDEC). (Corresponding Author: Jae-Joon Kim).

REFERENCES

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, et al. "Language models are few-shot learners". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020.

[2] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo. "McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge". In: *IEEE Access* 8 (2020), pp. 135223–135243.

[3] *DDR5 SDRAM Standard*. JESD79-5. JEDEC Solid State Technology Association. 2020.

[4] F. Devaux. "The true processing in memory accelerator". In: *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society. 2019, pp. 1–24.

[5] J. Devlin. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[6] A. Dosovitskiy. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[7] *Graphics Double Data Rate (GDDR6) SGRAM Standard*. JESD250D. JEDEC Solid State Technology Association. 2018.

[8] H. Kal, C. Yoo, and W. W. Ro. "AESPA: Asynchronous Execution Scheme to Exploit Bank-Level Parallelism of Processing-in-Memory". In: *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '23. Toronto, ON, Canada: Association for Computing Machinery, 2023, pp. 815–827.

[9] H. Kim, H. Park, T. Kim, K. Cho, E. Lee, S. Ryu, et al. "GradPIM: A Practical Processing-in-DRAM Architecture for Gradient Descent". In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2021, pp. 249–262.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[11] D. Kwon, S. Lee, K. Kim, S. Oh, J. Park, G.-M. Hong, et al. "A 1ynm 1.25V 8Gb 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep Learning Application". In: *IEEE Journal of Solid-State Circuits* 58.1 (2023), pp. 291–302.

[12] Y. Kwon, K. Vladimir, N. Kim, W. Shin, J. Won, M. Lee, et al. "System architecture and software stack for GDDR6-AiM". In: *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE. 2022, pp. 1–25.

[13] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, et al. "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications". In: *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. 2021, pp. 350–352.

[14] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, et al. "A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. 2022, pp. 1–3.

[15] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, et al. "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pp. 43–56.

[16] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob. "DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator". In: *IEEE Computer Architecture Letters* 19.2 (2020), pp. 106–109.

[17] *Low Power Double Data Rate (LPDDR) 5/5X*. JESD209-5C. JEDEC Solid State Technology Association. 2023.

[18] Y. Paik, C. H. Kim, W. J. Lee, and S. W. Kim. "Achieving the Performance of All-Bank In-DRAM PIM With Standard Memory Interface: Memory-Computation Decoupling". In: *IEEE Access* 10 (2022), pp. 93256–93272.

[19] J. Park, S. Lee, and J. Lee. "NTT-PIM: Row-centric architecture and mapping for efficient number-theoretic transform on pim". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2023, pp. 1–6.

[20] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo. "McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2613–2622.

[21] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[22] A. Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

3D-TokSIM: Stacking 3D Memory with Token-Stationary Compute-in-Memory for Speculative LLM Inference

Wentao Zhao[1], Boya Lv[1], Meng Wu[1], Peiyu Chen[2,3], Fengyun Yan[2], Yufei Ma[1], Tianyu Jia[1,*], Ru Huang[1], and Le Ye [1,3,*]

[1]Peking University, Beijing, China
[2]Nano Core Chip Electronic Technology, Hangzhou, China
[3]Advanced Institute of Information Technology of Peking University, Hangzhou, China
[*]Corresponding Authors: tianyuj@pku.edu.cn, yele@pku.edu.cn

***Abstract*—The LLM decoding process poses a significant challenge for memory bandwidth due to its autoregressive nature. Prior 2D memory solutions fail to overcome this memory bottleneck due to limited memory-to-logic bandwidth. In this work, we propose 3D-TokSIM, a cross-stack solution by stacking 3D memory on logic die with a specially designed token-stationary compute-in-memory (CIM) to efficiently accelerate speculative decoding. Our CIM is developed with novel token-stationary dataflow to reduce data movement on logic die to save power and balance computation and memory access. To further reduce the buffer requirements, we perform architecture exploration and allocate notable CIM resources for achieving higher decoding parallelism. Compared to RTX 3090 GPU, 3D-TokSIM achieves 15.1× throughput and 324× energy efficiency improvements on speculative Llama2-7B decoding.**

***Index Terms*—LLM, Process-Near-Memory, Compute-In-Memory**

I. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language processing tasks and beyond. However, it is quite challenging for existing hardware solutions, e.g. GPUs, to achieve efficient LLM deployment. Due to the intrinsic autoregressive model nature, LLMs require loading the entire model to generate one new token at each decoding step [1]. Such operation behaviors lead to a severe bottleneck of memory bandwidth for LLM inference. Meanwhile, the existing GPUs or XPUs are designed with massively parallel computing units with quite limited off-chip memory bandwidth. Hence it is clear from many research papers that the LLM inference is a memory-bound task and desires cross-stack optimizations for bandwidth [2]–[5].

To alleviate the bandwidth bottleneck, 3D memory architecture using the hybrid bonding (HB) technique is a promising solution [6], [7]. By stacking the DRAM memory on top of the compute logic die, DRAM bandwidth can be significantly improved by parallel accessing DRAM banks, which were originally accessed at interleaved time. Based on the public data, the HB 3D memory can provide 135× bandwidth and 8× density improvements with more than 90% energy saving compared to existing 2D DRAM [8], [9].

Although 3D memory stacking brings valuable memory bandwidth, there is little exploration on how to efficiently deploy LLMs on such architecture. Even with 3D technology, the entire architecture remains memory-bound due to the inefficient autoregressive model behavior. A promising algorithmic strategy to alleviate the bandwidth bottleneck is speculative decoding (SD) technique [4], which utilizes an efficient drafter to produce multiple tokens and then simultaneously sends these tokens into LLM model for parallel verification.

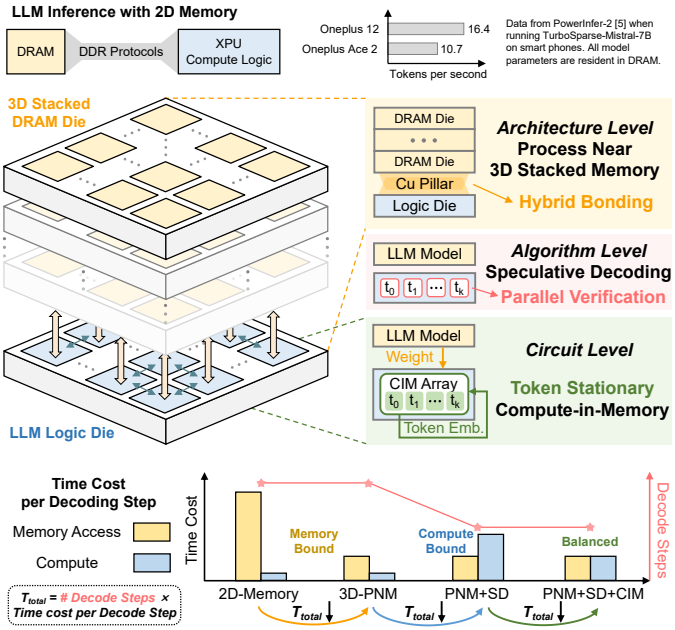


Fig. 1. Efficient LLM deployment on 3D stacking technique with cross-stack optimizations.

SD can successfully reduce the total decoding steps and alleviate the bandwidth requirement, with the cost of increasing computational workload in each decoding step. However, conventional digital design approach fails to efficiently process SD at the logic die, which effectively nullifies the high-bandwidth advantage of 3D memory stacking.

In this work, we explore a LLM deployment solution **3D-TokSIM**, which combines **3D** stacked DRAM processing-near-memory (PNM) and SRAM-based **tok**en **s**tationary compute-**i**n-**m**emory (CIM) for efficient speculative inference. As shown in Fig.1, at the architecture level, we leverage 3D stacked memory to significantly enhance the memory access bandwidth for each decoding step. At the algorithm level, we leverage the latest speculative decoding approach for the 3D stacked architecture to further reduce the number of decoding steps and bandwidth requirements. At the circuit level, we employ compute-in-memory (CIM) design with token stationary dataflow to efficiently support SD. Our cross-stack design and optimizations achieve fewer decoding steps with a balanced memory-compute workload per step. This results in reduced decoding time and a higher rate of token generation. Evaluated on our design using TSMC 22nm, it is observed that 3D-TokSIM achieves 15.1× throughput

and 324× energy efficiency improvements compared to RTX 3090 GPU. Moreover, 3D-TokSIM outperforms systolic array accelerated PNM of both throughput and energy efficiency by 1.5× and 6.4×, respectively. To the best of our knowledge, this is the first exploration work that enables fast LLM inference using 3D stacking on top of CIM architecture.

The contributions of this paper are summarized as follows:

- We propose 3D-TokSIM, a 3D stacked DRAM and SRAM-based CIM architecture targeting speculative LLM inference.
- We leverage token stationary dataflow with careful optimizations to balance the CIM design and DRAM bandwidth.
- We further perform several buffer optimizations to place abundant CIM for higher decoding parallelism.
- Our evaluation demonstrates that the proposed 3D-TokSIM has 15.1× throughput and 324× energy efficiency improvements compared to RTX 3090 GPU.

II. Background

A. Hybrid Bonding 3D Stacking

The ever-growing size of AI models has significantly exceeded the capacity of on-chip SRAM, resulting in frequent data transfers between DRAM and the main processor chip. Hybrid bonding process-near-memory (HB-PNM) [7] has emerged as a promising solution to the "memory wall" problem, offering fast and energy-efficient data access. In this approach, DRAM and logic die are integrated through direct copper-to-copper bonding. Each logic block can directly access its closely coupled DRAM banks and communicate with other logic blocks via the on-chip network (NoC). HB-PNM can achieve ultra-high interconnection density with a 3um pitch compared to about 50um microbump (ubump) pitch in HBM. Moreover, the resistance of HB is also much lower than ubump and through-silicon-vias (TSVs) [6].

HB technique has been initially studied for advanced memories. For example, SEDRAM [6] connected a DRAM array wafer and logic wafer using HB technology, with control, I/O, and peripherals located on the logic die. This motivates HB-PNM solutions for memory intensive applications. Niu et al. [7] developed an HB-PNM engine for recommendation systems that significantly outperformed CPU-DRAM system efficiency. Yue et al. [10] accelerated vision AI models with HB-PNM and gained notable efficiency improvement compared to prior accelerators. In addition to single layer DRAM stacking, multilayer array stacking is achieved with HB and mini-TSV in [9]. Driven by this, Chen et al. [11] presented an encryption engine for private inference with 4-layer stacked DRAM.

In the era of LLMs, it has already been well-studied that LLM decoding is heavily constrained by DRAM bandwidth [2], [3]. The memory bandwidth bottleneck is more severe for mobile devices, such as mobile phones or laptops. For example, mobile devices with the LPDDR5x protocol generally offer a peak bandwidth of 68GB/s, e.g. SK hynix 24GB LPDDR5x mobile DRAM [12]. When running a 4-bit quantized Llama-7B model, the theoretical throughput upper bound is 19.4 tokens per second (TPS), estimated by dividing the model size by the peak bandwidth. After adopting HB-PNM for edge devices, TBps-level bandwidth can be obtained resulting in approximately 500 TPS and 23-29× bandwidth improvement compared to standard LPDDR5x.

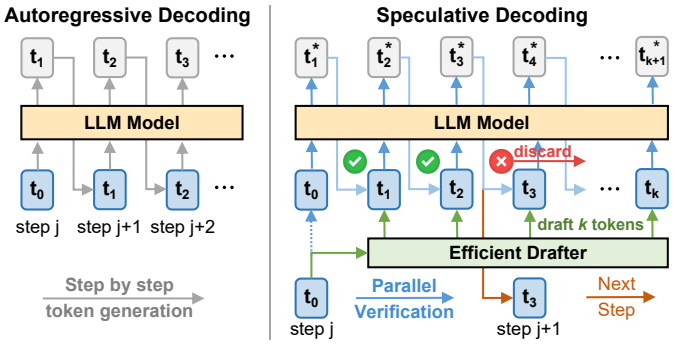


Fig. 2. Autoregressive decoding vs. Speculative decoding.

B. Speculative Decoding

Despite the use of high-bandwidth 3D technology, the LLM decoding may still be bounded by memory access due to its inherently autoregressive nature. Based on previous profiling [3], when the OPT-6.7B model is executed on an A100 GPU, the GEMV operation during the decoding phase accounts for over 80% of the total latency, while the GPU utilization remains below 25%. Speculative decoding [4] has emerged as a promising paradigm to boost LLM decoding by enabling the verification of multiple tokens in a single decoding step.

As shown in Fig. 2, SD first employs an efficient drafter to generate multiple tokens and then sends these tokens into the model in parallel to verify if they align with the LLM inference results. For a sequence of tokens to be verified, the first token that does not match the LLM result should be replaced with the LLM output, and any subsequent tokens should be discarded [4]. An important hyperparameter for SD is the number of draft tokens, denoted as k. While a higher k can lead to fewer decoding steps, the computational overhead grows up with k, potentially causing two other problems: 1) memory-bounded decoding steps will be shifted to computation-bounded, and 2) computations for those unaccepted tokens are inefficient and waste energy. Hence it is crucial to co-optimize SD across algorithm and hardware for optimal efficiency.

Since the first presentation of SD, there have been quite a few algorithm variants of SD to reduce the bandwidth requirement. For example, Leviathan et al. [13] utilized the small version of the T5 series to accelerate the larger ones. Yang et al. [14] introduced an early exiting scheme to initiate future tokens drafting, and He et al. [15] retrieved relevant tokens from a knowledge reservoir to draft tokens. Among these variants of SD, Inference with Reference (IwR) [16] is highly efficient and easy to deploy since it does not require an additional model for drafting. By exploring the massive overlaps between in-context references and outputs, IwR directly copies tokens from reference for parallel verification. This approach is highly practical in scenarios such as retrieval-augmented generation, cache-assisted generation, and multi-turn conversations.

III. 3D-TokSIM Architecture

A. Overall Architecture

In this paper, we propose **3D-TokSIM**, a **3D**-stacked architecture with **tok**en **s**tationary C**IM** for efficient LLM inference. As shown in Fig. 3 left, 3D-TokSIM is composed of multiple parallel LLM

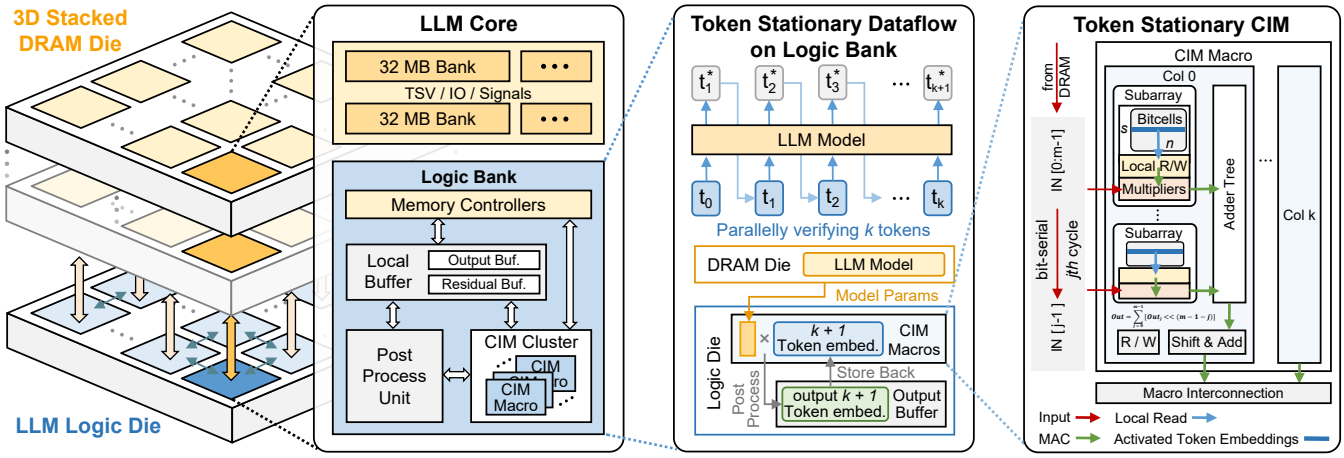


Fig. 3. Architecture overview: Stacking 3D memory above the logic die with token-stationary compute-in-memory clusters.

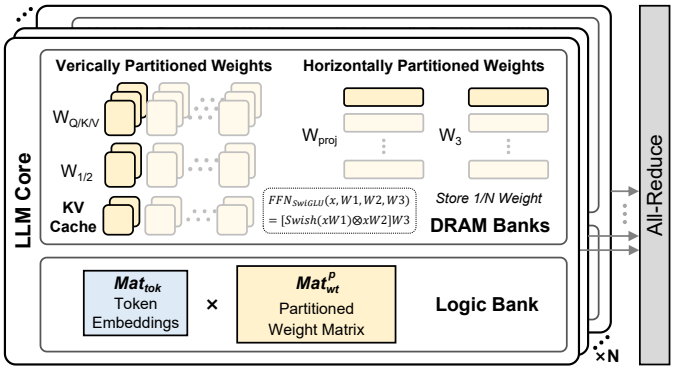


Fig. 4. Distributed Mapping for Parallel Acceleration.

Cores, and each core is composed of several DRAM Banks on DRAM dies and one logic bank on a logic die. DRAM dies and logic die are closely integrated through HB technique, which allows fast and energy-efficient DRAM access for logic banks. Each logic bank consists of memory controllers corresponding to above DRAM banks, a post-processing unit for handling non-linear operations, a CIM cluster responsible for multiply-and-accumulation (MAC) operation, and a local buffer. The LLM cores are connected through a NoC to enable inter-core data transfer.

Fig. 4 shows a typical mapping process to deploy multi-head-attention (MHA) and feed-forward-network (FFN) onto N parallel LLM Cores. For MHA, each LLM Core requires storing vertically partitioned weights for Q/K/V generation, KV cache, and horizontally partitioned projection weights. For FFN, the weights are partitioned vertically before activation and horizontally after activation. During LLM decoding, each logic bank loads weight or KV cache from its corresponding DRAM banks and performs multiplication with token embeddings. Following the last multiplication in MHA/FFN, an all-reduce operation merges results across cores, and then the full results are added to the initial token embeddings in the residual buffer to complete residual connection.

For each LLM core, we designate the matrix multiplications (MMs) in charge as $Mat_{tok} \times Mat^{p}_{wt}$, where p stands for partition. Either Mat^{p}_{wt} or Mat_{tok} can be mapped to the CIM cluster, which corresponds to **weight stationary** or **token stationary** dataflow. Noticed that even though Mat^{p}_{wt} constitutes only 1/N of the original weight matrix, its size remains excessively large. Consequently, Mat^{p}_{wt} must be divided into smaller tiles when being mapped to CIM cluster, rendering two disadvantages: 1) CIM cluster needs to frequently update its weight, and Mat_{tok} must be read from local buffer repeatedly, resulting in massive data read/write operations, and 2) The input buffer is necessary to hold the entire input token embeddings, which further tightening the area constraints.

In contrast, we map Mat_{tok} instead of Mat^{p}_{wt} to the CIM cluster, referred to as **token stationary** dataflow, to liberate our design from the aforementioned concerns. Since the IwF typically dispatches a small group of tokens for parallel verification, it becomes possible to store the entire Mat_{tok} within the CIM by allocating sufficient storage resources. This eliminates the need for further tiling and input buffering.

B. Logic Die: Token Stationary CIM

In our work, a **token stationary** dataflow is developed to efficiently support speculative decoding. As illustrated in the middle part of Fig.3, the CIM macros hold all token embeddings Mat_{tok} in bitcell arrays. The DRAM sends model parameters as the input of CIM to perform MMs. The output result first undergoes post-processing and is held in the output buffer, which is then sent to CIM macros for subsequent MM.

Compared to conventional **weight stationary** dataflow, our **token stationary** dataflow reduces frequent read/write operations on the logic die and saves considerable energy. Note that due to low-resistance HB, the power consumption of data transfer from DRAM has been significantly reduced, for instance, to 0.88pJ/bit in [7] and 0.66pJ/bit in [9]. Consequently, the energy consumption of SRAM read/write operation is now comparable to that of DRAM data transfer. Therefore, we employ token stationary dataflow to eliminate frequent SRAM read/write operations. As shown in the middle of Fig.3, for every MM, we maintain all token embeddings Mat_{tok} in CIM and load weight from DRAM as CIM input. The entire MM process does not require storing weights in the CIM, nor does it require reading token embeddings from the local buffer. The outcomes are post-processed and stored back to CIM before subsequent MM.

TABLE I
NOTATIONS FOR DESIGN PARAMETERS AND LLM MODEL

Architecture Design Parameters	
N	Number of Parallel LLM Cores
$DRAM_{width}$	Number of DRAM I/O within a LLM Core
$DRAM_{freq}$	Frequency of DRAM I/O
CIM_{num}	Number of CIM Macros within a CIM Cluster
$Macro_{col}$	Number of CIM Columns within a CIM Macro
$Macro_{row}$	Number of Subarrays within a CIM Column
CIM_{freq}	Frequency of CIM
(s, n)	Rows and Columns of Subarray Bitcells
LLM Model Parameters	
dim	Model Dimension
d_{ff}	Intra-FFN Dimension
d_{head}	Attention Head Dimension
k	Number of Draft Tokens in Each Decoding Step

Our CIM macros are designed to meet the requirement of token stationary dataflow. As depicted in Fig.3 right, CIM macros are composed of several identical CIM columns. A CIM column has multiple CIM subarrays and an adder tree. Within each subarray, there are $s \times n$ SRAM bitcells, n local read/write (R/W) units, and n 1-b multipliers, where n equals the minimum supported data width. Token embeddings Mat_{tok} are stored within bitcells, which are then activated and read by local read/write units. The multipliers are shared by s row bitcells, where s indicates the optimal storage-compute ratio for token stationary dataflow. An m-bit input vector requires m cycles for bit-serial computation, and a shifter-and-adder is responsible for accumulating m partial sums. Multiple CIM macros can be connected vertically or horizontally to support various shapes of matrix multiplication. The notations utilized in this work for architecture design and LLM model are summarized in Table I.

For a single MM, each token embedding of length d performs a dot product with weights or KV cache, which is done in parallel for $k+1$ token embeddings by broadcasting. Therefore, we store individual token embeddings in separate CIM columns, ensuring that each token embedding is mapped to its own distinct column:

$$Macro_{col} = k+1, \quad n = TokenEmb_{width} \tag{1}$$

where $TokenEmb_{width}$ denotes the bit width of token embeddings.

Given that $k+1$ is usually much smaller than d, a single CIM macro is designed to accommodate $k+1$. Multiple macros are connected vertically to cover the entire length of the maximum d as follows:

$$CIM_{num} \times Macro_{row} \times s = max(dim, \frac{d_{ff}}{N}) \tag{2}$$

IV. ARCHITECTURE OPTIMIZATIONS

A. Exploration Constraints

Based on the above analysis, we have identified token stationary as the suitable dataflow for speculative LLM inference. Furthermore, we perform architecture modeling to guide the design of token-stationary CIM cluster to take full advantage of this dataflow.

To fully exploit the 3D DRAM bandwidth, the CIM cluster should be equipped with sufficient computational resources. More specifically, the rate at which data are read from DRAM should be aligned with the rate at which CIM consumes them. The bit width

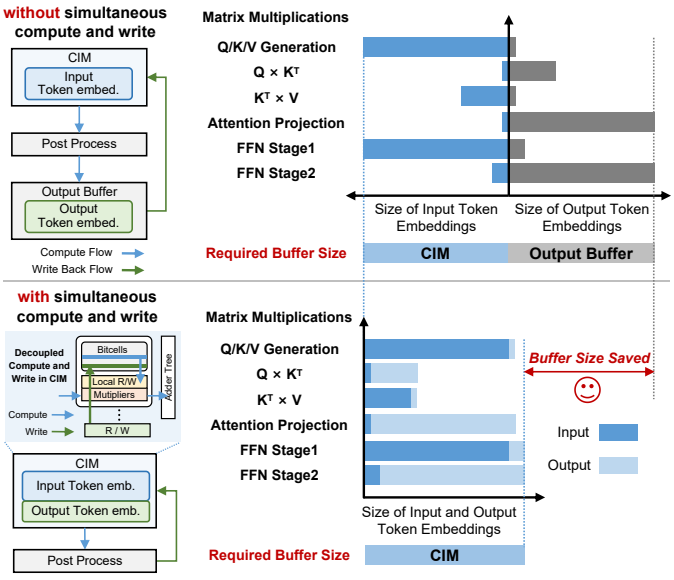


Fig. 5. Eliminating Buffer by Writing Output to CIM.

of data is represented by $Data_{bits}$. Within a unit of time T_{unit}, the number of data read from DRAM is represented by:

$$Data_{read} = T_{unit} \times \frac{DRAM_{width}}{Data_{bits}} \times DRAM_{freq} \tag{3}$$

Since data are sent into CIM bit-serially, the number of data consumed by CIM within T_{unit} is expressed as:

$$Data_{consume} = T_{unit} \times \frac{CIM_{num} \times Macro_{row} \times CIM_{freq}}{Data_{bits}} \tag{4}$$

Equalizing $Data_{read}$ and $Data_{consume}$ results in:

$$CIM_{num} \times Macro_{row} \times CIM_{freq} = DRAM_{width} \times DRAM_{freq} \tag{5}$$

However, the reduction length of intermediate token embeddings during MHA or FFN is small, and they should be copied to fully utilize CIM performance and DRAM bandwidth. For each copy, it should be able to perform MAC calculations independently. Therefore, the minimal reduction length achievable by CIM is set to correspond with the minimum d:

$$Macro_{row} = min(d_{head}, \frac{d_{ff}}{N}) \tag{6}$$

Within the area constraints, the integer solution closest to the above requirements is selected for CIM design. For larger model dimensions, CIM capacity per column expands by adding more bitcells to subarrays. As CIM column equips more capacity or performance, its area increases, ultimately reducing the total number of CIM columns which corresponds to supported k.

B. Architecture Optimizations

Since the area of a single logic bank is constrained by the area of its corresponding DRAMs above it, the maximum supported k is also limited by the available CIM area. To ensure ample CIM resources, we further optimize the on-chip SRAM size and reduce the buffer requirement for each LLM core. Optimization opportunities are explored at both intra- and inter-core levels to minimize buffer requirements for MM output and residual connection, respectively.

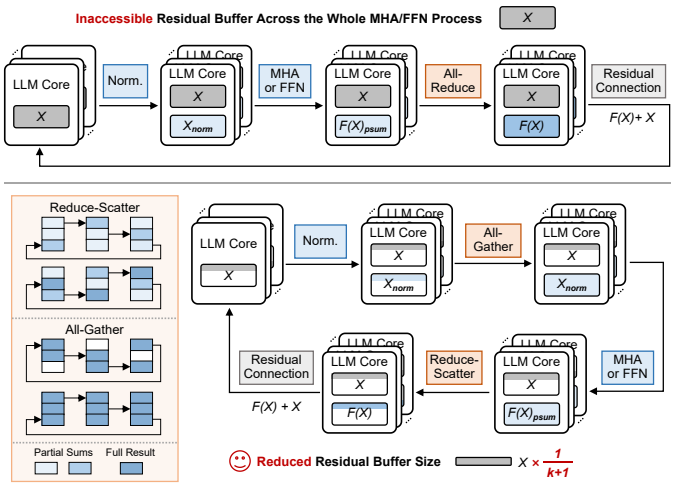


Fig. 6. Reducing Residual Buffer by Jointly Considering Residual Connection with All-Reduce Operation.

1) Output Buffer Optimizations: We first focus on optimizing the scheduling of MM within each core, aiming to eliminate output buffer. As illustrated in the Fig.5 top, our token stationary dafalow ensures input token embeddings are fully resident in CIM, while the output token embeddings are stored in the output buffer. The required capacity of CIM and output buffer are determined by the largest token embeddings size across all MMs as depicted in the upper right of Fig.5. Since the output of MM will be returned to CIM for subsequent MM, by enabling simultaneous compute and write of CIM, the output can be directly written back to CIM without temporarily residing in output buffer.

Therefore, we implement simultaneous compute and write CIM as shown in the lower left of Fig.5, where the local read/write units have decoupled multipliers and adder trees from bitcells, showing no resource contention between compute and write operation. Consequently, the elements of output token embeddings can be written back to CIM after being post-processed even if the CIM is still computing. As shown in the lower right of Fig.5, the output buffer is now eliminated at the cost of slightly increased CIM size, and we adjust our previous CIM design size in (2) to:

$$CIM_{num} \times Macro_{row} \times s = dim + max(d_{head}, \frac{d_{ff}}{N}) \quad (7)$$

For the typical Llama2-7B model ($dim = 4096$, $d_{head} = 128$ and $d_{ff} = 11008$) accelerated with 32 cores, the capacity of CIM increases about 8.4% according to (7) and (2), which demonstrates minimal area overhead.

2) Residual Buffer Optimizations: Residual connection (RC, $F(X) + X$) impedes resource-constrained model inference as it requires extra buffer. Previous work has attempted to reorganize residual connections to enable tiny devices to accommodate CNN models [17]. As shown at the top of Fig.6, for parallel LLM inference, RC requires keeping the unnormalized input token embeddings of MHA or FFN, denoted as X, across the whole computing process. The buffer tailored for X is inaccessible during MHA or FFN, leading to constraint area for CIM.

As displayed at the bottom of Fig.6, we reduce the residual buffer size for each LLM core by jointly considering RC with all-reduce (AR) operation between cores. We adopt bandwidth optimal ring AR [18], [19], which is further divided into two phases: reduce-scatter (RS) and all-gather (AG). Noticing that each core holds only a portion of the full result after RS, inserting RC between RS and AG allows each core to store only the corresponding portion of X. In addition, AG begins after normalization and each core receives the normalized token embeddings from other cores. Because token embeddings are stored column-wise in CIM, vertical read/write mode similar to prior CIMs [20], [21] is leveraged to support above proposal with the aid of local read/write units. With k draft tokens in SD, our scheme reduces the residual buffer size in each core to $1/(k+1)$. For typical k larger than 5, the area cost of the residual buffer is reduced to less than 16.7%.

TABLE II
3D-TOKSIM SPECIFICATION

System Configuration	
Number of Cores	32
Stacked DRAMs	4
DRAM within Single LLM Core	
Number of Banks	4
Bank Size	32 MB
Total Bandwidth	62.5 GB/s
CIM within Single LLM Core	
Number of Macros	8
Macro Size	128 × 13
Subarray Size	5 × 16
Performance	3.25 TOPS
Energy Efficiency	9.51 TOPS/W
Area	1.39 mm^2

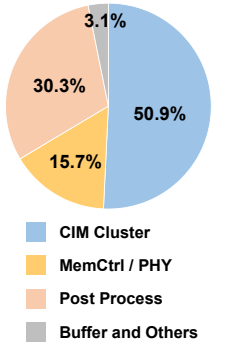


Fig. 7. Area Breakdown of Logic Bank

V. EVALUATION

A. Experiment Setup

Hardware Setup. In our work, we selected a 32MB DRAM bank with 256 I/O as the fundamental unit for our memory design, with area referring to [10]. Both DRAM die and Logic die are operating at 500MHz. The energy consumption of multilayer DRAM refers to the latest publication [9]. Our logic die is designed using TSMC 22nm technology. The area of memory controller and PHY are based on [11], which has similar technology node to ours. We designed an in-house 22nm CIM macro with post-layout simulation. The energy efficiency of our CIM macro is 9.51TOPS/W for W4A16/W8A8 data format with 50% input toggle rate and 50% weight bit-level sparsity. The local buffer is generated by Arm Memory Compiler. The area and power of processing unit are based on our RISC-V VPU with various special functions integration referred from [22]. A performance model incorporating above hardware information has been constructed for architecture exploration.

To fairly compare weight stationary (WS) and proposed token stationary (TS) dataflow, the CIM under WS is also carefully designed, where the storage and performance of CIM are set to ensure that the weight update and the computation overlap as much as possible. In addition, a systolic array based on Gemmini [23] is synthesized by Cadence Genus. It is used as a comparison baseline for more comprehensive architecture evaluation.

Algorithm Setup. We evaluated the speculative decoding using the model from [16]. For different draft lengths k ranging from 0 to 18, we recorded the decoding information across 100 samples. The decoding information includes the length of KV cache and whether

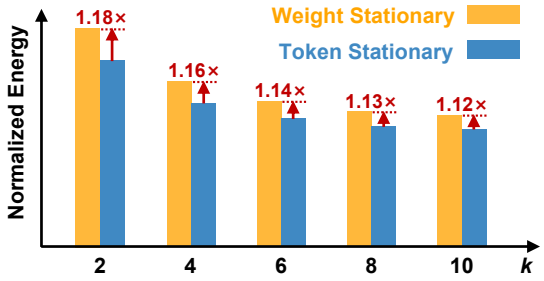


Fig. 8. Energy Cost of Weight Stationary and Token Stationary Dataflow.

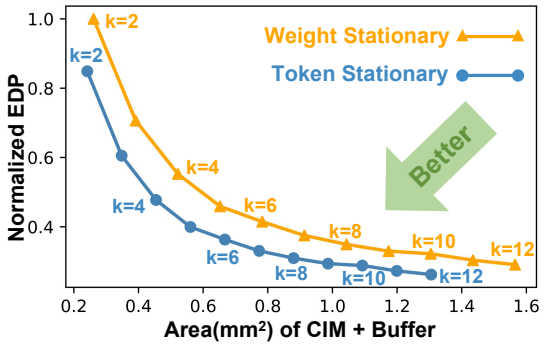


Fig. 9. EDP vs. Area of Weight Stationary and Token Stationary Dataflow.

coping k tokens or not in each decoding step. We use this record as the algorithm input for our performance model.

B. Implementation and Exploration

We implemented 3D-TokSIM with 32 LLM cores, with each core containing 4 DRAM banks stacked vertically to support the typical LLama2-7B [1] model with W4A16 quantization. The 4-layer stacked DRAM offers 2TB/s bandwidth and 4GB capacity in total. The specification of 3D-TokSIM is listed in Table II. Each LLM core contains 8 CIM macros, and the column of each CIM macro is 13 to support SD with k equals 12. These 8 cores together provide a performance of 3.25TOPS and 130KB capacity. The area breakdown of a single logic bank is depicted in Fig.7. It illustrates that our optimization ensures ample CIM area, while the buffer area is minimal.

We decomposed our architecture optimization into two parts: the energy saving through token stationary dataflow and the enhancements in energy-delay product (EDP) by further buffer reduction. Our optimization yields similar results across different models, and here we present the results for the 4-bit quantized LLAMA2-7B model. The energy costs of WS and TS dataflow with different k are shown in Fig.8. For k ranges from 2 to 10, the TS dataflow offers 1.18×-1.12× better energy efficiency than WS. As k increases, the improvement decreases slightly. This is due to TS primarily optimizing data transfer overhead, while a larger k increases energy costs for computation. Fig.9 illustrates the EDP of WS and TS dataflows, with supported k ranging from 2 to 12. It can be clearly seen that TS always achieves better EDP than WS under the same area constraint. For k larger than 12, they are not depicted. This is because the EDP almost no longer decreases due to the higher compute overhead and saturated decoding steps.

Our architecture is extended to support various LLM models. Recently, low-bit quantization of LLMs has been developed, even reaching down to 2-bit quantization [24]. Therefore different architecture configurations are set to adapt larger model sizes with 3/2 bit quantization. As listed in Table III, compared to WS dataflow, the optimized TS dataflow consistently supports larger k and brings 1.13×-1.23× EDP improvement across various LLM models.

TABLE III
OPTIMIZATION OF VARIOUS LLM MODELS

LLM Model	Arch. Cfg.*	k Opt.	EDP Improve.
LLAMA2-7B-4bit	32×1×4	10→12	1.23×
LLAMA2-13B-2bit	40×1×4	9→12	1.21×
OPT-30B-3bit	56×2×4	11→12	1.13×
OPT-66B-2bit	72×2×4	10→12	1.16×

* Architecture Configuration: Number of LLM Cores × DRAM Banks per Core per Layer × Stacked DRAM Layers

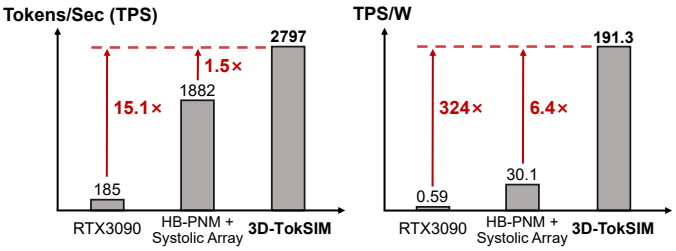


Fig. 10. Comparison result with RTX 3090 GPU and systolic array accelerated HB-PNM.

C. Comparison and Discussion

Fig.10 shows the comparison result of 3D-TokSIM with RTX 3090 GPU and systolic array accelerated HB-PNM on speculative Llama2-7B model. Our 3D-TokSIM generates 2797 tokens per second (TPS), which is 15.1× faster than RTX 3090. Notably, the RTX 3090 provides a formidable 936GB/s DRAM bandwidth, and our 3D stacked memory only delivers twice the speed at 2TB/s. Therefore, this improvement is not solely due to higher bandwidth, but rather attributed more to the architecture. Compared to HB-PNM with systolic array acceleration, 3D-TokSIM achieves 1.5× higher TPS. This is mainly due to the lower performance density of the systolic array compared to CIM, resulting in the system being compute-bound. In addition, the energy efficiency of 3D-TokSIM outperforms RTX 3090 and HB-PNM with systolic array acceleration by 324× and 6.4×, which demonstrates the efficiency of proposed token stationary CIM.

VI. CONCLUSION

This paper presents a cross-stack solution for efficient LLM inference, i.e. 3D-TokSIM, which incorporates 3D stacked DRAM and token stationary CIM to efficiently accelerate speculative decoding. Optimization of the token stationary dataflow further reduces buffer needs, enhancing decoding parallelism and improving EDP with ample CIM resources. The implemented 3D-TokSIM using TSMC 22nm demonstrates 15.1× and 1.5× faster token generation speed with 324× and 6.4× energy efficiency compared to RTX 3090 and systolic array accelerated PNM.

ACKNOWLEDGMENT

This work was supported in part by the NSFC under Grant 62225401; and in part by the National Key R&D Program of China under Grant 2024YFB4505000.

REFERENCES

[1] H. T. et al., "Llama 2: Open foundation and fine-tuned chat models," 2023.

[2] J. H. K. et al., "Samsung pim/pnm for transfmer based ai : Energy efficiency on pim/pnm cluster," in *2023 IEEE Hot Chips 35 Symposium (HCS)*, pp. 1–31, 2023.

[3] S.-S. Park *et al.*, "An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large language models," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[4] H. Xia *et al.*, "Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding," *arXiv preprint arXiv:2401.07851*, 2024.

[5] Z. Xue *et al.*, "Powerinfer-2: Fast large language model inference on a smartphone," 2024.

[6] B. Fujun *et al.*, "A stacked embedded dram array for lpddr4/4x using hybrid bonding 3d integration with 34gb/s/1gb 0.88pj/b logic-to-memory interface," in *2020 IEEE International Electron Devices Meeting (IEDM)*, 2020.

[7] D. Niu *et al.*, "184qps/w 64mb/mm23d logic-to-dram hybrid bonding with process-near-memory engine for recommendation system," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, pp. 1–3, 2022.

[8] K.-D. Hwang *et al.*, "A 16gb/s/pin 8gb gddr6 dram with bandwidth extension techniques for high-speed applications," in *2018 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 210–212, 2018.

[9] S. Wang *et al.*, "A 135 gbps/gbit 0.66 pj/bit stacked embedded dram with multilayer arrays by fine pitch hybrid bonding and mini-tsv," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI)*, 2023.

[10] Z. Yue *et al.*, "Exploiting similarity opportunities of emerging vision ai models on hybrid bonding architecture," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 396–409, 2024.

[11] Z. Chen *et al.*, "A high-throughput private inference engine based on 3d stacked memory," in *61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

[12] Michail, "SK hynix is now shipping its 24GB LPDDR5X DRAM, will debut on OnePlus Ace 2 Pro." https://www.gsmarena.com/sk_hynix_is_now_shipping_its_24gb_lpddr5x_dram_will_debut_on_oneplus_ace_2_pro-news-59516.php, 2023. [Online; accessed 2024-11-18].

[13] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, JMLR.org, 2023.

[14] S. Yang, G. Lee, J. Cho, D. Papailiopoulos, and K. Lee, "Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding," *ArXiv*, vol. abs/2307.05908, 2023.

[15] Z. He, Z. Zhong, T. Cai, J. D. Lee, and D. He, "Rest: Retrieval-based speculative decoding," *ArXiv*, vol. abs/2311.08252, 2023.

[16] N. Yang *et al.*, "Inference with reference: Lossless acceleration of large language models," Tech. Rep. MSR-TR-2023-17, Microsoft, April 2023.

[17] H. R. Mendis, C.-K. Kang, C.-H. Lin, M.-S. Chen, and P.-C. Hsiu, "Deep reorganization: Retaining residuals in tinyml," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, (New York, NY, USA), Association for Computing Machinery, 2024.

[18] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distributed Comput.*, vol. 69, pp. 117–124, 2009.

[19] Baidu, ""baidu-research/tensorflow-allreduce"." https://github.com/baidu-research/tensorflow-allreduce. [Online; accessed 2024-10-09].

[20] F. T. et al., "A 28nm 15.59µj/token full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, pp. 466–468, 2022.

[21] J. Wang *et al.*, "A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, 2020.

[22] E. Reggiani, R. Andri, and L. Cavigelli, "Flex-sfu: Accelerating dnn activation functions by non-uniform piecewise approximation," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023.

[23] H. G. et al., "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 769–774, 2021.

[24] J. Chee, Y. Cai, V. Kuleshov, and C. D. Sa, "Quip: 2-bit quantization of large language models with guarantees," *Advances in neural information processing systems*, vol. 36, pp. 4396–4429, 2023.

Routability-aware Packing for High-density Nonvolatile FPGAs

Huichuan Zheng
School of Computer Science and Technology, Shandong University

Yuqing Xiong
School of Computer Science and Technology, Shandong University

Jian Zuo
School of Computer Science and Technology, Shandong University

Hao Zhang
School of Computer Science and Technology, Shandong University

Zhenge Jia
School of Computer Science and Technology, Shandong University

Mengying Zhao*
School of Computer Science and Technology, Shandong University

***Abstract*—Nonvolatile field-programmable gate arrays (NVFPGAs) can use multi-level cell (MLC) nonvolatile memories (NVMs) to enhance their logic density. However, the high-density design of NVFPGAs degrades the intra-routability of configurable logic blocks (CLBs), which significantly prolongs the time consumed by the packing process in the computer-aided design (CAD) flow. To relieve the efficiency degradation, in this paper, we propose a routability-aware re-pair stage to adjust the logical-physical look-up table (LUT) assignments to mitigate the congestion and improve their intra-routability, thereby reducing the packing time. In addition, exploiting the structural equivalence of MLC LUTs, we remove unnecessary intra-routing attempts from packing to further improve efficiency. Evaluation shows the proposed strategies reduce packing time by 41.48% on average.**

***Index Terms*—nonvolatile memory (NVM), multi-level cell (MLC), field-programmable gate array (FPGA), computer-aided design (CAD), packing.**

I. Introduction

Field-programmable gate array (FPGA) is a promising option for many emerging applications, including near memory computing [1], [2], computer vision [3], [4], data encryption [5], [6], and large language model [7], [8], due to its high parallelism and reconfigurable flexibility. To deploy these applications, corresponding functions are transformed into small truth tables, which are stored in look-up tables (LUTs) inside configurable logic blocks (CLBs). With support of other programmable resources, i.e., routing resources and on-chip memories, the functionality can be implemented on FPGAs. As the complexity of applications grows, the demand for FPGA's capacity also continues to expand [9]–[12].

Traditional FPGAs adopt static random-access memory (SRAM) as their on-chip memory, which suffers from issues such as limited logic capacity and high leakage power. In contrast, emerging nonvolatile memories (NVMs) have advantages of high storage density, near-zero leakage power, and non-volatility. Thus, they are proposed to replace SRAM to build nonvolatile FPGAs (NVFPGAs), e.g., phase change memory based NVFPGA [13]–[15], magnetoresistive RAM based NVFPGA [16], and Resistive RAM based NVFPGAs [17]–[19], where resources including logic [20], routing [21]–[23] as well as on-chip memories [24] can be designed with NVMs. These representative NVMs can also operate as multi-level cell (MLC), which further increases the capacity of NVFPGAs.

Although MLC NVM enables high-density design for NVFPGAs, it also introduces challenges for the FPGA computer-aided design (CAD). The CAD flow transforms a target circuit in Hardware Description Language (HDL) to the bitstreams that can run in FPGA platforms. As shown in Fig. 2 (a), a typical FPGA CAD flow contains the main processes of front-end synthesis, technology mapping, packing, placement, and routing [25]. Taking the packing process as an example, it clusters LUTs into CLBs, so that the subsequent placement and routing can be conducted at the granularity of CLBs instead of LUTs. A CLB is successfully packed if logical LUTs can be well paired with physical positions inside it, and these logics are routable with limited intra-CLB routing resources. Since MLC FPGA architecture enhances the logic density, the number of logical connections within a CLB also significantly increases, which degrades the intra-routability. Thus, the packing algorithm has to invoke the time-consuming intra-routing process more frequently to ensure the routability of the packing solution, which seriously affects the efficiency of the packing process. According to [26], the time consumed by the packing process for MLC NVFPGA is about 8.9× higher than that for traditional SRAM-based FPGA.

Thus, in this paper, to relieve the degradation of intra-routability, we leverage routability considerations to guide the packing process, thereby reducing intra-routing invocations and improving packing efficiency. Specifically, we propose a routability-aware re-pair stage and integrate it into the existing packing process. To mitigate the intra-CLB congestion, it pairs logics from heavily congested regions to those regions with fewer logics. Through the proposed re-pair stage, some unroutable packing solutions could become routable so that a large number of intra-routing invocations could be avoided,

This work was supported by the National Key Research and Development Program of China (Grant No. 2022YFB4502004), the Natural Science Foundation of China (Grant No. 62372270), and Young Talent of Lifting engineering for Science and Technology in Shandong, China (Grant No. SDAST2024QTA078).

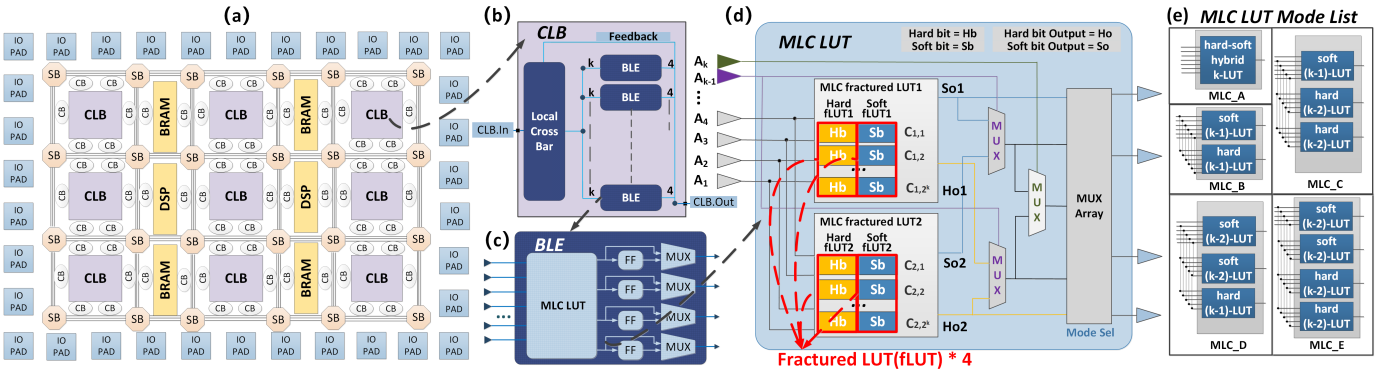


Fig. 1. The NVFPGA architecture proposed in [20]: (a) FPGA model, (b) CLB model, (c) BLE model, (d) MLC LUT model, and (e) MLC LUT mode list.

which can significantly improve the efficiency of the packing process for MLC NVFPGAs. In addition, exploiting the structural characteristics of MLC logic resources, we propose a duplicate intra-routing removal technique to avoid the routing attempts for input-equivalent packing solutions. The contributions of this paper are listed as follows.

- A routability-aware re-pair stage is proposed to improve the intra-routability of initially unroutable CLBs.
- Based on structural characteristics of MLC physical LUTs, an input-equivalent LUT removal technique is proposed to improve the efficiency of the packing process.
- Comprehensive evaluations are conducted on the proposed techniques to validate their effectiveness.

The remainder of this paper is organized as follows. Section II introduces the background and summarizes related work. Section III introduces the re-pair stage based method to optimize the packing efficiency. Section IV gives the evaluation results and discussions, followed by overhead analysis. Section V concludes this paper.

II. Background and Related Work

A. MLC LUT in NVFPGA

A typical FPGA architecture includes configurable logic blocks (CLBs), switch boxes (SBs), connection boxes (CBs), and block random access memories (BRAMs), as shown in Fig. 1 (a). Among these components, CLBs are the main logic resources to implement combinational and sequential logics. As Fig. 1 (b) demonstrates, each CLB contains a cluster of basic logic elements (BLEs) along with a fully populated crossbar. As shown in Fig. 1 (c), a BLE can be further decomposed into one look-up table (LUT), some flip-flops (FFs), and several multiplexers (MUXs). Fig. 1 (d) shows the MLC k-LUT structure proposed in [20], whose logic capacity is doubled compared with the original LUT structure. Here k denotes the maximum number of inputs the LUT can support. An MLC LUT contains four fractured $(k-2)$-LUTs, which are the smallest unit of logics in this structure. Fig. 1 (e) illustrates the mode list of the proposed MLC LUT structure. These fractured LUTs can be configured independently or work in group as a bigger LUT with more inputs, i.e., as $(k-1)$-LUT or k-LUT. This structure significantly improves the logic density and flexibility of CLBs. As reported in [20], compared with SRAM-based FPGAs, this MLC NVFPGA architecture achieves improvements in area, performance, and leakage power by 32.66%, 7.45%, and 96.13%, respectively.

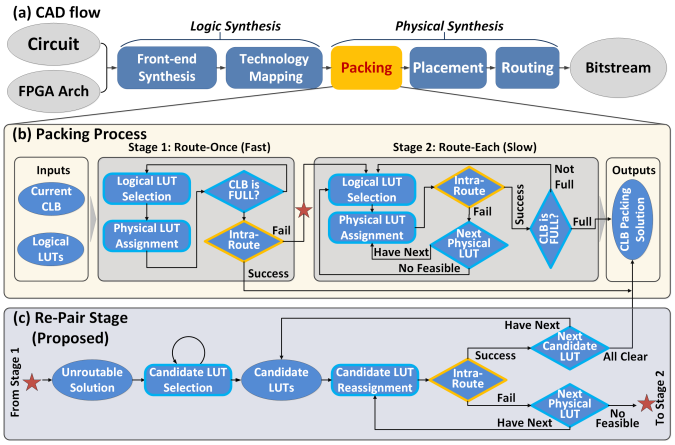


Fig. 2. The FPGA CAD flow, including: (a) the overview of a typical FPGA CAD flow, (b) the current packing process in the Verilog-To-Routing (VTR) flow [25], (c) the proposed re-pair stage.

B. Packing Process for FPGAs

The packing process clusters individual LUTs into CLBs, whose results will influence the subsequent processes of physical synthesis. A well-packed solution can significantly reduce the number of CLBs and the inter-CLB connection, greatly improving the efficiency of the placement and routing procedures. Currently, the representative packing algorithm adopted in Verilog-To-Routing (VTR) [25] contains two stages, denoted as Route-Once stage and Route-Each stage in this paper. As Fig. 2 (b) shows, in the Route-Once stage, LUTs are packed into the CLB until no more can be added. After these LUTs are packed, an intra-routing process is invoked to verify whether the packing solution is routable. If the current CLB is routable, the packing process for this CLB is finished. If it is unroutable, the packing process enters the Route-Each stage, where the routing process will be invoked after each time an LUT is added into the CLB. Since routing is usually the most time-consuming process in the CAD flow, the Route-Each stage takes significantly longer time than the Route-Once stage.

There have been many studies conducted on the packing process. Li et al. [27] propose a physical-and-congestion-aware packing algorithm, which achieves the co-optimization

in wirelength and routability. Elgammal and Betz [28] propose a re-clustering technique to improve the wirelength and critical path delay through dynamically updating the packed netlist during the packing and placement processes. Yu et al. [29] propose a quantitative rule to guide the packing process for neural networks, thereby improving the performance of the final FPGA design. Chen et al. [30] propose DP-Pack to perform packing in parallel. Rajarathnam et al. [31] propose DREAMPlaceFPGA-PL, which uses GPU to accelerate the packing process.

However, the above-mentioned works are designed for traditional SRAM-based FPGAs. Since NVFPGA introduces new characteristics, the CAD flow should also be modified to bridge the architectural differences and address the challenges. Liu et al. [26] propose a performance-aware packing algorithm for MLC NVFPGAs, which addresses the performance asymmetry of MLC NVM and reduces the critical path delay. However, it does not address the issue of packing efficiency degradation caused by MLC LUTs.

In this paper, focusing on the packing efficiency for MLC NVFPGAs, we propose a re-pair stage to reduce the number of invoking the time-consuming intra-routing process.

III. Routability-Aware Re-Pair for Packing Optimization

In this section, we will introduce the proposed routability-aware re-pair based packing optimization method for MLC NVFPGAs in detail.

A. Motivation and Main Idea

To look into details of packing in MLC FPGAs, we conduct preliminary experiments on a series of representative test benches and find that the Route-Each stage occupies 91.16% of the total packing time. The statistics also show that 73.53% of CLBs cannot obtain routable solutions at the Route-Once stage, resulting in extensive invocations of the time-consuming Route-Each stage. Thus, the packing efficiency can be improved if the number of CLBs entering Route-Each is reduced.

When tracking packing solutions of CLBs, we found that some CLBs are unroutable at the Routh-Once stage, but become routable after the Routh-Each stage without changes in LUT selection. The difference lies in the logical-physical LUT assignments.

In the Route-Once stage, each LUT is assigned to the first physical LUT selected by the packing algorithm, resulting in imbalanced logic distribution and poor routability. On the contrary, in the Route-Each stage, these congested LUTs can be assigned to other physical LUTs if the previous attempts fail, as the routability can be checked immediately. Thus, the Route-Each stage keeps congestion-free throughout the stage.

These initially unroutable CLBs can become routable through adjusting the logical-physical LUT pairs, so that the extensive intra-routing of the Route-Each stage can be reduced. According to the statistical results, there are on average 30.30% of initially unroutable CLBs select the same LUTs in the subsequent Route-Each stage, indicating a promising optimization potential.

Algorithm 1 The Proposed Re-Pair Stage

Require: Congestion information from previous routing attempt C_{prev}; List of LUTs packed in the previous stage.
Ensure: PACKING_STATUS
// Candidate LUT Selection
1: Initialize CPI for each LUT according to C_{prev};
2: Initialize N_{sel} based on size of available physical LUTs;
3: Initialize stack $CLUT$ for selected candidate LUTs;
4: **while** $N_{sel} > 0$ **do**
5: Select LUT_{sel} from unmarked BLE with the max CPI;
6: $CLUT.push(LUT_{sel})$;
7: $N_{sel} = N_{sel} - Sizeof(LUT_{sel})$;
8: Mark the corresponding BLE;
9: **if** all congested BLEs are marked **then**
10: **break;**
11: **end if**
12: **end while**
// Candidate LUT Reassignment
13: **while** $!CLUT.empty()$ **do**
14: $LUT_c = CLUT.pop()$;
15: **while** LUT_c has feasible physical LUT **do**
16: Select physical LUT P with the min RS;
17: **if** P is duplicate **then**
18: **continue;**
19: **end if**
20: Assign LUT_c in P;
21: **if** $Try_route_CLB()$ is *TRUE* **then**
22: Record LUT_c's new assignment;
23: **break;**
24: **else if** P is a $(k-2)$-LUT **then**
25: Set $(k-2)$-LUTs within P's BLE as duplicate;
26: **end if**
27: **end while**
28: **if** reassigning LUT_c fails **then**
29: PACKING_STATUS = FAIL;
30: **break;**
31: **end if**
32: **end while**
33: **return** PACKING_STATUS;

The main idea of the proposed re-pair stage is to consider intra-routability and utilize unoccupied physical LUTs in the previous failed packing solution to mitigate intra-CLB congestion. Specifically, congested LUTs will be paired to BLEs with fewer logics so that routability can be improved. If a CLB successfully becomes routable through re-pair, it will save a lot of time compared with entering the Route-Each stage.

The re-pair stage presented in Fig. 2 (c) can be formulated as Algorithm 1, which will be invoked after the Route-Once stage fails and before the Route-Each stage. It consists of two main procedures, i.e., candidate LUT selection and candidate LUT reassignment. In addition, we propose an input-equivalent LUT removal technique to improve the overall efficiency of the packing process.

B. Candidate LUT Selection

The first procedure of re-pair is candidate LUT selection (Lines 1-12). The algorithm selects LUTs that cause congestion during intra-routing. These candidate LUTs will

be reassigned to feasible physical LUTs in the subsequent procedure. A candidate LUT should at least meet the following two requirements:

- This LUT causes congestion during intra-routing.
- There are other free physical LUTs in this CLB that can fit this LUT.

For the first requirement, we have recorded the congestion information from the previous failed routing attempt, denoted as C_{prev} in Algorithm 1. However, since each congestion involves at least two LUTs, it is difficult to determine exactly which one causes congestion. Thus, we use each LUT's accumulated number of congestion to evaluate its contribution to the overall congestion. And the algorithm tends to select LUTs that encounter congestion more frequently.

Considering the second requirement, the algorithm tends to select LUTs with fewer inputs, as they offer higher flexibility when searching for alternative physical LUTs. Therefore, in the proposed re-pair algorithm, we use accumulated congestion per input (CPI) for each LUT to evaluate its priority in candidate selection (Line 1), as Equation 1.

$$CPI = \frac{N_{congestion}}{N_{input}} \tag{1}$$

Here, $N_{congestion}$ is the number of times this LUT has been involved in congestion with other LUTs, and N_{input} is the number of inputs of this LUT. When selecting candidate LUTs, the LUT with the highest CPI will be selected (Lines 5-6), and the corresponding BLE will be marked (Line 8). For fairness considerations, the subsequent candidate LUTs will not be selected from these marked BLEs.

In addition, the number of candidates should also be carefully determined. If too many candidates are selected, the exploration space of re-pair will be too big to be efficiently searched. On the contrary, selecting too few LUTs could lead to omissions and result in a failed re-pair attempt. Therefore, in the proposed re-pair stage, the number of candidate LUTs is restricted by the logic size of available physical LUTs, denoted as N_{sel} in Algorithm 1. Specifically, we define the logic size of a k-LUT as 4, a $(k-1)$-LUT as 2, and a $(k-2)$-LUT as 1. The value of N_{sel} equals the sum of logic size of unoccupied LUTs in the previous packing solution (Line 2). During the selection procedure, the value of N_{sel} decreases each time an LUT is selected, with the variation determined by the size of the selected LUT (Line 7). The candidate selection ends when N_{sel} is exhausted or all BLEs that encounter congestion are marked (Lines 4, 9-11).

C. Candidate LUT Reassignment

After all candidate LUTs are settled, the algorithm will reassign them back into the CLB, and determine whether a routable packing solution can be obtained (Lines 13-32).

The selected candidate LUTs are stored in a stack, denoted as $CLUT$. These candidate LUTs will be packed back in the reversed order. The algorithm will try pairing each candidate LUT with feasible physical LUTs (Lines 13-15).

To ensure the solution generated by the re-pair stage is always routable, the algorithm verifies its routability (Line 21). However, the routability check is time-consuming. Therefore, when selecting a physical LUT, the logical-physical LUT pair with potentially better intra-routability should be verified first to avoid invoking the intra-routing process as much as possible.

To incorporate routability-awareness into the reassignment procedure, we evaluate the routability through investigating the distribution of logics. Specifically, we use the number of BLE's occupied inputs to estimate its logic density. For a new CLB packing solution generated by the re-pair process, we define its routability score as Equation 2.

$$RS = \sum_{i=1}^{n_{BLE}} input_i^3 \tag{2}$$

Here, $input_i$ is the number of occupied inputs of the $i-th$ BLE in the CLB, and n_{BLE} is the number of BLEs in the CLB. The proposed RS amplifies the penalty for those BLEs that fully use their inputs, which are likely to encounter congestion in routing. A smaller RS indicates better routability for the corresponding packing solution.

In the routability-aware candidate reassignment procedure, the algorithm will select the physical LUT P with the smallest RS (Line 16). After pairing the candidate logical LUT LUT_c with the selected physical LUT P, intra-routing process will be performed to validate whether the current packing solution is routable (Lines 20-21). If routable, the logical-physical LUT pair will be recorded, and the procedure will proceed to handle the next candidate LUT (Lines 22-23). Otherwise, it will proceed to check the next feasible physical LUT. If one of the candidate LUTs cannot find any routable reassignments, the re-pair stage will be terminated immediately, and the Route-Each stage in the original packing algorithm will be invoked (Lines 28-31).

D. Input-Equivalent LUT Removal

As we introduce in Section II, an MLC k-LUT can be used as at most 4 independent $(k-2)$-LUTs. According to the structure demonstrated in Fig. 1 (d), for each single $(k-2)$-LUT in the same MLC LUT, its accessible inputs, i.e., the inputs from A_1 to A_{k-2}, are exactly the same. The rest of the inputs, i.e., A_k and A_{k-1}, are used to control the multiplexers for output selection. Thus, these physical $(k-2)$-LUTs within the same BLE are equivalent from the perspective of input.

Our experimental observation indicates that congestion usually occurs among inputs instead of outputs during the packing process, since the routing resources for outputs are relatively sufficient. Each MLC LUT preserves 4 outputs, as it can store at most 4 logical LUTs. However, when packing, an MLC LUT can hardly be fully occupied by 4 logical LUTs due to the input constraint, where the true bottleneck lies. Thus, when analyzing intra-routability, we only focus on the situation of input occupations.

For these input-equivalent LUTs, if pairing a candidate LUT to one of them does not result in a routable solution, it would

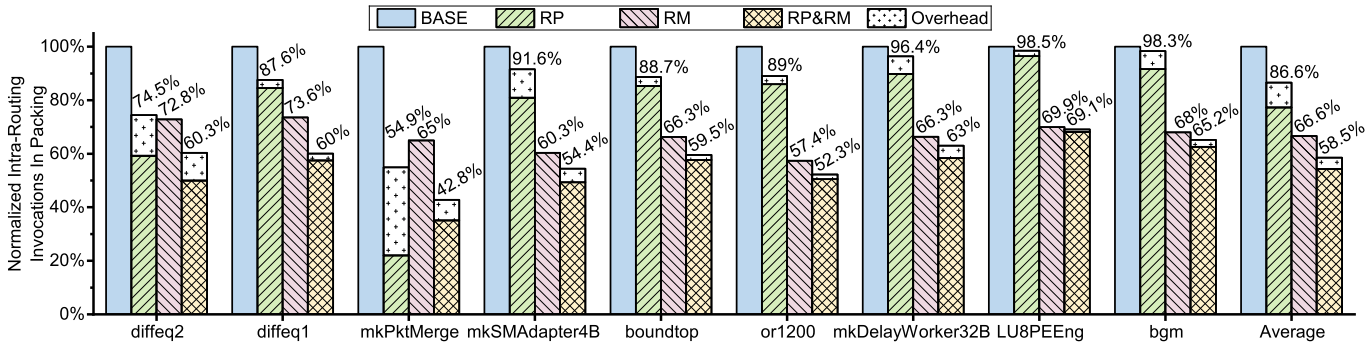


Fig. 3. Normalized intra-routing invocations of different packing methods on various circuits.

also not succeed for the rest of them. Thus, to improve the efficiency of the re-pair process, filtering input-equivalent LUTs from the physical LUT selection is a promising approach.

It should be noted that the input-equivalence only exists among $(k-2)$-LUTs, as a k-LUT or $(k-1)$-LUT would not be equivalent to a $(k-2)$-LUT. When a k-LUT or $(k-1)$-LUT is configured through combining multiple fractured $(k-2)$-LUTs, it enables access to the inputs A_k and A_{k-1}. Since these extra inputs offer higher flexibility in pin assignment, the inputs are no longer equivalent to those of $(k-2)$-LUTs, even in the same BLE.

To identify the input-equivalent physical LUTs, after a packing attempt fails, if the physical LUT being assigned is a $(k-2)$-LUT, the algorithm will mark the other $(k-2)$-LUTs within the same BLE as duplicate (Lines 24-26). When a duplicate physical LUT is selected later, the algorithm will skip it and proceed to fetch the next feasible physical LUT (Lines 17-19). Similarly, this technique can also be applied to the Route-Each stage to further enhance the overall packing process efficiency.

IV. Experiment

In this section, we will evaluate the proposed techniques and report the efficiency improvements in the packing process. Besides, we analyze the overhead and the effects on the quality-of-results (QoR) of the proposed scheme.

A. Experiment Setup

In the evaluation, the proposed techniques are tested on an NVFPGA architecture that uses STT-RAM based MLC LUTs to build up its CLBs. In the experiment, the number of MLC LUT's inputs, i.e., k, is set to 7. This FPGA architecture[1] is based on Altera Stratix IV device.

Our proposed re-pair method is integrated into the widely-used Verilog-To-Route (VTR) CAD flow [25]. It will be activated after the Route-Once stage of the packing process.

We implement several packing methods for comparison.

- **BASE**: It uses the packing algorithm proposed in [26], including two stages, i.e., Route-Once and Route-Each.

[1]The architecture file is *k6_frac_N10_mem32k_40nm.xml* (height: 100 width: 100), which has 7500 CLBs and 208 BRAMs in total.

TABLE I
Benchmark Information

Circuit	Number of LUTs				FFs	CLBs
	2/3/4-LUT	5-LUT	6-LUT	7-LUT		
diffeq2	53	57	44	81	96	20
diffeq1	185	26	31	136	193	28
mkPktMerge	172	10	19	15	36	16
mkSM-Adapter4B	580	275	251	608	983	145
boundtop	1362	248	337	678	1671	191
or1200	835	547	376	914	691	231
mkDelay-Worker32B	1771	1227	930	770	2491	394
LU8PEEng	5705	2614	2184	9601	6630	1983
bgm	8068	4415	4662	11923	5362	2931

- **RP (Re-Pair)**: On the basis of **BASE**, it implements the proposed re-pair stage, which is invoked when the Route-Once stage fails.
- **RM (ReMoval)**: On the basis of **BASE**, it adopts the input-equivalent LUT removal technique.
- **RP&RM**: On the basis of **BASE**, it integrates both the re-pair stage and the input-equivalent LUT removal technique in packing.

A series of benchmark circuits, as listed in Table I, are adopted for evaluation, which have different complexities in terms of scale of circuits. The composition of LUT sizes also differs among them. For example, 79.63% of LUTs in $mkPktMerge$ have 4 or fewer inputs. On the contrary, 47.76% and 41.02% are 7-LUTs in $LU8PEEng$ and bgm, respectively.

B. Improvement in Packing Efficiency

With our observation, the packing time is basically proportional to the number of intra-routing invocations, so we use it to evaluate the packing efficiency. Fig. 3 shows the normalized number of intra-routing invocations during the packing process for various circuits. In addition, for methods that involve the re-pair stage, i.e., RP and RP&RM, we specifically highlight the intra-routing invocations of the re-pair stage, i.e., the Overhead in Fig. 3. On average, RP and RM achieve 13.39% and 33.37% packing efficiency improvements, respectively. The result indicates that both the re-pair stage and the input-equivalent LUT removal technique can effectively improve the efficiency of the packing process. These two

techniques are orthogonal and can work together to achieve better improvements. On average, RP&RM achieves 41.48% less intra-routing invocations compared with BASE.

The effectiveness of the re-pair stage is influenced by the circuit's distribution of LUT sizes. Among the circuits we tested, RP achieves the best efficiency improvement in $mkPktMerge$. This circuit contains a lot of CLBs composed of 3-LUTs. These LUTs are gathered to several BLEs while the rest of BLEs are still empty due to the CLB input constraint. Thus, these CLBs can easily become routable through re-pair. On the contrary, for circuits like $LU8PEEng$ and bgm, which both contain more 7-LUTs, the improvements are less significant. The logical-physical assignments of these 7-LUTs are hard to be adjusted, due to the lack of available physical 7-LUTs.

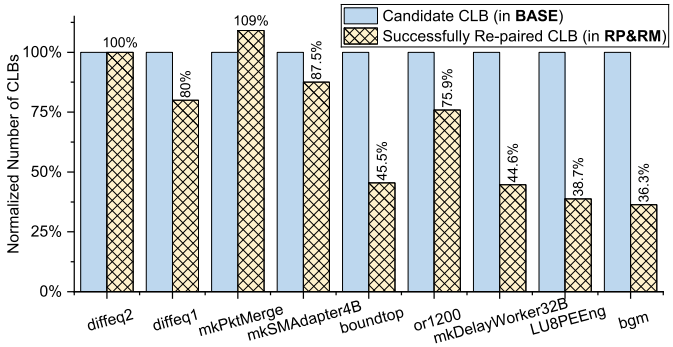


Fig. 4. Normalized number of candidate and successfully re-paired CLBs.

Fig. 4 compares the number of successfully re-paired CLBs (in RP&RM) and candidate CLBs (in BASE), i.e., CLBs that contain same LUTs in both the Route-Once stage and the Route-Each stage. On average, the proposed method successfully handles 68.62% of the candidate CLBs, which verifies the effectiveness of the proposed routability-aware re-pair technique. It should be noted that a successfully re-paired CLB may generate a different packing solution from the one in BASE, which changes the follow-up packing process so the subsequent CLBs are different. Thus, it is possible for the number of successfully re-paired CLBs to exceed the candidate CLBs, as observed in $mkPktMerge$.

The experiment results show that the proposed re-pair stage can effectively identify the re-pairable CLBs and make them routable with less intra-routing invocations.

C. Overhead Analysis

The proposed re-pair stage also involves intra-routing invocations, which are referred to as overhead. As shown in Fig. 3, on average, the re-pair stage contributes 4.15% of the total number of intra-routing invocations. However, the original packing process needs $4.45\times$ more intra-routing invocations to pack these CLBs through the Route-Each stage. Thus, the re-pair stage delivers significant packing time reduction with little overhead.

To reduce the overhead, we propose RS to guide the reassignment procedure, and define LUT input-equivalence to filter unnecessary intra-routing attempts. As Fig. 5 shows, the

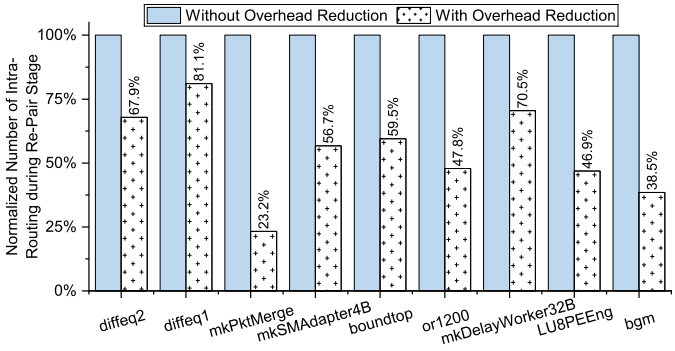


Fig. 5. Normalized number of intra-routing invocations in re-pair stage.

TABLE II
THE AREA AND PERFORMANCE EVALUATION.

Circuit	Area(#CLBs)			Performance(ns)		
	BASE	RP&RM	Ratio	BASE	RP&RM	Ratio
diffeq2	20	20	100%	16.41	16.34	99.58%
diffeq1	28	27	96.43%	21.15	21.32	100.8%
mkPktMerge	16	16	100%	4.041	3.980	98.50%
mkSM-Adapter4B	145	144	99.31%	5.898	5.743	97.38%
boundtop	192	191	99.48%	6.472	6.169	95.31%
or1200	231	230	99.57%	12.68	12.57	99.13%
mkDelay-Worker32B	394	393	99.75%	8.080	7.455	92.27%
LU8PEEng	1990	1979	99.45%	103.4	103.2	99.76%
bgm	2927	2926	99.97%	24.55	24.97	101.7%
Average	**660**	**658**	**99.33%**	**22.53**	**22.42**	**98.27%**

above-mentioned overhead reduction techniques achieve an average reduction of overhead by 54.69%.

D. Design Quality Evaluation

Since the re-pair stage potentially generates a different packing solution, the final design may be different in area and performance. Table II shows the QoR of BASE and RP&RM, where area is estimated using the number of CLBs, and the performance is evaluated using the delay of the critical path. The proposed method improves the area by 0.67% and the performance by 1.73%, on average. The results indicate that, the proposed techniques would not degrade the QoR, but lead to slight QoR improvement.

V. CONCLUSION

In this paper, we propose a routability-aware re-pair strategy to improve the efficiency of the packing process in the CAD flow for MLC NVFPGAs. To reduce the intra-routing invocations, we propose the re-pair stage, which contains two procedures, i.e., candidate LUT selection and candidate LUT reassignment. To incorporate the routability-awareness into the re-pair stage, we propose a routability evaluation technique to guide the candidate LUT reassignment. In addition, leveraging the input equivalence of some physical LUTs, we identify and remove the unnecessary intra-routing attempts during packing, thereby further improving the overall efficiency. We conduct experiments to evaluate the efficiency improvements of the proposed techniques and give discussions. The experimental results show an average packing time reduction of 41.48%.

REFERENCES

[1] G. Singh, M. Alser, D. S. Cali, D. Diamantopoulos, J. Gómez-Luna, H. Corporaal, and O. Mutlu, "FPGA-based near-memory acceleration of modern data-intensive applications," *International Symposium on Microarchitecture (MICRO)*, vol. 41, no. 4, pp. 39–48, 2021.

[2] H. Zhang, J. Zuo, H. Zheng, S. Liu, M. Luo, and M. Zhao, "Implementing neural networks on nonvolatile FPGAs with reprogramming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 11, pp. 3961–3972, 2024.

[3] K. Zeng, Q. Ma, J. W. Wu, Z. Chen, T. Shen, and C. Yan, "FPGA-based accelerator for object detection: a comprehensive survey," *The Journal of Supercomputing (TJSC)*, vol. 78, no. 12, pp. 14 096–14 136, 2022.

[4] N. Zhang, X. Wei, H. Chen, and W. Liu, "FPGA implementation for CNN-based optical remote sensing object detection," *MDPI Electronics*, vol. 10, no. 3, p. 282, 2021.

[5] Y. Zhu, X. Wang, L. Ju, and S. Guo, "FxHENN: FPGA-based acceleration framework for homomorphic encrypted CNN inference," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2023, pp. 896–907.

[6] A. A. Yazdeen, S. R. Zeebaree, M. M. Sadeeq, S. F. Kak, O. M. Ahmed, and R. R. Zebari, "FPGA implementations for data encryption and decryption via concurrent and parallel computation: A review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 8–16, 2021.

[7] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of FPGA-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2024.

[8] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on FPGAs," in *Symposium on Field Programmable Gate Arrays (FPGA)*, 2024, pp. 223–234.

[9] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *International Symposium on Microarchitecture (MICRO)*, 2016, pp. 17:1–17:12.

[10] A. Boutros, E. Nurvitadhi, R. Ma, S. Gribok, Z. Zhao, J. C. Hoe, V. Betz, and M. Langhammer, "Beyond peak performance: Comparing the real performance of AI-optimized FPGAs and GPUs," in *International Conference on Field-Programmable Technology (FPT)*, 2020, pp. 10–19.

[11] A. Tauhid, L. Xu, M. Rahman, and E. Tomai, "A survey on security analysis of machine learning-oriented hardware and software intellectual property," *Elsevier High-Confidence Computing (HCC)*, pp. 100–114, 2023.

[12] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. R. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *Communications of the ACM (CACM)*, vol. 59, no. 11, pp. 114–122, 2016.

[13] Y. Chen, J. Zhao, and Y. Xie, "3D-nonFAR: three-dimensional non-volatile FPGA architecture using phase change memory," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2010, pp. 55–60.

[14] K. Huang, Y. Ha, R. Zhao, A. Kumar, and Y. Lian, "A low active leakage and high reliability phase change memory (PCM) based non-volatile FPGA storage element," *IEEE Transactions on Circuits and Systems I: Regular Papers (TCASI)*, vol. 61, no. 9, pp. 2605–2613, 2014.

[15] P. Narayanan, S. Ambrogio, A. Okazaki, K. Hosokawa, H. Tsai, A. Nomura, T. Yasuda, C. Mackin, S. C. Lewis, A. Friz *et al.*, "Fully on-chip MAC at 14 nm enabled by accurate row-wise programming of PCM-based weights and parallel vector-transport in duration-format," *IEEE Transactions on Electron Devices (TED)*, vol. 68, no. 12, pp. 6629–6636, 2021.

[16] S. Paul, S. Mukhopadhyay, and S. Bhunia, "Hybrid CMOS-STTRAM non-volatile FPGA: design challenges and optimization approaches," in *International Conference on Computer Aided Design (ICCAD)*, 2008, pp. 589–592.

[17] B. Khaleghi and H. Asadi, "A resistive RAM-based FPGA architecture equipped with efficient programming circuitry," *IEEE Transactions on Circuits and Systems I: Regular Papers (TCASI)*, vol. 65, no. 7, pp. 2196–2209, 2018.

[18] Y. Long, T. Na, and S. Mukhopadhyay, "ReRAM-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 26, no. 12, pp. 2781–2794, 2018.

[19] H. P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, "Metal-oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[20] K. Liu, M. Zhao, L. Ju, Z. Jia, C. J. Xue, and J. Hu, "Design exploration for multiple level cell based non-volatile FPGAs," in *International Conference on Computer Design (ICCD)*, 2017, pp. 257–264.

[21] J. Cong and B. Xiao, "FPGA-RPI: A novel FPGA architecture with RRAM-based programmable interconnects," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 22, no. 4, pp. 864–877, 2013.

[22] D. Choi, K. Choi, and J. D. Villasenor, "New non-volatile memory structures for FPGA architectures," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16, no. 7, pp. 874–881, 2008.

[23] H. Ochi, K. Yamaguchi, T. Fujimoto, J. Hotate, T. Kishimoto, T. Higashi, T. Imagawa, R. Doi, M. Tada, T. Sugibayashi *et al.*, "Via-switch FPGA: Highly dense mixed-grained reconfigurable architecture with overlay via-switch crossbars," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 26, no. 12, pp. 2723–2736, 2018.

[24] L. Ju, X. Sui, S. Li, M. Zhao, C. J. Xue, J. Hu, and Z. Jia, "NVM-based FPGA block RAM with adaptive SLC-MLC conversion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 11, pp. 2661–2672, 2018.

[25] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J. Legault, E. Sha, A. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 9:1–9:55, 2020.

[26] K. Liu, M. Zhao, L. Ju, Z. Jia, J. Hu, and C. J. Xue, "Applying multiple level cell to non-volatile FPGAs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 4, pp. 27:1–27:22, 2020.

[27] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 4, pp. 869–882, 2018.

[28] M. A. Elgammal and V. Betz, "Breaking boundaries: Optimizing FPGA CAD with flexible and multi-threaded re-clustering," in *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2023, pp. 11–18.

[29] L. Yu, B. Guo, T. Zhi, and L. Bai, "Improving seed-based FPGA packing with indirect connection for realization of neural networks," *MDPI Electronics*, vol. 12, no. 12, p. 2691, 2023.

[30] Q. Chen, M. Shen, and N. Xiao, "DP-Pack: Distributed parallel packing for FPGAs," in *International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 282–285.

[31] R. S. Rajarathnam, Z. Jiang, M. A. Iyer, and D. Z. Pan, "DREAMPlaceFPGA-PL: An open-source GPU-accelerated packer-legalizer for heterogeneous FPGAs," in *International Symposium on Physical Design (ISPD)*, 2023, pp. 175–184.

Efficient Recycling Subspace Truncation Method for Periodic Small-Signal Analysis

Yuncheng Xu[1], Fan Yang[2], Yangfeng Su[1]
[1]School of Mathematical Sciences, Fudan University, Shanghai, China
[2]State Key Laboratory of Integrated Chips and Systems, Microelectronics Department, Fudan University, Shanghai, China

***Abstract*—Periodic small-signal analysis is crucial but time-consuming in RF simulation, since it may deal with many frequency points. While the Krylov subspace recycling method has greatly accelerated the simulation, the increasing memory cost in large-scale RF simulation has become a new bottleneck. A remedy for memory shortage is to restart the recycling algorithm, but may cause excessive extra iterations. To address this issue, this paper outlines a framework of recycling subspace truncation method for periodic small-signal analysis, provided with an efficient initial guess choice method and a Floquet-based subspace truncation strategy. Numerical results show that compared to the existing methods, the proposed method achieves up to $2.5\times$ speedup in the same memory cost.**

***Index Terms*—RF simulation, periodic small-signal analysis, Krylov subspace recycling, subspace truncation.**

I. Introduction

PERIODIC small-signal analysis, including PAC (periodic AC), PXF (periodic transfer function), and PNoise (periodic noise), plays a crucial role in understanding the behavior of radio-frequency (RF) circuits operating under repetitive conditions, such as oscillators, switching converters, and communication systems [1]. Periodic small-signal analysis computes the steady-state response of a linear periodically time-varying circuit to a sinusoidal stimulus at different frequencies.

The primary challenge arising in periodic small-signal analysis is how to efficiently solve a sequence of linear systems corresponding to different frequency points [2], [3], because the computational cost of solving all the linear systems increases linearly with the number of frequency points. The authors in [2] transform the original linear systems into a sequence of shifted linear systems, and then the solution of these shifted linear systems can be accelerated using the Krylov subspace recycling method. The recycling generalized conjugate residual (R-GCR) method introduced in [2] is up to forty times faster than the original direct factorization method. A more efficient and robust version of recycling algorithm based on generalized minimal residual (GMRES) method [4] is developed in [3].

However, with the explosive growth of RF circuit scale, the recycled subspace expands fast in frequency sweep, causing excessive amount of memory and computational costs. The issue of memory shortage may diminish the stability and robustness of the previous recycling algorithms. A remedy for memory shortage is to restart the whole recycling algorithm, but may cause excessive extra iterations. In the field of numerical algebra, the GCRO-DR method [5] is a well-known recycling method with subspace truncation to solve a sequence of linear systems, but this method does not take full advantage of the special properties in periodic small-signal analysis.

In this paper, we outline a framework of recycling subspace truncation method for periodic small-signal analysis to address the issue of memory shortage. Previous subspace recycling methods for periodic small-signal analysis are applicable in our framework. Two core problems in our framework are: (1) how to choose a good initial guess from the last system; (2) what vectors should be retained in the recycled subspace. Inappropriate choice of initial guess and recycled subspace may cause excessive number of additional iterations, or even stagnate the convergence of Krylov subspace iterative methods.

For the first problem, we propose an optimal initial guess recycling method to obtain a better initial guess for the next system. For the second problem, we develop a Floquet-based subspace truncation strategy to reduce the extra iterations caused by subspace truncation. We have observed that the coefficient matrix can be identified as the discretization of the state-transition matrix of a linear periodically time-varying system. The Floquet theory in [6] can be applied to characterize the eigenvalue distribution of the state-transition matrix. Based on the eigenvalue distribution, we reveal that the subspace corresponding to the largest Floquet multipliers significantly affect the convergence of iterative methods, and then we can establish the formulation of our Floquet-based subspace truncation strategy.

The paper is organized as follows. In Section II, we review the derivation of the shifted linear systems in periodic small-signal analysis, and then present our observation of the eigenvalue distribution of the coefficient matrix based on the Floquet theory. In Section III, we outline a framework of recycling subspace truncation method for periodic small-signal analysis, provided with our optimal initial guess recycling method and Floquet-based subspace truncation strategy. In Section IV, we apply the proposed method to several examples to demonstrate the effectiveness of our method. Finally, conclusions are given in Section V.

This research is supported by National Key R&D Program of China 2020YFA0711900, 2020YFA0711902.

II. Shifted Linear Systems in Small-Signal Analysis

A. Derivation

Using modified nodal analysis [7], the RF circuit can be characterized by the differential equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(q(v(t))\right) + i(v(t)) + u(t) = 0, \tag{1}$$

where all the vectors are in $\mathbb{R}^N$, $v(t)$ is the node voltages, and $i(v(t)), q(v(t))$ are the resistive node currents and node charges respectively.

Suppose the periodic operating point $v_L(t)$ (with periodic T_L) has been computed by the periodic steady-state (PSS) analysis [1]. Linearizing the circuit equations (1) around $v_L(t)$ yields the linear periodically time-varying system of small-signal response $v_s(t)$ [2],

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(C(v_L(t))v_s(t)\right) + G(v_L(t))v_s(t) + u_s(t) = 0, \tag{2}$$

where

$$C(v_L(t)) = \frac{\mathrm{d}q}{\mathrm{d}v}\Big|_{v=v_L(t)}, \quad G(v_L(t)) = \frac{\mathrm{d}i}{\mathrm{d}v}\Big|_{v=v_L(t)}.$$

From the theory of periodically time-varying systems [6], it is known that the steady-state small-signal response $v_s(t)$ satisfies the periodic boundary condition

$$v_s(t+T_L) = v_s(t)\sigma(f_s), \quad \sigma(f_s) = \exp(2\pi j(f_s/f_L)).$$

Now we consider solving $v_s(t)$ using time-domain discretization methods (backward Euler, Gear2 or others). Given a partition of the interval $[0, T_L]$ as $0 = t_0 < t_1 < \cdots < t_M = T_L$. Let $h_i = t_i - t_{i-1}$, $C_i = C(v_L(t_i))$ and $G_i = G(v_L(t_i))$. For each t_i, the approximate value of $v_s(t_i)$ is denoted by $v_{s,i}$. For simplicity, we use backward Euler method to discretize (2) at $t_i (1 \le i \le M)$. After assembling the M equations at each time point, we can obtain the linear systems

$$(\sigma(f_s)L + B)\begin{bmatrix} v_{s,1} \\ v_{s,2} \\ \vdots \\ v_{s,M} \end{bmatrix} = -\sigma(f_s)u_s(f_s), \tag{3}$$

where

$$L = \begin{bmatrix} \frac{C_1}{h_1} + G_1 & & & \\ -\frac{C_1}{h_2} & \frac{C_2}{h_2} + G_2 & & \\ & \ddots & \ddots & \\ & & -\frac{C_{M-1}}{h_M} & \frac{C_M}{h_M} + G_M \end{bmatrix},$$

$$B = \begin{bmatrix} 0 & \cdots & 0 & -\frac{C_M}{h_1} \\ 0 & \cdots & 0 & 0 \\ \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{bmatrix}.$$

Take L as the preconditioner and transform the original linear system to a shifted linear system

$$(\sigma(f_s)I_{MN} + L^{-1}B)v_s = -\sigma(f_s)L^{-1}u_s(f_s).$$

Note that the coefficient matrix $\sigma I_{MN} + L^{-1}B$ has the block structure

$$\sigma I + L^{-1}B = \begin{bmatrix} \sigma I_N & \cdots & 0 & -P_1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \sigma I_N & -P_{M-1} \\ 0 & \cdots & 0 & \sigma I_N - P_M \end{bmatrix}. \tag{4}$$

Hence, one can first solve $v_{s,M}$ from the shifted linear system

$$(\sigma(f_s)I_N - P_M)v_{s,M} = -\sigma(f_s)(L^{-1}u_s(f_s))_M. \tag{5}$$

Then the remaining $v_{s,i}$ for $i = M-1, M-2, \cdots, 1$ can be obtained through backward substitution. After simplifying the notations, the main task of the periodic small-signal solver is to efficiently solve a sequence of shifted linear systems

$$(\sigma^{(k)}I_N - P_M)x = b^{(k)}, \quad \sigma^{(k)} = \exp(2\pi j(f_s^{(k)}/f_L)),$$

corresponding to a sequence of frequency points $f_s^{(k)}$.

B. Relation between P_M and state-transition matrix

Denote $\Phi(t, s)$ as the state-transition matrix of the linear periodically time-varying system of DAEs

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(C(t)v(t)\right) + G(t)v(t) = 0. \tag{6}$$

Note that (6) is exactly the homogeneous part of (2).

Suppose we are now required to perform a transient simulation on the homogeneous system (6) with the initial condition $\tilde{v}(0)$. Using the same discretization method as in Section II-A, the approximate solution $\tilde{v}(t)$ at $t = t_i$ can be obtained as

$$\tilde{v}(t_i) = P_i\tilde{v}(0), \quad 1 \le i \le M,$$

where P_i is defined in (4). In fact, performing one preconditioner L on (3) can be thought of as performing a transient simulation over one period.

In particularly, we have $\tilde{v}(T_L) = P_M\tilde{v}(0)$. Hence, P_M can be considered as the discretization of the state-transition matrix $\Phi(T_L, 0)$. Therefore, it can be predicted that if the time step h_i is small enough, P_M may have some properties similar to the state-transition matrix $\Phi(T_L, 0)$.

C. Eigenvalue distribution of P_M

From the Floquet theory of linear periodically time-varying systems of DAEs [6], under some assumptions the state-transition matrix $\Phi(t, s)$ for (6) can be written as

$$\Phi(t, s) = U(t)D(t-s)V(s)^*C(s)$$

where $U(t), V(s) \in \mathbb{C}^{N\times r}$ are T_L-periodic matrices of rank r, and $D(t) = \mathrm{diag}\{\exp(\rho_1 t), \ldots, \exp(\rho_r t)\}$ with all $\rho_i \neq 0$. Here $\rho_1, \ldots, \rho_r$ are called Floquet exponents, and $\lambda_i = \exp(\rho_i T_L)$ are defined as Floquet multpliers. The r Floquet multpliers are exactly all the nonzero eigenvalues of the state-transition matrix $\Phi(T_L, 0)$.

It can be shown (e.g. [8]) that if one of the Floquet multpliers is 1, and the magnitude of other $r-1$ Floquet multpliers is less than 1, then (6) has a asymptotically orbitally stable solution $\dot{v}_L(t)$. All practical stable oscillators satisfy this

condition [6], i.e. all the nonzero eigenvalues of $\Phi(T_L, 0)$ have magnitude less than 1 except one eigenvalue equal to 1.

Since the coefficient matrix P_M can be identified as the discretization of $\Phi(T_L, 0)$, the eigenvalues of P_M may well approximate the Floquet multpliers. Hence, we can predict that P_M may have similar eigenvalue distribution to $\Phi(T_L, 0)$. In numerical experiments, we observe that the maximal magnitude of eigenvalues of P_M is close to 1 (often less than 1) and P_M may have several eigenvalues near 1 (see Section IV-A).

III. Recycling Subspace Truncation Method for Periodic Small-Signal Analysis

A. Previous Krylov Subspace Recycling Methods

We briefly review the previous Krylov subspace recycling methods for periodic small-signal analysis. The details of the algorithms can be seen in [2], [3].

The Krylov subspace recycling methods for a sequence of shifted linear systems are based on the relation

$$(\sigma_2 I - P)\, v = (\sigma_1 I - P)\, v + (\sigma_2 - \sigma_1)\, v. \tag{7}$$

The relation (7) means that if $(\sigma_1 I - P)\, v$ has been computed, then $(\sigma_2 I - P)\, v$ can be obtained just using operations on vectors. Hence, the subspace computed in the last system can be recycled to the next system.

Given recycled subspace U_m, C_m of dimension m satisfying $\left(\sigma^{(k)} I - P\right) U_m = C_m$. The R-GCR method proposed in [2] computes the j-dim augmented subspace V_j, $\tilde{V}_j$ satisfying the relation

$$\left(\sigma^{(k)} I - P\right) [U_m, V_j] = [C_m, \tilde{V}_j] \tag{8}$$

where we call $[U_m, V_j]$ ($[C_m, \tilde{V}_j]$) the total subspace. The dimension of the total subspace is $m + j$. With the relation (8), the approximate solution $x^{(k)}$ and residual $r^{(k)}$ can be updated as

$$\begin{aligned} y_{m+j} &= [C_m, \tilde{V}_j]^* r_0^{(k)}, \\ x^{(k)} &= x_0^{(k)} + [U_m, V_j] y_{m+j}, \\ r^{(k)} &= r_0^{(k)} - [C_m, \tilde{V}_j] y_{m+j}. \end{aligned}$$

Recently, a more robust and efficient recycling method for periodic small-signal analysis called R-GMRES is developed in [3]. R-GMRES computes the following relation

$$\left(\sigma^{(k)} I - P\right) [U_m, V_j] = [C_m, V_{j+1}] \begin{bmatrix} T_m & B_j \\ 0_{(j+1)\times m} & \underline{H}_j \end{bmatrix}. \tag{9}$$

Similar to R-GCR, the approximate solution $x^{(k)}$ and residual $r^{(k)}$ can be updated with the relation (9).

B. A framework of recycling subspace truncation method for periodic small-signal analysis

In the process of R-GCR or R-GMRES, if the total subspace is too large, the issue of memory shortage is encountered. To address the issue, we retain a part of vectors in the recycled subspace and discard others. Here we outline a framework of recycling subspace truncation method for periodic small-signal analysis. The framework is summarized in Algorithm 1. In our framework, either R-GCR or R-GMRES can be used.

In the whole process of frequency sweep, we set the maximal dimension of recycled subspace and augmented subspace to be $m_{\max}$ and $j_{\max}$.

Before starting to perform the recycling algorithm, if we have computed the total subspace $[U_m, V_j]$ from the last system, then the total subspace should be truncated to a new recycled subspace. Here we still denote the new recycled subspace as U_m. The dimension of recycled subspace U_m after truncation should be no larger than $m_{\max}$.

Before starting to solve each linear system, we need to select a good initial guess $x_0^{(k)}$ from the last system (for the first system, $x_0^{(1)}$ is set to be zero). If the initial guess is chosen appropriately, then the initial residual $r_0^{(k)}$ can be computed from the residual $r^{(k-1)}$ of the last system without any matrix-vector multiplication (see Section III-C).

In the process of solving each shifted linear system, the augmented subspace V_j is computed by R-GMRES or R-GCR, and the total subspace is expanded to $[U_m, V_j]$. If the dimension $m + j$ of the total subspace $[U_m, V_j]$ is larger than $m_{\max} + j_{\max}$, then we also need to truncate the total subspace to a recycled subspace of dimension $m_{\max}$. Then R-GMRES or R-GCR is restarted with the new recycled subspace after truncation.

Algorithm 1 A framework of recycling subspace truncation method for periodic small-signal analysis

Input: U_m, V_j: the recycled subspace and augmented subspace from the last system;
$m_{\max}$: the maximal dimension of recycled subspace;
$j_{\max}$: the maximal dimension of augmented subspace;
tol: tolerance of relative residual.

Output: Approximate solutions $x^{(k)}$ for shifted linear system $\left(\sigma^{(k)} I - P\right) x = b^{(k)}$.

1: Truncate the total subspace $[U_m, V_j]$ to a new recycled subspace U_m of dimension $m \le m_{\max}$.
2: Obtain a good initial guess $x_0^{(k)}$ and compute the initial residual $r_0^{(k)}$.
3: **while** $\left\|r^{(k)}\right\|_2 / \left\|b^{(k)}\right\|_2 > \text{tol}$ **do**
4: Perform the recycling algorithm R-GMRES or R-GCR to compute the augmented subspace V_j, and update the approximate solution $x^{(k)}$ and residual $r^{(k)}$.
5: **if** $m + j > m_{\max} + j_{\max}$ **then**
6: Truncate the total subspace $[U_m, V_j]$ to a new recycled subspace U_m of dimension $m = m_{\max}$.
7: Restart R-GMRES or R-GCR with the truncated recycled subspace.

In the entire simulation process, the main memory consumption in storing the total subspace is to store at most $(2m_{\max} + j_{\max} + 1)$ vectors of dimension N for R-GMRES, or $2(m_{\max} + j_{\max})$ vectors for R-GCR.

Two core problems in our framework are: (1) how to choose a good initial guess from the last system; (2) what vectors

should be retained in the new recycled subspace. For the first problem, our optimal initial guess recycling method is discussed in Section III-C. For the second problem, we propose a Floquet-based subspace truncation strategy in Section III-D.

The effectiveness of the recycling method may depend on the order in which the small-signal frequencies are arranged. In our numerical experiments (see Section IV), for simplicity, the small signal frequencies f_s are arranged in aescending order. An effective sweeping order can be seen in [3].

C. Optimal initial guess recycling method

In this section, we discuss how to select a good initial guess. We present our optimal initial guess recycling method.

Suppose we have computed the approximate solution $x^{(k-1)}$ and the residual $r^{(k-1)}$ of the last system. The traditional approach (for example, in GCRO-DR [5]) is to directly take the initial guess $x_0^{(k)} = x^{(k-1)}$. However, in most cases, this is not a good choice. Our idea is to introduce a new parameter α, and take $x_0^{(k)} = \alpha x^{(k-1)}$ as the initial guess. The optimal parameter α is determined s.t. the initial residual $r_0^{(k)}$ is minimized, i.e.

$$\alpha = \arg\min_{\alpha\in\mathbb{C}} \left\| b^{(k)} - \left(\sigma^{(k)} I - P\right)\left(\alpha x^{(k-1)}\right)\right\|_2 .$$

Denote $w = \left(\sigma^{(k)} I - P\right) x^{(k-1)}$. From the relation (7), w can be computed only using vector additions and scalar multiplications as

$$w = \left(b^{(k-1)} - r^{(k-1)}\right) + \left(\sigma^{(k)} - \sigma^{(k-1)}\right) x^{(k-1)}. \quad (10)$$

Then the optimal parameter α can be determined by the least-squares method as

$$\alpha = w^* b^{(k)} / w^* w. \quad (11)$$

The algorithm is decribed in Algorithm 2.

Algorithm 2 Optimal initial guess recycling method

Input: $x^{(k-1)}$: approximate solution of the last system;
$r^{(k-1)}$: residual of the last system.
Output: Initial guess $x_0^{(k)}$ and initial residual $r_0^{(k)}$.
1: Compute w as in (10).
2: Compute α as in (11).
3: Compute initial guess $x_0^{(k)} = \alpha x^{(k-1)}$
4: Compute initial residual $r_0^{(k)} = b^{(k)} - \alpha w$.

Note that the residual $r_0^{(k)}$ is computed without any matrix-vector multiplication and the computational cost of our initial guess method involves only 2 inner-products plus no additional matrix-vector multiplication, but it can save several iterations in iterative algorithms (see Section IV-B).

In fact, since the order of the small-signal frequencies can be arbitray, the initial guess gained from the last system may not be the best. A better choice is to obtain the initial guess from the computed system corresponding to the nearest small-signal frequency.

D. Floquet-based subspace truncation strategy

In this section, we discuss our Floquet-based subspace truncation strategy. We have observed that inappropriate subspace truncation strategy may result in excessive additional iterations. Hence, an effective truncation strategy is required.

The convergence of Krylov subspace methods depends to a large degree on the eigenvalue distribution [9]. Indeed, small eigenvalues may greatly stagnate the convergence. Hence, in GCRO-DR [5], the authors retain the harmonic Ritz vectors corresponding to the smallest harmonic Ritz values. This strategy is applicable to general cases, but for the shifted linear systems arising in small-signal analysis, a better truncation strategy can be obtained based on Floquet multipliers.

From the observation in Section II-C, P_M has some eigenvalues λ whose magnitude $|\lambda| \approx 1$. Since $\sigma(f_s) = \exp(2\pi j(f_s/f_L))$, then for some small-signal frequency f_s, the coefficient matrix $\sigma(f_s)I - P$ may have eigenvalues near 0. In this situation, $\sigma(f_s)I - P$ is ill-conditioned, and the Krylov subspace iterative methods may need more iterations to converge.

Returning to the original RF circuits, these eigenvalues close to 1 correspond to slow time constants [10]. In the circuit with slow time constants, the response may be mixed with slowly decaying parts. Hence, the iterative methods need more iterations to detect these parts.

From the above analysis, our idea is to retain in the recycled subspace the eigenvectors corresponding to the eigenvalues with magnitude near 1. In Section II-C, we have observed that the maximal magnitude of the eigenvalues of P_M is near 1. Hence, these eigenvalues with magnitude near 1 are exactly the largest eigenvalues of P_M (which correspond to the largest Floquet multipliers). The largest eigenvalues can be computed by the Ritz values [11] during the frequency sweep. The detailed algorithm is presented as follows.

Here we take R-GMRES as an example. Suppose we have obtained the relation (9), then the QZ algorithm [12] (for the case where $m = 0$, the QR algorithm [13], [14] is enough) is applied to compute all the eigenpairs (z_i, μ_i) of the matrix pencil (E, F) where

$$E = \begin{bmatrix} (U_m^* C_m) T_m & (U_m^* C_m) B_m + (U_m^* V_{j+1}) \underline{H}_j \\ 0_{j\times m} & H_j \end{bmatrix},$$
$$F = \begin{bmatrix} I_m & U_m^* V_j \\ (U_m^* V_j)^* & I_j \end{bmatrix}. \quad (12)$$

Then choose the $\tilde{m}$ eigenvectors corresponding to the eigenvalues with largest $\left|\sigma^{(k)} - \mu_i\right|$. The $\tilde{m}$ eigenvectors are stored in $Z_{\tilde{m}} \in \mathbb{C}^{(m+j)\times\tilde{m}}$. The new recycled subspace $U_{\tilde{m}}$ can be computed with $Z_{\tilde{m}}$. The algorithm of our Floquet-based subspace truncation strategy is described in Algorithm 3.

The main computational cost of our Floquet-based strategy lies in computing $U_m^* C_m$, $U_m^* V_{j+1}$ and a $(m + j)$-dim generalized eigenvalue problem. Since m and j are usually much smaller than N, the computational cost is a very small portion of the whole solving process.

Algorithm 3 Floquet-based subspace truncation strategy

Input: $\tilde{m}$: dimension of recycled subspace after truncation;
U_m, C_m: the recycled subspace;
V_{j+1}: the augmented subspace.

Output: Recycled subspace after truncation: $U_{\tilde{m}}$, $C_{\tilde{m}}$ satisfying $\left(\sigma^{(k)}I - P\right)U_{\tilde{m}} = C_{\tilde{m}}$.

1: Compute all the eigenpairs (z_i, μ_i) of the matrix pencil defiend in (12).
2: Choose the $\tilde{m}$ eigenvectors corresponding to the eigenvalues with the largest $\left|\sigma^{(k)} - \mu_i\right|$ and store them in $Z_{\tilde{m}}$.
3: Compute $U_{\tilde{m}} = [U_m, V_j]Z_{\tilde{m}}$.
4: Compute $C_{\tilde{m}} = [U_m, V_{j+1}]\left(\begin{bmatrix} T_m & B_j \\ 0 & \underline{H}_j \end{bmatrix} Z_{\tilde{m}}\right)$.

IV. Numerical Experiments

We have tested our proposed method using MATLAB. Test cases are provided by our industrial partner, as shown in Table I. The "Discretization" column in the table describes the discretization scheme. BE(1) means that the backward Euler scheme is applied only to the first time point, while BE(1, 2) applies to the first and second time point. For the accuracy of simulation, Gear2 scheme is applied to other time points. In AC analysis, the frequency sweep usually comes in three types: linear (LIN), octave (OCT) and decade (DEC) [15]. We focus on LIN and DEC in our numerical experiments.

TABLE I
Details of the test cases

Index	# Timesteps	# Nodes	Discretization
case 1	2003	3886	BE(1,2), Gear2
case 2	2451	9482	BE(1,2), Gear2
case 3	729	16224	BE(1,2), Gear2
case 4	801	3058	BE(1), Gear2

In practical computation, the matrix-vector multiplication $P * x$ is implicitly performed. The LU factorization of the diagonal blocks of L is precomputed, so the runtime we present does not include this portion.

All the numerical experiments are implemented in MATLAB R2020b with Intel Xeon Gold 6226R @2.90 GHz. For all the linear systems, the tolerance of relative residual is set to be 10^{-8}.

A. Eigenvalue distribution of P_M

To verify our prediction of the eigenvalue distribution of P_M in Section II-C, we compute all the eigenvalues of P_M using eigs function in MATLAB. The results are shown in Table II. It is observed that the maximal magnitude of P_M's eigenvalues are close to 1 in four test cases. Additionally, P_M has several eigenvalues close to 1, causing that $\sigma(f_s)I - P_M$ has several eigenvalues close to 0 when f_s is near an integer multiple of f_L. Since almost all the eigenvalues of P_M have magnitude less than 1, the eigenvalues near 1 are the ones with the largest magnitude. The results of the numerical experiments are in agreement with our previous prediction in Section II-C.

TABLE II
The eigenvalue distribution of P_M

Index	#Nodes	$\lvert\lambda\rvert_{\max}$	# $\lvert\lambda - 1\rvert \leq 0.05$
case 1	3886	1.0000	4
case 2	9482	0.9864	1
case 3	16224	1.0000	18
case 4	3058	1.0000	48

B. Choice of initial guess

We demonstrate the effectiveness of our optimal initial guess recycling method. We test three different initial guess choices by performing GMRES method with no recycled subspace on case 4. The sweeping type is chosen as "LIN 201 0 f_L". The results are shown in Figure 1. "Zero" means setting the initial guess to be zero, while "Original" means taking the approximate solution of the previous system as the initial guess for the next system. Our method is referred to as "Proposed".

From Figure 1, the number of iterations required for "Original" and "Zero" is almost the same, though at some frequency points "Original" may require more steps to converge. Hence, it is shown that taking the solution of the previous system as the initial guess is not a good choice. However, with our proposed method, convergence iterations can be greatly reduced over a wide range of small-signal frequencies. When f_s is far from an integer multiple of f_L, the convergence of iterative methods is really good, and our proposed method can still save several iterations.

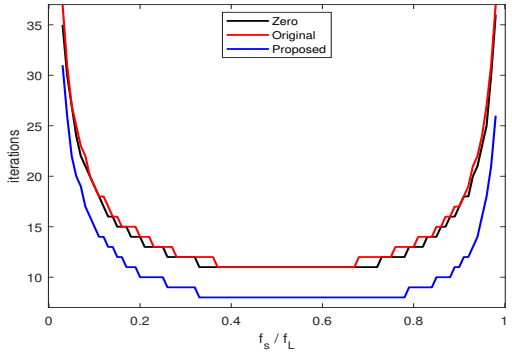


Fig. 1. Number of iterations for case 4 with different initial guess choices

C. Subspace truncation strategy

This section demonstrates the effectiveness of our Floquet-based subspace truncation strategy. We compare our Floquet-based truncation strategy with the deflated restarted (DR) method used in GCRO-DR [5], [9]. We test two subspace truncation methods in our framework of algorithm on case 4 for the sweeping type "DEC 20 10 10^7". The maximal dimension of recycled subspace is set to be 25. To better highlight the difference between two methods, the initial guess is set to be zero. The results are shown in Figure 2. "DR" means using deflated restarted method while "Floquet" is our Floquet-based subspace truncation method.

TABLE III
COMPARISON BETWEEN GCRO-DR AND OUR PROPOSED METHOD FOR "DEC 6 10 10^8" (TIME UNIT: SECOND).

Index	$m_{\max} = 20$, $j_{\max} = 40$					$m_{\max} = 25$, $j_{\max} = 30$				
	GCRO-DR		Proposed			GCRO-DR		Proposed		
	#mvm	Time	#mvm	Time	Speedup	#mvm	Time	#mvm	Time	Speedup
case 1	110	293.50	**79**	**180.98**	**1.62**	74	202.37	**57**	**133.70**	**1.51**
case 2	189	2029.30	**112**	**1026.15**	**1.98**	75	1073.05	**51**	**463.52**	**2.32**
case 3	223	1269.17	**103**	**565.55**	**2.24**	175	1027.06	**79**	**439.68**	**2.34**
case 4	2239	1579.53	**1700**	**1222.87**	**1.29**	2095	1480.24	**1537**	**1099.54**	**1.35**

TABLE IV
COMPARISON BETWEEN GCRO-DR AND OUR PROPOSED METHOD FOR "LIN 41 0 f_L" (TIME UNIT: SECOND).

Index	$m_{\max} = 20$, $j_{\max} = 40$					$m_{\max} = 25$, $j_{\max} = 30$				
	GCRO-DR		Proposed			GCRO-DR		Proposed		
	#mvm	Time	#mvm	Time	Speedup	#mvm	Time	#mvm	Time	Speedup
case 1	281	690.91	**145**	**321.70**	**2.15**	245	610.04	**106**	**240.86**	**2.53**
case 2	277	2874.33	**184**	**1729.40**	**1.66**	244	2536.48	**105**	**957.53**	**2.65**
case 3	322	1796.86	**197**	**1058.18**	**1.70**	279	1580.90	**153**	**817.34**	**1.93**
case 4	1220	845.26	**811**	**562.13**	**1.50**	1108	770.82	**792**	**553.93**	**1.39**

From Figure 2, a noticeable reduction in iterations is observed for our proposed method. When f_s is very small (near 10 Hz), the DR method requires significantly more iterations than our method. During the process of frequency sweep, DR may experience sudden increases of iterations, while this phenomenon does not occur in our method.

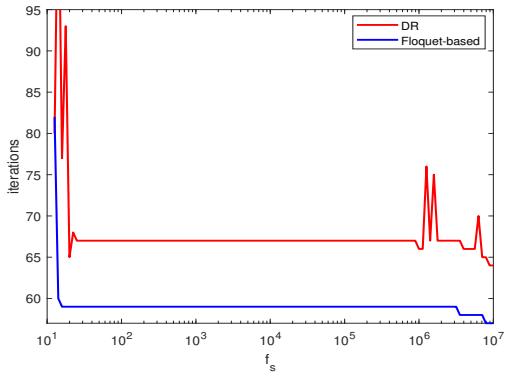


Fig. 2. Number of iterations for case 4 with different subspace truncation methods

D. Comparing proposed method with GCRO-DR

We compare our recycling subspace truncation method for periodic small-signal analysis with GCRO-DR [5]. Two types of frequency sweeps are tested: "DEC 6 10 10^8" and "LIN 41 0 f_L". For both frequency sweeps, the frequency points are arranged in ascending order. To better reflect the real simulation process, we choose $m_{\max}$ and $j_{\max}$ such that the memory consumption for storing the total subspace remains the same. The numerical results are shown in Table III and Table IV. "#mvm" means the total number of matrix-vector multiplications, which is exactly the number of iterations in the whole algorithm. "Speedup" is the ratio of the runtime.

From Table III and Table IV, it can be observed that for both GCRO-DR and our proposed method, $m_{\max} = 25$ is faster than $m_{\max} = 20$, which means appropriately increasing the maximum dimension of recycled subspace can reduce the total number of matrix-vector multiplications. Compared to GCRO-DR, our proposed method can significantly reduce the simulation time of both frequency sweeps. Our proposed method can achieve up to a $2.65\times$ speedup, and when the maximal dimension of recycled subspace increases, the speedup of our proposed method is better. Considering the magnitudes of speedup, the performance of our proposed method is very competent.

V. CONCLUSION

In this paper, we present a framework of recycling subspace truncation method for periodic small-signal analysis. For initial guess, we propose an efficient optimal initial guess recycling method, which can save many iterations with low computational cost. For subspace truncation, we observe that the subspace corresponding to the largest Floquet multipliers significantly affect the convergence of iterative methods. Based on the observation, we develop the Floquet-based subspace truncation strategy. The effectiveness of our proposed method is verified by the numerical experiments. We hope that our proposed method can help overcome the difficulty of memory shortage encountered in periodic small-signal analysis for large-scale circuit designs. Possible future improvement would be to find a better sweeping order of small-signal frequencies.

ACKNOWLEDGMENT

The authors sincerely acknowledge Primarius for their technical collaboration in this research.

REFERENCES

[1] R. Telichevesky, K. Kundert, and J. White, "Receiver characterization using periodic small-signal analysis," in *Proceedings of Custom Integrated Circuits Conference*, ser. CICC-96. IEEE, 1996.

[2] ——, "Efficient AC and noise analysis of two-tone RF circuits," in *33rd Design Automation Conference Proceedings, 1996*. ACM, 1996.

[3] L. Ouyang, C. Jin, and Q. Chen, "A fast recycling GMRES method with smart frequency sweeping for efficient periodic small-signal analysis," *IEEE Trans. Circuits Syst. II: Express Briefs*, 2024.

[4] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 7, pp. 856–869, 1986.

[5] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti, "Recycling Krylov subspaces for sequences of linear systems," *SIAM J. Sci. Comput.*, vol. 28, pp. 1651–1674, 2006.

[6] A. Demir, "Floquet theory and non-linear perturbation analysis for oscillators with differential-algebraic equations," *Int. J. Circuit Appl.*, vol. 28, no. 2, pp. 163–185, 2000.

[7] C.-W. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits Syst.*, vol. 22, pp. 504–509, 1975.

[8] M. Farkas, *Periodic Motions*. Berlin, Heidelberg, Germany: Springer-Verlag, 1994.

[9] R. B. Morgan, "GMRES with deflated restarting," *SIAM J. Sci. Comput.*, vol. 24, pp. 20–37, 2002.

[10] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*. Berlin, Heidelberg, Germany: Springer Science + Business Media, 1990.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins University Press, 2013.

[12] B. Kågström and D. Kressner, "Multishift variants of the qz algorithm with aggressive early deflation," *SIAM J. Matrix Anal. Appl.*, vol. 29, pp. 199–227, 2007.

[13] K. Braman, R. Byers, and R. Mathias, "The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance," *SIAM J. Matrix Anal. Appl.*, vol. 23, pp. 929–947, 2002.

[14] ——, "The multishift QR algorithm. Part II: Aggressive early deflation," *SIAM J. Matrix Anal. Appl.*, vol. 23, pp. 948–973, 2002.

[15] P. W. Tuinenga, *SPICE (3rd ed.): a guide to circuit simulation and analysis using PSpice*. USA: Prentice-Hall, Inc., 1995.

Generalizable Lithographic Hotspot Detection Using Asynchronous Meta-Learning with Only One Shot

Cong Jiang*, Yujia Wang*, Dan Feng*, Haoyu Yang† and Kang Liu*
*Huazhong University of Science and Technology, Wuhan, CN, †NVIDIA Corp., Austin, TX, USA
*{jiangconghust, yujiawang, dfeng, kangliu}@hust.edu.cn, †haoyuy@nvidia.com

***Abstract*—With integrated circuits shrinking in feature size, layout printability has become increasingly challenging, making lithographic hotspot detection ever-crucial in computer-aided design (CAD) flows. In recent years, numerous studies have explored deep learning to detect lithographic hotspots, offering promising results. However, neural networks can easily be biased and overfit when lacking sufficient training data, especially in the CAD domain. A generalizable DL-based hotspot detector should learn the genuine lithography principle and ensure consistent accuracy across layouts from various designs at the same technology node, regardless of their varying design styles. However, we find that existing convolutional neural network (CNN)-based hotspot detectors fail to generalize to different circuit layouts other than the design it has been trained for. To this end, we propose a few-shot learning-based framework for generalizable CNN-based hotspot detection. We develop a meta-learning scheme that asynchronously updates the CNN feature extraction and classification component to obtain a meta-initialized model that can quickly adapt to new designs using as few as one training layout clip. We propose a layout topology-based sampling strategy for few-shot adaptation to enhance generalization stability. Experimental results on ICCAD 2012 and 2019 datasets show that our framework enables superior generalization capabilities than prior arts on unseen new designs.**

***Index Terms*—Lithographic hotspot detection, generalizability, few-shot learning**

I. Introduction

With the shrinkage of feature sizes in advanced technology nodes, integrated circuit layouts are susceptible to printing defects, making lithographic hotspot detection a crucial part of the design-for-manufacturability (DFM) flow. Conventional lithography simulation achieves accurate hotspot detection at the expense of high computation costs. Pattern matching-based methods expedite the detection process by comparing layout patterns with existing hotspots but fail to identify hotspots unseen before. Recently, with advancements in deep learning (DL), researchers have proposed DL-based solutions that demonstrate promising results in hotspot detection.

Most existing DL-based works focus on improving hotspot detection accuracy. For example, Yang et al. [1] proposed a DCT-based layout feature representation for hotspot detection. Geng et al. [2] introduced attention modules in the DL model, achieving an end-to-end hotspot detection flow.

Kang Liu is partly supported by National Natural Science Foundation of China No. 62202190, Hubei Natural Science Foundation No. 2023AFB237, and the Knowledge Innovation Program of Wuhan-Shuguang. Kang Liu is the corresponding author.

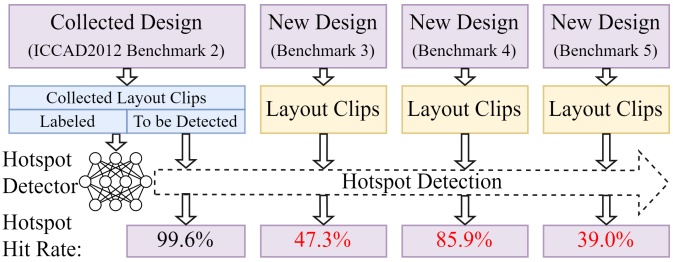


Fig. 1 Detection accuracy of SoTA attention-based CNN hotspot detector [2] when validated on various layout designs using the same 28 nm technology node. *Note that ICCAD 2012 Benchmark 1 is developed using a different 32 nm technology node and thus is not tested here.*

Jiang et al. [3] proposed a Bayesian optimization-based neural architecture search scheme to automatically design high-performance convolutional neural networks (CNNs). Shao et al. [4] integrated a learned lithography simulator with an object detection backbone to extract layout features, thereby improving detection performance. In addition to accuracy enhancements, other studies explore orthogonal directions in DL-based hotspot detection, including inference speed-up [5], [6], data scarcity [7], [8] and imbalance [9]–[11], robustness and security [12]–[14], and interpretability [15], [16]. Other works detect multiple hotspots on full layouts instead of partitioned layout clips [17], [18].

However, despite the verified efficiency of CNNs for hotspot detection as demonstrated by their accuracy when trained and validated using a certain circuit design, it has never received deliberation whether they have learned the genuine lithography principle rather than a biased design style pertaining to a specific design. Given that the CNNs are notorious for having overfitting issues and are easy towards biased learning [19], in this work, we seek to answer the following question:

Can DL-based lithographic hotspot detectors generalize to unseen new designs different from their trained design at the same technology node?

We use a state-of-the-art (SoTA) attention-based CNN hotspot detector from [2] as our motivating case study, and we examine its detection accuracy on various circuit designs from the ICCAD 2012 dataset that uses the same 28 nm technology node. Specifically, we use the training dataset from benchmark 2 for training and the rest of the layout clips from benchmark 2 to 5 for testing, as shown in Fig. 1. We witness a drastic

drop between detection accuracy on benchmark 2 and that on benchmark 3, 4, and 5, despite their shared technology node used in layout design, indicating severe generalization limitations of existing hotspot detection CNNs.

Several studies [20]–[23] have explored transfer learning or active learning with a certain budget for CNN-based hotspot detection when design environments and parameters change. However, they rely heavily on significant amounts of labeled data and are sensitive to the initial model, which can significantly affect the final performance. In the case of CAD, large amounts of quality labeled data are not always accessible.

To this end, we propose a training framework for generalizable CNN-based hotspot detection on unseen new designs using a minimum additional budget. Specifically, we propose an asynchronous meta-learning scheme that obtains a generalizable meta-initial model that can adapt to new designs using few-shot learning. Asynchronous updates are applied to CNN's feature extraction and classification component, alleviating the overfitting of specific feature representations of individual designs. We further develop a layout topology-based sampling strategy to select the most representative layout patterns in the new design for a few-shot adaptation of the first stage meta-initialized model. Our experimental results demonstrate the efficiency of our framework, showcasing both stable and high-performance hotspot detection across designs. Our main contributions are as follows:

- We investigate for the first time the generalizability of DL-based lithographic hotspot detectors on different circuit designs at the same technology node.
- An asynchronous meta-learning framework that trains a CNN hotspot detector that can generalize to unseen new designs using as few as one-shot adaptation.
- A layout topology-based sampling strategy for one-shot adaptation that enables the most stable generalization.
- Extensive experimental results on the advantages of our proposed training framework for generalizable DL-based hotspot detection compared to all baseline solutions.

II. Preliminaries and Problem Definition

A. Meta-learning

Meta-learning refers to a variety of techniques that learn knowledge with accumulated data of multiple tasks from prior experiences and then quickly adapt to new tasks with few data points. In [24], Bai et al. categorize existing studies in meta-learning into three types, including metric-based, memory-based, and learning-based methods. Among them, MAML [25] is a representative learning-based algorithm designed to learn a cross-task meta-initial parameter θ that enables rapid adaptation to new tasks.

In MAML, the dataset $\mathcal{D}$ comprising all tasks is first divided into two non-overlapping meta sets: D^{train} for meta-training, and $\mathcal{D}^{val}$ for validation. $\mathcal{D}^{train}$ is further partitioned into the *Support set* $\mathcal{S}^{train}$ and the *Query set* $\mathcal{Q}^{train}$ to facilitate the *Inner loop* and the *Outer loop* of meta-training. The inner loop refines the initial parameter θ using $\mathcal{S}_i^{train}$ for each task i, yielding task-specific intermediate parameters θ_i. The outer loop then refines the meta-initial parameter with $\mathcal{Q}^{train}$ to enhance cross-task generalization. After meta-training, adaptation and evaluation are conducted using $\mathcal{S}^{val}$ and $\mathcal{Q}^{val}$, respectively, sampled from $\mathcal{D}^{val}$.

B. Evaluation Metrics and Problem Formulation

We formulate lithographic hotspot detection as a binary classification problem. Given a set of labeled layout clips $\mathcal{D}^{train} = \{\mathcal{D}_1, \mathcal{D}_2 \cdots \mathcal{D}_N\} = \{(X_i,\ Y_i)\}_{i=1}^{N}$ from N circuit designs, our proposed framework aims to learn a model f with an initial parameter θ such that $Y = f_\theta(X)$ and can quickly adapt to a new design using only K sampled layout clips for training. We adopt the same metrics used in prior lithographic hotspot detection work as follows.

Definition 1 (Hotspot Hit Accuracy, *Acc*). The ratio of correctly classified hotspot clips among all the actual hotspots.

Definition 2 (False Alarm Rate, *FAR*). The ratio of clips misclassified as hotspots among all the actual non-hotspots.

Definition 3 (F1 Score, *F1*). The harmonic mean of Acc and Precision, where Precision denotes the ratio of correctly identified hotspots among the total number of predicted hotspots. F1 comprehensively reflects Acc and FAR performance.

$$F1 = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{1}$$

Problem 1 (Generalizable CNN-based Lithographic Hotspot Detection). Given a set of layout clips from collected designs that contain hotspots and non-hotspots, train an initial model that can generalize to both the collected designs and unseen new designs at the same technology node using K-shot (i.e., $K = 1$) adaptation that achieves the maximum Acc and F1 and the minimum FAR.

III. Proposed Method

A. Overview

In this paper, we aim to enhance the generalizability of CNN-based lithographic hotspot detection across various unseen new designs at the same technology node as the trained design, despite their different design styles. Specifically, we propose an asynchronous meta-learning-based framework to train an initial model that can quickly adapt to new designs with a minimum budget, i.e., using one-shot learning with a sampling strategy.

Fig. 2 provides an overview of our proposed framework. Our generalizable hotspot detection process begins with a collection of layout clips and their corresponding labels derived from various designs to construct the meta-training dataset $\mathcal{D}^{train}$. A randomly initialized hotspot detection model is trained on $\mathcal{D}^{train}$ for multiple epochs using the proposed asynchronous meta-learning algorithm to obtain the meta-initial parameter, θ. Following this training phase, the meta-initialized model is adapted to new designs using a sampled set of K layout clips per category from the new design, which is selected by our proposed layout topology-based sampling method.

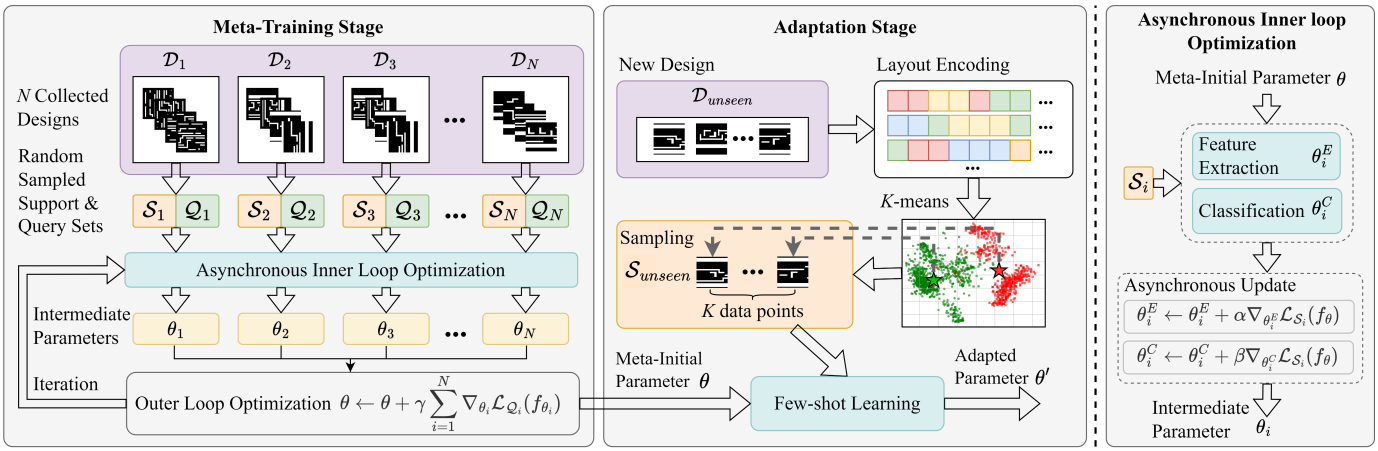


Fig. 2 Illustration of our proposed asynchronous meta-learning framework for generalizable DL-based lithographic hotspot detection using few-shot adaptation.

B. Asynchronous Meta-training

A meta-learning framework incorporates inner loop optimization and outer loop optimization, which update the network parameters based on layout clips from individual designs and all collected designs, respectively. However, the subtle and complex lithography principle is hard to learn, whereas, on the contrary, the extreme bias of design styles of the small support sets (i.e., 2 samples) from a specific design can easily overtake the CNN's learning directions. To suppress overfitting on the design styles of individual layout circuits and gradually converge to the common lithography rules shared among different designs, we constrain the learning capacity of the CNN's feature extraction component when learning each specific design in the inner loop.

We propose an asynchronous meta-training method such that different learning rates are applied to the feature extraction and classification components within each inner loop for each design. Specifically, the feature extraction component is updated with a smaller learning rate, α, to prevent overfitting on individual design features, and the classification component is updated with a relatively larger learning rate, β, to facilitate discrimination capabilities. During the outer loop, when updating with a larger batch size of the query set, both feature extraction and classification components are updated using the same learning rate, γ, to enhance CNN's overall generalization across designs.

1) Inner loop optimization: Given a set of collected layout clips $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2 \cdots \mathcal{D}_N\}$ from N designs, support sets $\mathcal{S}_i$ and query sets $\mathcal{Q}_i$ are randomly sampled for each design $\mathcal{D}_i$. The intermediate parameters θ_i are obtained through asynchronous inner loop optimization, formulated as follows:

$$\theta_i^E \leftarrow \theta_i^E + \alpha \nabla_{\theta_i^E} \mathcal{L}_{\mathcal{S}_i} \tag{2}$$

$$\theta_i^C \leftarrow \theta_i^C + \beta \nabla_{\theta_i^C} \mathcal{L}_{\mathcal{S}_i} \tag{3}$$

Equation 2 and Equation 3 represent the asynchronous update for CNN's feature extraction component θ_i^E and classification component θ_i^C, using different learning rates α and β. We adopt the same loss function $\mathcal{L}$ as used in [2] in our training framework, where a prediction and metric loss combine to provide an enhanced hotspot detection accuracy, as expressed in Equation 4, Equation 5, and Equation 6. Here, the metric loss optimizes the distances between the embedding vectors $f_{\theta^E}(x)$ of layout clip x and other layouts from the same (x^+) and different (x^-) categories in the batch. M is a positive constant.

$$\mathcal{L} = \mathcal{L}^{pred} + \mathcal{L}^{metric} \tag{4}$$

$$\mathcal{L}^{pred} = -\sum ylog(f_\theta(x)) + (y-1)log(1 - f_\theta(x)) \tag{5}$$

$$\mathcal{L}^{metric} = \sum max(0, M + \|f_{\theta^E}(x) - f_{\theta^E}(x^+)\|_2^2 - \|f_{\theta^E}(x) - f_{\theta^E}(x^-)\|_2^2) \tag{6}$$

2) Outer loop optimization: After the inner loop optimization, we obtain a set of intermediate parameters, denoted as $\{\theta_1,\ \theta_2 \cdots \theta_N\}$, for each design. Subsequently, gradients of each θ_i are computed based on their corresponding query sets $\mathcal{Q}_i$. These gradients are then aggregated to update the meta-initial parameter θ during the outer loop optimization using learning rate γ, as shown in Equation 7:

$$\theta \leftarrow \theta + \gamma \sum_{i=1}^{N} \nabla_{\theta_i} \mathcal{L}_{\mathcal{Q}_i}(f_{\theta_i}) \tag{7}$$

Note that in the outer loop optimization, gradients are computed based on a set of design-specific parameters θ_i obtained from the inner loop yet applied to the meta-initial parameter θ. This meta-training procedure ensures the CNN converges toward a common initial parameter, facilitating generalization to new designs that observe a universal lithography principle at the same technology node.

C. Layout Topology-based Few-shot Sampling of New Designs

After meta-training, the CNN takes few-shot adaptive training to generalize to unseen new designs. However, the generalization performance highly depends on the selection of

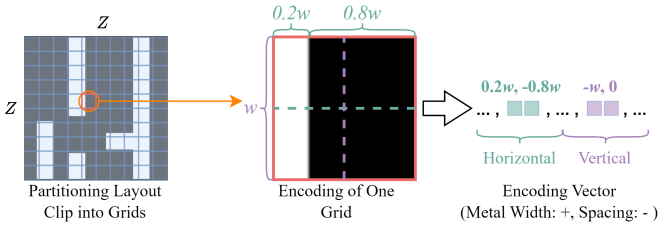


Fig. 3 Illustration of our layout topology-based encoding. A layout clip is partitioned into $Z \times Z$ grids of size $w \times w$. Along horizontal and vertical directions, the lengths of metal widths and spacings are encoded with + and - values, respectively.

layout clips used for adaptation. Training on outlier data can obviously result in inaccurate predictions for the new design. To address this issue, we propose a layout topology-based encoding strategy that assists in sampling the most representative layout clips of the new design. Our layout topology-based encoding represents each layout clip as a fixed-length vector, capturing its structural characteristics, based on which we employ K-means clustering (2-way for hotspots and non-hotspots) that selects the cluster heads and their nearest neighbors as the most representative hotspot and non-hotspot clips in the new design for few-shot adaptation.

1) Layout topology-based encoding: Various layout encoding methods have been proposed for hotspot detection, such as concentric circle-based [26], graph-based [6], or density-based [27] representation. However, they either produce variable-length feature vectors or miss out on critical layout information (i.e., using only density information). In the context of new design clips with no accessible ground-truth labels, variable-length or incomplete layout features pose significant challenges for unsupervised layout analysis.

To this end, we propose a layout topology-based encoding method that employs layout topology features for fixed-length feature representation. As illustrated in Fig. 3, for a layout clip of size $R \times R$ with half-pitch p, we partition each clip into the same number of $Z \times Z$ grids, each sizing $w \times w$. Here Z is a user-defined parameter satisfying the constraint $Z > \frac{R}{p}$, such that each grid contains at most one metal wire. We next use horizontal and vertical scanning lines to traverse each grid of the layout clip and capture the geometric information of the metal and spacing within each grid. Specifically, in each grid, a central horizontal line scans from left to right, and a central vertical line scans from top to bottom. Metal and spacing dimensions of the grid are used and encoded as a one-dimensional fixed-length vector, which stacks over all the grids of the layout clip. This scanning approach and encoding ensure full coverage of the layout's topological features. To differentiate metal and spacing characteristics, we assign metal widths with positive values and spacing widths with negative values. Our fixed-length layout topology-based encoding enables next-step layout comparisons and analyses.

2) K-means clustering-based few-shot selection: Since the layout clips of a new design are unlabeled, we employ a 2-way K-means clustering to partition these new clips into two groups in an unsupervised fashion, where the most representative clips within each group are selected for a few-shot adaptation. Specifically, our proposed layout topology-based encoding for each new clip is used as input to the K-means clustering algorithm. We select the cluster heads and their nearest $K-1$ neighbors as the K most representative clips for each layout class. We apply lithography simulation on these sampled layout clips for ground-truth labeling and train them in a one-pass few-shot adaptation for CNN generalization.

IV. Experimental Results

A. Experimental Setup

TABLE 1: Layout clip numbers of each benchmark in ICCAD 2012 & 2019 that use 28 nm technology node.

Dataset		ICCAD 2012				ICCAD 2019			
		B2	B3	B4	B5	B2	B3	B4	B5
Train	HS	174	909	95	26	38	370	46	13
	NHS	5285	4643	4452	2716	5365	5166	4498	2729
Test	HS	498	1808	177	41	278	622	81	20
	NHS	4146	3541	3386	2111	4362	4654	3473	2132

1) Datasets: We evaluate the effectiveness of our training framework for generalizable CNN-based hotspot detection on two layout datasets: ICCAD 2012 [28] and ICCAD 2019 [7]. In each dataset, four 28 nm benchmarks (B2 to B5) are employed to train and assess the CNN's generalizability to new designs developed at the same technology node (Benchmark 1 uses a 32 nm technology node.) Note that layouts in ICCAD 2012 and 2019 are labeled using different lithography systems. Thus, we do not evaluate CNN's detection generalizability across different datasets. We show in Table 1 the number of hotspot (HS) and non-hotspot (NHS) clips in ICCAD 2012 and 2019 in separate training and testing sets. We use the Test-I dataset in ICCAD 2019 for evaluation and exclude Test-II, which is manually augmented based on Test-I.

In our experiments, we train our CNN hotspot detector on two collected designs and evaluate its generalizability on two new designs. We train on layout collection of the training sets of either (B2 and B4) or (B3 and B5) and evaluate the test set of the rest of the two designs.

2) Baselines for Comparison: We compare our proposed framework against four baselines: (1) retraining from scratch using layout clips from new designs, (2) transfer learning the well-trained model on collected designs to new designs, (3) the ADAPT framework proposed by [23] that employs unsupervised Bayesian-based accuracy estimation for initial model selection and active learning with balanced data selection, and (4) meta-initial training as proposed in MAML [25] along with few-shot adaptation. We additionally evaluate our generalization framework that applies synchronous updates for CNN feature extraction and classification components as in standard meta-initial training, denoted as "Ours-" in Table 2.

We randomly sampled K layout clips per category, i.e., hotspot and non-hotspot, for our four baselines in their few-shot adaptation to the base model. Our training framework and "Ours-" use specially designed layout topology-based sampling strategy for few-shot adaptation. In consistency with

TABLE 2: Accuracy comparison across various frameworks incorporating initial training on collected designs, followed by one-pass few-shot adaptation and validation on new designs. In a few-shot adaptation, 1 or 5 layout clips are selected per category from new designs, either randomly (RT, TL, ADP, MAML) or sampled (Ours-, Ours). RT: retraining from scratch, TL: transfer learning, ADP: ADAPT [23]. MAML [25] and ours employ meta-learning for initial training. Ours-: our framework with synchronous meta-learning. Red and blue denote the best and second-best results, respectively.

Old Design	New Deisgn	Method	ICCAD 2012						ICCAD 2019					
			1-shot			5-shot			1-shot			5-shot		
			Acc	FAR	F1	Acc	FAR	F1	Acc	FAR	F1	Acc	FAR	F1
B2 & 4	B3	RT	56.7	52.4	43.7	80.0	79.0	47.8	60.0	59.9	19.7	61.6	62.9	19.5
		TL	58.4	2.3	71.7	66.7	4.0	76.4	82.0	31.3	39.4	83.9	26.9	43.6
		ADP	97.1	27.7	77.2	96.2	24.1	79.1	32.3	9.6	31.7	39.6	11.2	35.5
		MAML	83.9	22.3	73.7	30.1	5.3	42.9	93.1	68.6	26.4	94.5	61.6	28.8
		Ours-	99.7	47.5	65.9	99.8	57.6	63.9	87.3	49.4	31.4	100.0	94.9	22.0
		Ours	95.5	2.1	95.7	96.5	2.8	95.6	99.6	31.2	76.6	99.7	0.3	98.7
	B5	RT	100.0	98.3	3.8	80.0	79.3	3.8	100.0	99.0	1.9	80.0	79.2	1.9
		TL	60.0	0.5	63.9	75.6	3.3	44.1	90.0	13.0	11.4	90.0	12.6	11.8
		ADP	95.1	14.2	20.6	92.2	11.2	24.0	30.0	2.9	13.8	45.0	4.2	15.2
		MAML	75.7	15.0	16.0	73.7	8.0	25.1	100.0	63.5	2.9	89.2	32.9	4.8
		Ours-	100.0	41.4	6.2	100.0	68.4	5.4	100.0	59.2	3.1	100.0	93.1	2.0
		Ours	97.5	1.5	70.9	97.5	1.6	69.6	100.0	7.0	21.3	100.0	3.3	35.7
B3 & 5	B2	RT	80.0	77.4	19.4	20.0	20.0	14.0	80.0	80.0	11.2	60.1	59.9	10.9
		TL	93.7	3.4	84.6	96.2	7.0	75.7	75.8	11.8	42.0	81.3	11.0	46.0
		ADP	94.2	41.1	35.1	90.7	35.7	37.2	70.1	13.1	37.4	65.2	12.6	35.9
		MAML	78.3	29.6	36.9	44.1	8.8	40.6	55.1	16.0	27.1	63.4	29.3	20.3
		Ours-	99.8	58.5	29.1	100.0	88.7	21.3	95.0	31.1	27.8	67.0	28.7	21.7
		Ours	99.4	4.1	75.3	99.4	2.0	91.9	97.5	7.4	62.2	97.4	6.8	64.1
	B4	RT	63.7	64.6	9.1	40.0	40.0	8.8	60.0	60.1	4.4	60.0	54.0	4.8
		TL	75.5	1.1	76.9	84.6	3.8	65.7	64.0	7.2	27.2	79.3	7.0	33.1
		ADP	67.8	23.9	21.7	70.7	26.4	20.9	60.5	7.1	25.9	63.0	7.6	25.8
		MAML	77.7	13.8	35.2	85.1	7.1	53.1	63.4	16.6	14.5	77.0	26.2	11.9
		Ours-	99.4	32.2	24.4	65.3	4.5	52.0	88.9	20.5	16.7	98.9	29.6	13.5
		Ours	96.6	6.6	59.7	95.4	4.7	66.7	95.0	4.0	51.7	97.6	4.9	48.0

common practices in few-shot learning studies, we evaluate $K = \{1, 5\}$ for adaptation. Notably, due to the triplet loss [29] employed in the base model, which optimizes the distances between samples of the same category, the input must include at least two shots per category. Therefore, for the 1-shot adaptation experiment, we use a single layout clip per category and duplicate it to meet this requirement.

3) Hyperparameters: In our experiments, we use SoTA CNN architecture for hotspot detection as proposed in [2] as our base model architecture which incorporates attention modules for enhanced detection accuracy. In our experiments, we set the number of meta-initial training steps to 10,000. Within each training step, the inner loop and outer loop are trained for 1 and 10 iterations, respectively, using randomly sampled layouts with batch sizes of 2 and 8.

B. Generalization Evaluation on Unseen New Designs

We show in Table 2 the generalization performance of our proposed training framework for CNN-based hotspot detection compared against four baselines outlined in Section IV-A2. All experimental results are averaged over 5 runs due to sampling randomness for few-shot adaptation. Our framework demonstrates superior performance across most benchmarks, outperforming baselines in terms of Acc, FAR, and F1 scores. Specifically, we achieve top-1 Acc results in 7 out of the 16 experimental groups and top-2 performance in 15 out of the 16. We note that for those methods surpassing our framework in Acc, they mostly exhibit excessively high FAR ($\geq 27\%$), compromising their practical applicability. In contrast, our framework maintains a significantly lower FAR ($\leq 6.6\%$) while achieving comparable Acc ($\geq 95.5\%$) in these groups. F1 scores provide comprehensive evaluation of CNN's detection performance on both hotspots and non-hotspots. Our framework achieves top-1 F1 scores in 14 out of the 16 experimental groups and second place in the remaining two groups. This consistency underscores our framework's superior generalizability, demonstrating its effectiveness in adapting to previously unseen designs while maintaining exceptional Acc and FAR.

We additionally evaluate the generalization performance of our framework using asynchronous or synchronous meta-learning (denoted as "Ours-") for meta-initial training. As shown in Table 2, our generalization framework using synchronous meta-learning exhibits either significantly high FAR ($\geq 20.5\%$ in 15 groups) or less accurate Acc (65.3% in the remaining group), indicating its limited capability to generalize to new designs. In contrast, our asynchronous meta-learning method achieves an Acc of at least 95% and a FAR of $\leq 7.5\%$ in 15 out of 16 cases, demonstrating a substantial improvement in generalizability.

C. Generalization Stability Using Different Sampling Methods

As shown by our experimental results in Table 3, the selection of layout clips for few-shot adaptation plays a pivotal role in CNN's generalization to unseen new designs. We compare the generalization performance of our asynchronous meta-

TABLE 3: Accuracy of various data sampling methods for one-pass few-shot adaptation in generalizing CNN-based lithographic hotspot detection. Acc, FAR, and F1 are averaged over 5 runs for random-based, KM-L, and our sampling methods due to sampling randomness. Red and blue indicate the best-performing means and variances, respectively. Rand: Randomness, Uncert: Probability uncertainty of the meta-initial model, KM-L: layout image-based K-means clustering.

Old Design	New Design	Method	ICCAD 2012						ICCAD 2019					
			1-shot			5-shot			1-shot			5-shot		
			Acc	FAR	F1	Acc	FAR	F1	Acc	FAR	F1	Acc	FAR	F1
B2 & 4	B3	Rand	92.1 ± 1.29	2.1 ± 0.99	93.8 ± 0.84	95.6 ± 3.09	3.4 ± 2.69	94.5 ± 1.31	92.1 ± 4.53	29.8 ± 1.19	73.0 ± 3.00	69.5 ± 31.78	21.7 ± 16.12	41.6 ± 6.37
		Uncert	92.1 ± /	1.2 ± /	94.8 ± /	94.9 ± /	2.1 ± /	95.4 ± /	94.6 ± /	27.5 ± /	69.0 ± /	97.1 ± /	26.2 ± /	49.4 ± /
		KM-L	94.7 ± 0.03	2.0 ± 0.01	95.4 ± 0.02	96.5 ± 0.13	2.6 ± 0.03	95.7 ± 0.08	99.3 ± 0.08	29.9 ± 0.05	76.2 ± 0.14	99.6 ± 0.00	0.4 ± 0.00	98.5 ± 0.00
		Ours	95.5 ± 0.15	2.1 ± 0.04	95.7 ± 0.08	96.5 ± 0.10	2.8 ± 0.05	95.6 ± 0.08	99.6 ± 0.00	31.2 ± 0.06	76.6 ± 0.14	99.7 ± 0.08	0.3 ± 0.08	98.7 ± 0.33
	B5	Rand	96.0 ± 0.55	1.5 ± 0.47	70.3 ± 6.24	96.6 ± 2.27	2.7 ± 2.72	64.4 ± 21.04	98.0 ± 2.74	7.7 ± 2.92	20.8 ± 6.50	91.9 ± 7.60	4.5 ± 3.23	33.2 ± 14.28
		Uncert	95.7 ± /	0.5 ± /	86.0 ± /	96.7 ± /	0.3 ± /	91.5 ± /	100.0 ± /	8.2 ± /	18.6 ± /	94.9 ± /	12.9 ± /	12.7 ± /
		KM-L	97.5 ± 0.16	1.5 ± 0.03	70.9 ± 0.34	97.5 ± 0.16	3.6 ± 0.03	51.0 ± 0.32	100.0 ± 0.00	8.2 ± 0.07	18.5 ± 0.13	100.0 ± 0.00	12.9 ± 0.12	12.7 ± 0.10
		Ours	97.5 ± 0.13	1.5 ± 0.03	70.9 ± 0.33	97.5 ± 0.16	1.6 ± 0.02	69.6 ± 0.20	100.0 ± 0.00	7.0 ± 0.00	21.3 ± 0.00	100.0 ± 0.00	3.3 ± 0.00	35.7 ± 0.00
B3 & 5	B2	Rand	99.4 ± 0.00	4.8 ± 1.04	71.5 ± 7.54	99.2 ± 0.22	2.5 ± 0.75	90.1 ± 2.64	83.1 ± 19.66	5.6 ± 2.46	60.9 ± 5.65	98.4 ± 3.42	15.2 ± 10.48	50.0 ± 16.40
		Uncert	99.4 ± /	10.7 ± /	65.4 ± /	99.2 ± /	6.4 ± /	78.4 ± /	74.0 ± /	6.3 ± /	54.3 ± /	96.6 ± /	5.0 ± /	70.2 ± /
		KM-L	99.4 ± 0.00	4.5 ± 0.02	74.5 ± 0.21	99.4 ± 0.00	2.5 ± 0.01	90.2 ± 0.05	94.7 ± 0.43	7.1 ± 0.03	61.8 ± 0.19	98.9 ± 0.13	10.2 ± 0.05	55.1 ± 0.11
		Ours	99.4 ± 0.00	4.1 ± 0.01	75.3 ± 0.03	99.4 ± 0.00	2.0 ± 0.00	91.9 ± 0.00	97.5 ± 0.43	7.4 ± 0.03	62.2 ± 0.20	97.4 ± 0.20	6.8 ± 0.01	64.1 ± 0.10
	B4	Rand	96.3 ± 0.37	6.8 ± 0.42	58.2 ± 1.47	95.4 ± 0.21	5.3 ± 0.54	64.5 ± 2.32	78.1 ± 15.87	1.9 ± 0.78	60.2 ± 3.69	97.3 ± 5.24	7.4 ± 4.30	42.3 ± 14.96
		Uncert	95.9 ± /	25.4 ± /	28.1 ± /	95.4 ± /	27.1 ± /	26.7 ± /	82.3 ± /	1.7 ± /	64.3 ± /	67.7 ± /	1.1 ± /	63.0 ± /
		KM-L	96.6 ± 0.00	7.8 ± 0.04	55.9 ± 0.12	95.4 ± 0.13	4.8 ± 0.03	66.2 ± 0.18	93.7 ± 0.40	4.3 ± 0.04	49.7 ± 0.14	96.2 ± 0.26	3.7 ± 0.07	54.0 ± 0.49
		Ours	96.6 ± 0.00	6.6 ± 0.04	59.7 ± 0.15	95.4 ± 0.00	4.7 ± 0.03	66.7 ± 0.12	95.0 ± 0.16	4.0 ± 0.03	51.7 ± 0.26	97.6 ± 0.16	4.9 ± 0.09	48.0 ± 0.43

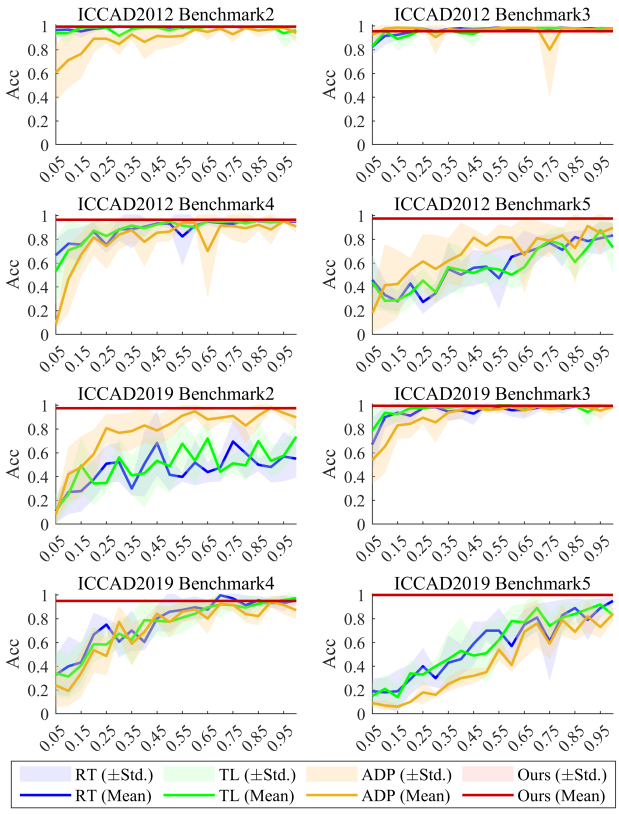


Fig. 4 Comparison of our generalizable framework using one-shot learning with baseline methods using varying data proportions of the new design (x-axis) for one-pass adaptation. The Std. of our framework is ≤ 0.0043.

learning-based framework using different sampling strategies of new clips, including random sampling (Rand), probability (of the meta-initial model)-based sampling (Uncert), and K-means clustering-based sampling based on either the original layout images (KM-L) or our proposed layout topology-based feature encoding (Ours). All experimental results are averaged over 5 runs for few-shot adaptation and evaluate both the accuracy and stability of generalization on new designs.

Our experimental results reveal that the two K-means-based sampling methods consistently outperform others in Acc and demonstrate superior performance in most evaluations of FAR and F1 scores. Compared to layout image-based sampling, our proposed layout topology-based encoding achieves consistently higher Acc and, in most cases, maintains a competitively extreme-low variance (≤ 0.0043). These findings highlight that our sampling method not only achieves superior generalization performance but also maintains stable results in few-shot sampling and adaptation.

D. Generalization vs. Data Efficiency for Adaptation

We evaluate the data efficiency of our training framework for generalizable CNN-based hotspot detection compared with other baseline solutions using varying numbers of layout clips for adaptation, including retraining (RT), transfer learning (TL), and ADAPT (ADP). These results reflect the amount of new layout clips necessary for the baseline models to achieve comparable generalization accuracy to our framework which uses only one-shot adaptation. As presented in Fig. 4, we report the mean (Mean) and standard deviation (Std.) of the generalization performance of 5 runs using randomly sampled data for adaptation. Our results demonstrate that the baseline models require substantially more layout clips and potentially additional training epochs to reach comparable generalization accuracy of ours. In contrast, our framework achieves the best generalization accuracy and stability using only one-shot adaptation, highlighting its superior generalizability in adapting to new designs.

V. Conclusion

In this work, we propose an asynchronous meta-learning-based framework for generalizable DL-based lithographic hotspot detection on unseen new designs using few-shot adaptation. Our proposed layout topology-based sampling strategy for few-shot adaptation enhances training stability for generalization. Our framework enables superior generalization capabilities than prior solutions on unseen new layout designs within the same technology node.

REFERENCES

[1] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline FY Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(6):1175–1187, 2019.

[2] Hao Geng, Haoyu Yang, Lu Zhang, Fan Yang, Xuan Zeng, and Bei Yu. Hotspot detection via attention-based deep layout metric learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2685–2698, 2022.

[3] Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via neural architecture search. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(6):1–16, 2022.

[4] Hao-Chiang Shao, Guan-Yu Chen, Yu-Hsien Lin, Chia-Wen Lin, Shao-Yun Fang, Pin-Yian Tsai, and Yan-Hsiu Liu. Lithohod: A litho simulator-powered framework for ic layout hotspot detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[5] FanYang YiyangJiang, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via binarized residual neural network ensemble. *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, 40(7), 2021.

[6] Shuyuan Sun, Yiyang Jiang, Fan Yang, Bei Yu, and Xuan Zeng. Efficient hotspot detection via graph neural network. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1233–1238. IEEE, 2022.

[7] Gaurav Rajavendra Reddy, Kareem Madkour, and Yiorgos Makris. Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[8] Liangjian Wen, Yi Zhu, Lei Ye, Guojin Chen, Bei Yu, Jianzhuang Liu, and Chunjing Xu. Layoutransformer: Generating layout patterns with transformer via sequential pattern modeling. In *Proceedings of the 41st IEEE/ACM international conference on computer-aided design*, pages 1–9. IEEE, 2022.

[9] Xu He, Yu Deng, Shizhe Zhou, Rui Li, Yao Wang, and Yang Guo. Lithography hotspot detection with fft-based feature extraction and imbalanced learning rate. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25(2):1–21, 2019.

[10] Wei Ye, Yibo Lin, Meng Li, Qiang Liu, and David Z Pan. Lithoroc: Lithography hotspot detection with explicit roc optimization. In *2019 24th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 1–7. IEEE, 2019.

[11] Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, and Bei Yu. Imbalance aware lithography hotspot detection: a deep learning approach. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 16(3):033504–033504, 2017.

[12] Jingyu Pan, Chen-Chia Chang, Zhiyao Xie, Jiang Hu, and Yiran Chen. Robustify ml-based lithography hotspot detectors. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–7. IEEE, 2022.

[13] Kang Liu, Benjamin Tan, Gaurav Rajavendra Reddy, Siddharth Garg, Yiorgos Makris, and Ramesh Karri. Bias busters: Robustifying dl-based lithographic hotspot detectors against backdooring attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(10):2077–2089, 2021.

[14] Kang Liu, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Training data poisoning in ml-cad: Backdooring dl-based lithographic hotspot detectors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(6):1244–1257, 2021.

[15] Haoyang Sun, Cong Jiang, Xun Ye, Dan Feng, Benjamin Tan, Yuzhe Ma, and Kang Liu. Interpretable cnn-based lithographic hotspot detection through error marker learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[16] Tao Zhang, Haoyu Yang, Kang Liu, and Zhiyao Xie. Apple: An explainer of ml predictions on circuit layout at the circuit-element level. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 374–379. IEEE, 2024.

[17] Ran Chen, Wei Zhong, Haoyu Yang, Hao Geng, Fan Yang, Xuan Zeng, and Bei Yu. Faster region-based hotspot detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(3):669–680, 2020.

[18] Tianyang Gai, Tong Qu, Shuhan Wang, Xiaojing Su, Renren Xu, Yun Wang, Jing Xue, Yajuan Su, Yayi Wei, and Tianchun Ye. Flexible hotspot detection based on fully convolutional network with transfer learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4626–4638, 2021.

[19] Quanshi Zhang, Wenguan Wang, and Song-Chun Zhu. Examining cnn representations with respect to dataset bias. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[20] Wei Ye, Mohamed Baker Alawieh, Meng Li, Yibo Lin, and David Z Pan. Litho-gpa: Gaussian process assurance for lithography hotspot detection. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 54–59. IEEE, 2019.

[21] Yifeng Xiao, Miaodi Su, Haoyu Yang, Jianli Chen, Jun Yu, and Bei Yu. Low-cost lithography hotspot detection with active entropy sampling and model calibration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 907–912. IEEE, 2021.

[22] Haoyu Yang, Shuhe Li, Cyrus Tabery, Bingqing Lin, and Bei Yu. Bridging the gap between layout pattern sampling and hotspot detection via batch active learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1464–1475, 2020.

[23] Mohamed Baker Alawieh and David Z Pan. Adapt: An adaptive machine learning framework with application to lithography hotspot detection. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 2021.

[24] Hassan Gharoun, Fereshteh Momenifar, Fang Chen, and Amir Gandomi. Meta-learning approaches for few-shot learning: A survey of recent advances. *ACM Computing Surveys*, 56(12):1–41, 2024.

[25] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[26] Allan Gu and Avideh Zakhor. Optical proximity correction with linear regression. *IEEE Transactions on Semiconductor Manufacturing*, 21(2):263–271, 2008.

[27] Tetsuaki Matsunawa, Jhih-Rong Gao, Bei Yu, and David Z Pan. A new lithography hotspot detection framework based on adaboost classifier and simplified feature extraction. In *Design-Process-Technology Co-optimization for Manufacturability IX*, volume 9427, pages 201–211. SPIE, 2015.

[28] J Andres Torres. Iccad-2012 cad contest in fuzzy pattern matching for physical verification and benchmark suite. In *Proceedings of the International Conference on Computer-Aided Design*, pages 349–350, 2012.

[29] De Cheng, Yihong Gong, Sanping Zhou, Jinjun Wang, and Nanning Zheng. Person re-identification by multi-channel parts-based cnn with improved triplet loss function. In *Proceedings of the iEEE conference on computer vision and pattern recognition*, pages 1335–1344, 2016.

An Energy-Efficient Kalman Filter Architecture with Tunable Accuracy for Brain-Computer Interfaces

Guy Eichler
Dept. of Computer Science
Columbia University
New York, New York, USA
guyeichler@cs.columbia.edu

Joseph Zuckerman
Dept. of Computer Science
Columbia University
New York, New York, USA
jzuck@cs.columbia.edu

Luca P. Carloni
Dept. of Computer Science
Columbia University
New York, New York, USA
luca@cs.columbia.edu

***Abstract*—Kalman Filter (KF) is the most prominent algorithm to predict motion from measurements of brain activity. However, little effort has been made to specialize KF hardware for the unique requirements of embedded brain-computer interfaces (BCIs). For this reason, we present the first configurable KF hardware architecture that enables fine-grained tuning of latency and accuracy, thereby facilitating specialization for neural data processing in BCI applications and supporting design-space exploration. Based on our architecture, we design KF hardware accelerators and integrate them into a heterogeneous system-on-chip (SoC). Through FPGA-based experiments, we demonstrate an energy-efficiency improvement of 15.3x and 10^3x better accuracy compared to state-of-the-art implementations.**

***Index Terms*—Kalman Filter (KF), System-on-Chip (SoC), Accelerator Design, Brain-Computer Interface (BCI), FPGA.**

I. Introduction

Real-world brain-computer interfaces (BCIs) establish a connection between the brain and the outside world to improve quality of life [1]–[3]. They rely on high-resolution neural interfaces, high-throughput data acquisition, and real-time computation [4]–[8]. While resolution and throughput continue to grow exponentially with the increasing number of electrodes in neural interfaces [2], [5], [6], [9], [10], achieving a wireless, implant-based BCI system that supports mobility, real-time processing, and low power consumption remains a challenge.

In this work, we focus on the Kalman Filter (KF), the most widely used online algorithm for motion prediction in the BCI field [11]–[16]. The KF estimates movement from neural signals and outputs real-time predictions of the position and velocity of specific body parts. These predictions are used to bypass the central nervous system and actuate a prosthesis or move a body part [17], [18]. However, BCI research has primarily focused on improving KF accuracy using machine learning, with less attention to its integration in real-world BCI systems [13], [14], [19]–[21]. Moreover, existing hardware implementations of the KF are not tailored for BCI use [22]–[25]. They rely on general-purpose techniques that are not optimized for processing diverse, high-dimensional neural data [2].

For this reason, we introduce KalmMind, a configurable architecture for agile development and design-space exploration of KF hardware accelerators tailored to BCI applications.

Fig. 1 provides a high-level overview of a KalmMind accelerator in a BCI system. The KF computation is offloaded from the implanted chip to an external device, referred to as a *relay station*, via wireless transmission of neural data [5], [26]. The relay station must be: *(1)* mobile, to support user movement; *(2)* low power, to operate within body-area networks (BAN) [27]; and *(3)* real-time, to maintain prediction relevance [28].

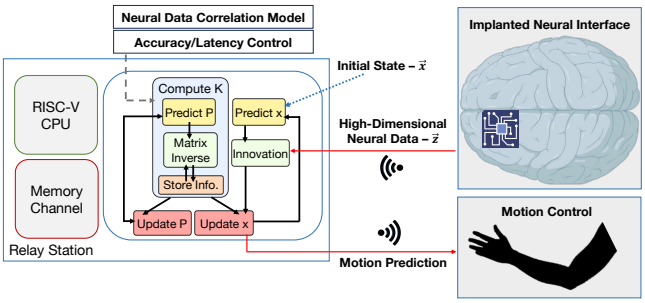


Fig. 1: KalmMind-based hardware accelerator in a BCI system.

To this end, KalmMind optimizes the main bottleneck of the KF: matrix inverse computation. It introduces a technique based on the Newton-Raphson method [29], [30] to control computational intensity by leveraging spatio-temporal correlations in neural activity across consecutive KF iterations [2], [3].

We demonstrate how KalmMind offers a flexible hardware architecture for thorough design-space exploration (DSE), aiming to minimize latency and maximize accuracy across neural datasets from different sources. We integrate additional approximation techniques inside the KF and implement a large group of configurable KF hardware accelerators [22], [31], [32].

We evaluate these accelerators by integrating them into a complete SoC that includes a RISC-V CVA6 processor [33] and prototyping the SoC on FPGA.

Our experiments show that KalmMind accelerators meet the real-time and low-power requirements of BCI systems. With its high design flexibility and fine-grained control over latency and accuracy, KalmMind opens up new opportunities for real-time KF predictions in embedded BCI systems.

We break down our main contributions as follows:

1) A reorganization of the KF algorithm that enables a modular and configurable hardware architecture.
2) A specialized approximation technique designed for use within a KF hardware accelerator in the BCI system.
3) A detailed design-space exploration to identify Pareto-optimal configurations in terms of latency and accuracy.
4) The implementation of KF accelerators that embed various approximation techniques and a comparative analysis.
5) The integration of KF hardware accelerators in a Linux-capable heterogeneous SoC, deployment on FPGA, and application to the processing of real brain data.

1: **function** KALMAN_FILTER($F, Q, H, R, \vec{x}_{n-1}, P_{n-1}, \vec{z}_n$)
2: //Predict
3: $\vec{x}_n = F \cdot \vec{x}_{n-1}$
4: $P_n = F \cdot P_{n-1} \cdot F^t + Q$
5: //Update
6: $\vec{y} = \vec{z}_n - (H \cdot \vec{x}_n)$ //Innovation
7: $S = H \cdot P_n \cdot H^t + R$
8: $K = P_n \cdot H^t \cdot S^{-1}$ //Compute Kalman Gain
9: $\vec{x}_n = \vec{x}_n + K \cdot \vec{y}$
10: $P_n = (I - K \cdot H) \cdot P_n$
11: **return** $\vec{x}_n, P_n$

Fig. 2: The Kalman Filter algorithm.

II. BACKGROUND

Computation in the BCI System. Online BCI applications compute over high-dimensional data that streams in from multiple *channels*. In invasive BCIs, these channels are implanted micro-electrodes that record electrical activity of localized groups of neurons [4]–[6]. While using invasive BCI systems is the most promising approach to improve the resolution of neural data [3], [4], running BCI applications close to the source becomes a very challenging task due to harsh biological constraints. In order to prevent cellular damage to the brain tissue, the power consumption of the implanted chip cannot increase the surrounding temperature by more than $1-2$°C [34].

For this reason, BCI systems have been decomposed into two main components: an implanted chip and an external relay station [5], [26]. The implanted chip records neural data and communicates it wirelessly to the relay station. The relay station executes applications in real time and can consume only up to $\sim 200mW$ within the BAN [7], [27], [35]. These BCI applications include machine learning (ML) algorithms that decode the neural data and classify it according to the desired application [13], [14], [36]. For example, BCI applications can be used in the fields of vision [37], authentication [38] and most commonly in motion decoding [13], [14], [16].

Motion Decoders for BCI. Traditionally, BCI applications of motion decoding from neural data have utilized linear methods for predicting kinematics (e.g., the Kalman Filter), which provide stable but limited accuracy [13], [39]. New methods of motion decoding use non-linear deep neural networks [40]–[42]. While these methods can provide high accuracy in some cases, they can impose long training times and also suffer from overfitting, due to the high complexity and variability of neural data [13], [40]. As a result, another group of decoders is emerging. These decoders use linear methods, such as the Kalman Filter (KF), as the main decoding component in combination with modern non-linear ML techniques to improve accuracy [14], [19], [43]–[45]. In this way, these decoders not only utilize the robustness of well-proven algorithms like the KF but also have the improved accuracy of non-linear methods.

The Kalman Filter Algorithm. The KF is a recursive algorithm that uses a series of measurements over time to predict the state of desired variables [16]. Fig. 2 reports the main function that is executed at each time step of the algorithm. Each step outputs a prediction state vector ($\tilde{x}_n$), which holds a value for each of the desired variables, and the updated covariance matrix (P_n), which estimates the current accuracy of the results.

We use x as the dimension of the state vector and z as the dimension of the measurement vector. The KF algorithm receives 5 matrices as inputs: *(i)* the previous $P_{x\times x}$, *(ii)* $F_{x\times x}$ – the state transition model, which represents the probability of transitioning from one state to another, *(iii)* $Q_{x\times x}$ – the process noise covariance, which determines the uncertainty in the transition between states, *(iv)* $H_{z\times x}$ – the observation model over the measurements, which models the relationship between the different states and the observations made, and *(v)* $R_{z\times z}$ – the observation noise covariance, which describes the error in the measurements. In the traditional KF, the matrices F, Q, R, H remain constant between consecutive iterations of the KF and constitute the *KF model*. At each iteration, the KF receives the previous $\vec{x}_{n-1}$ and a new measurement vector $\vec{z}_n$.

The KF executes the "predict" step and then the "update" step. Calculating the Kalman gain K by inverting the matrix $S_{z\times z}$ lies at the core of the KF computation.

TABLE I: The Accuracy of the KF with Different Methods

Accuracy Metric	Gauss [50]	IFKF [23]	Taylor [22]	SSKF [31]	Newton [29]
MSE	3.8×10^{-12}	53.8	0.05	0.1	6.6×10^{-6}
MAE	7×10^{-7}	2.7	0.08	0.06	0.0004
*Max. Difference (%)	0.008	2.2×10^{4}	9.7×10^{2}	5.3×10^{2}	4
*Avg. Difference (%)	0.0001	350	9	4.8	0.035

*These scores are normalized with respect to the original KF output [13].

Kalman Filter Design for BCI. With the increase in the number of channels in BCIs [2], the size of the measurements increases. Consequently, the KF requires large matrix operations, particularly the inversion of a large matrix. For edge devices, this requirement is often addressed by a hardware KF implementation [24], [25], [46]. In many cases, the KF incorporates an approximation method, sacrificing accuracy to meet throughput and power constraints [22], [23], [32]. We aim to design a hardware accelerator that introduces no more than $\sim$10% error compared to the standard KF, ensuring adequate control over fine motor tasks with an embedded BCI [47]–[49].

TABLE I reports KF accuracies when integrated with several candidate computation techniques from literature, which we implemented in software. For all methods, the KF predicts motion based on neural data from the brain of a non-human primate (NHP) [13], [51] and runs for 100 KF iterations. The KF output is compared to the one provided by Glaser *et al.* [13].

Gaussian elimination (Gauss) [50] is the standard method for matrix inversion [8], [24], [25] and provides the most accurate result, as it calculates the matrix inverse directly. However, Gauss suffers from high complexity ($O(n^3)$) and internal dependencies that limit parallel processing. Inverse Free KF (IFKF) [23], Taylor expansion of K (Taylor) [22], Steady-State KF (SSKF) [31] and the Newton-Raphson method (Newton) [29] offer different approximation techniques of K or S^{-1}. IFKF provides the worst accuracy, because it requires dimensionality reduction of the measurements and assumes minimal cross-correlation, despite the high correlation in simultaneous neural data measurements [2], [3]. Other methods provide sufficient accuracy when tested on neural data. Nevertheless, Newton provides the best accuracy among these approximations. Our work is the first to embed Newton inside the KF. In Section III, we explain our approach in detail.

III. KALMMIND

Our goal is to design KF hardware accelerators that are specialized for the requirements of an embedded BCI system.

Our architecture offers design flexibility and control over the potential trade-off between computational accuracy and latency.

Reorganization of the KF Algorithm. Traditionally, the KF begins by computing initial predictions of $\tilde{x}_n$ and P_n, then updates them with additional information from the KF model at each iteration (Fig. 2). While the main bottleneck of the KF is the computation of K (line 8), K is independent from the current $\tilde{x}_n$, the current $\tilde{z}_n$ and the innovation y (line 6). K only depends on the initial prediction of P_n (line 4) and S (line 7). As a result, we reorganize the KF in order to leverage parallel processing and allow more flexibility for hardware design.

In Fig. 1, we identify the main modules of the KF and their dependencies, then isolate the computation of K (compute K), enabling its easy modification with different computation techniques. Specifically, since K is independent from the input measurements, new measurements can be processed in parallel to the compute K module. In addition, we can switch between different matrix inversion modules. In some cases, we can even pre-compute K or S^{-1}, pre-load them into the memory of the device, and avoid their online computation [31]. Since our KF is designed for BCI, we propose a technique to efficiently compute the inverse matrix by storing and propagating information from earlier iterations of processed neural data.

Calculation of the Matrix Inverse. We define *calculation* as directly calculating the matrix inverse without using any approximations. Gauss is the standard calculation method for the matrix inverse. While generally accurate, it relies on floating-point divisions, which can introduce numerical errors. Moreover, each element in the inverse depends on the calculation of other elements, limiting the ability to decompose the calculation and exploit parallel processing. Alternative methods such as LU factorization [52], Cholesky decomposition (Cholesky) [53] and QR decomposition (QR) [54] have been proposed. These methods can reduce both the number of operations and the numerical errors. However, they still demonstrate dependencies between operations and experience increased memory usage.

Matrix Inverse Approximation. Reducing computational latency is typically achieved by using *approximation* techniques [22], [23], [55]. Approximation requires fewer computational steps and may reduce dependencies between operations, thereby facilitating more parallel processing.

The Newton-Raphson method (Newton) is one of the oldest methods for approximating the matrix inverse [29], [56]. As it requires simple matrix multiplications, it allows for parallel processing of the computation [30]. The method is recursive, and increasing the recursion depth with more iterations improves the approximation of the inverse [57]. Specifically, an iterative approximation of a matrix inverse is given by:

$$V_{i+1} = f_approx(V_i, A) \tag{1}$$

where $f_approx(\cdot)$ computes an approximated matrix inverse V_{i+1} from the previous approximation V_i and the matrix to invert A. The Newton method defines the iterative process as:

$$V_{i+1} = V_i \cdot (2I - A \cdot V_i), \quad i = 0, 1, \ldots, m-1 \tag{2}$$

where the iterative process executes m iterations and V_m is the final output of the process. Since it does not involve divisions, approximation is less prone to numerical errors.

For the Newton method to converge after a minimal number of iterations, it is crucial to select a reliable *initial seed*. The initial seed V_0 must comply with the constraint:

$$\|I - A \cdot V_0\|_2 < 1 \tag{3}$$

which means that the seed is not too far from the optimal A^{-1}.

Combining Calculation and Approximation. Neural datasets are highly diverse, as they are recorded from different brain regions and neural interfaces, each requiring varying levels of computational accuracy and real-time performance.

We propose a new technique, which interleaves two different methods between consecutive KF iterations: one for calculation of the inverse and another for approximation of the inverse. The advantage of our technique lies in balancing a highly accurate but costly calculation with a faster, less accurate approximation.

At each iteration of the KF, one of the two methods is selected. The frequency of calculating the inverse is set by a user-configuration parameter (*calc_freq*). The inverse is calculated at every n-th iteration of the KF where $n \ \% \ calc_freq = 0$.

In approximation, S_n is the matrix to invert at the n-th iteration of the KF (Fig. 2). The choice of V_0 is based on one of the two policies we develop, which initialize the approximation using a matrix inverse computed at a previous KF iteration. We propose two *seed policies* as follows:

$$V_{1,n} = S_{n-1}^{-1} \cdot (2I - S_n \cdot S_{n-1}^{-1}) \tag{4}$$

$$V_{1,n} = S_j^{-1} \cdot (2I - S_n \cdot S_j^{-1}), \quad j = n - n \ \% \ calc_freq \tag{5}$$

where $V_{1,n}$ is the result of the first internal iteration of inverse approximation in the n-th iteration of the KF. Equation (4) defines a straightforward policy for initializing the approximation with $V_0 = S_{n-1}^{-1}$, the matrix inverse from the previous KF iteration (n-1). Equation (5) defines a policy where only a calculated inverse can serve as a seed, mitigating potential inaccuracies inherent in an approximated inverse. It sets $V_0 = S_j^{-1}$ where j is the last iteration that has utilized an inverse calculation.

These policies are particularly effective for BCI, as they utilize the strong temporal and spatial correlations between neural data measurements, leveraging an inverse matrix computed for previous measurements to approximate the inverse matrix for current measurements [2], [3].

Overall, using approximation with a small number of recursive iterations is expected to yield lower accuracy but improved latency. Frequent use of calculation should enhance accuracy with higher latency. Nonetheless, calculation may introduce numerical errors, potentially degrading accuracy compared to approximation with a high count of inner iterations.

IV. Configurable Hardware Acceleration

High-Level Architecture. The KalmMind accelerator consists of three main functions: *load*, *compute* and *store* (Fig. 3a). It exposes 7 memory-mapped configuration registers that control its communication with main memory and, in particular, the matrix operations executed by the *compute* function (Fig. 3b).

The `x_dim` and `z_dim` registers configure the dimensions of the matrices (F, Q, H, R, P) and the sizes of the vectors $(\vec{x}, \vec{z})$ to be expected by the accelerator. The `chunks` and `batches` registers configure the number of KF iterations to be done in one invocation of the accelerator and provide fine-grained control of the number and size of its direct-memory access (DMA) transactions. The `approx`, `calc_freq` and `policy` registers control the dataflow of computation inside the accelerator by selecting the data path used to invert S at each iteration of the KF (Fig. 3b). Specifically, `calc_freq` sets the frequency (in terms of KF iterations) of calculating the inverse; `approx` determines how many internal iterations of approximation will be executed per KF iteration; `policy` sets the seed policy according to Equation (4) or Equation (5).

Communication and Local Memories. The *load* function loads F, Q, H, R and the initial $\vec{x}_0, P_0$ from main memory (Section II) and stores them in private local memories (PLMs) inside the accelerator [58]. The PLMs are implemented as multi-bank memories that expose multiple read/write ports; their size is configurable at design time. The matrices F, Q, H, R can be reused in consecutive iterations of the KF without reloading them from main memory. For each subsequent DMA transaction, `chunks` configures the number of measurement vectors to be loaded from main memory. *load* receives (`chunks`×`z_dim`) measurements and stores them in the PLM. The *store* function sends the computed state vectors $(\vec{x}_n)$ and the state covariance matrices (P_n) to main memory. The `batches` register configures the total number of DMA transactions to be executed in one invocation of the accelerator.

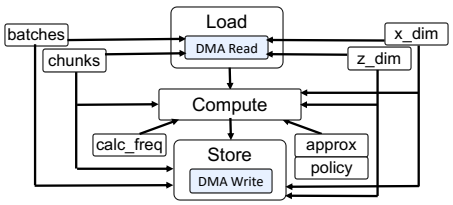
KF Computation. Fig. 3b shows how the *compute* function implements the KF with blue arrows to mark the computation path of the Kalman gain K. The state vector $\vec{x}_n$ is small, as it represents motion kinematics (Section II); hence, every computation with dimension `x_dim` is relatively short. As a result, we chose to implement these computations with the objective of reusing hardware resources instead of increasing throughput. With the exception of the matrix inverse, all matrix operations in Fig. 3b are fully pipelined. The inner-most accumulation loops are not unrolled in order to save hardware resources.

At each KF iteration, *compute* uses a double-buffer to store both the previous and new $\vec{x}_n$ and P_n (Fig. 3b). The double-buffers are swapped at the end of every iteration. The total number of iterations in one invocation is set to (`batches`×`chunks`).

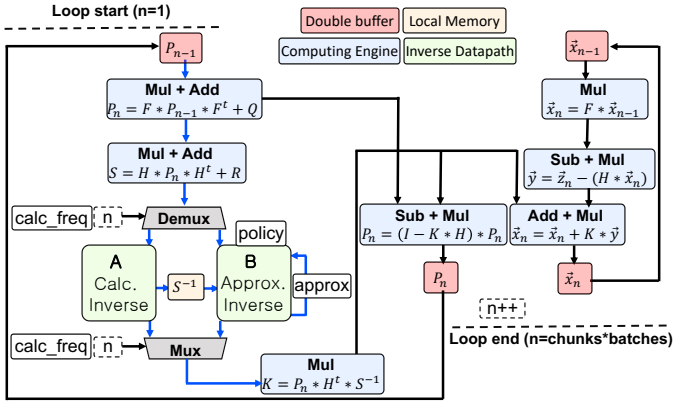
Thanks to our modular architecture, we are also able to replace the computation of K at design time with an approximation technique (Taylor) or a constant K (SSKF) and easily change the datatype between floating-point and fixed-point [32].

Implementation of the Matrix Inverse. At each KF iteration, a matrix inversion method is chosen according to the current KF iteration (`n`) and the configuration of `calc_freq`.

Path A in Fig. 3b implements a calculation method for matrix inversion. We refactor the computation to minimize the number of read/write interactions with the PLM and to be able to fully pipeline the computation of the inner-most loop. In Section V, we implement this path with Gauss, Cholesky and QR. We also replace it with a pre-computed constant S^{-1} (SSKF Inverse),



(a) Main accelerator functions.



(b) The compute function.

Fig. 3: A configurable KF hardware accelerator architecture.

inspired by the work of Malik *et al.* [31].

Path B in Fig. 3b implements an approximation method. For most of our accelerators, we implement the Newton method. We use 8 multiply-and-accumulate (MAC) units in parallel to accelerate the computation of matrix multiplications. In its first internal iteration, path B uses the seed specified by `policy`. If set to 0, the policy is set to Equation (5) and the most recently calculated inverse from path A is used as the seed. If set to 1, the policy is set to Equation (4) and the inverse from the previous KF iteration is used instead. Path B performs a fixed number of internal iterations as set by `approx`.

Overall, path B is expected to provide a better throughput compared to path A when `approx` is small, while the accuracy of path A per iteration should be higher. When `approx` grows larger, the accuracy is improved and can surpass that of path A, due to numerical errors in the inverse calculation. Interleaving the two paths in different ways offers various combinations of latency and accuracy in the KF.

V. Experimental Evaluation

Methodology: Three datasets of electrocorticography neural data from distinct brain regions are tested: the motor cortex of a non-human primate (NHP) [51], the somatosensory cortex of an NHP [42], and the hippocampus of a rat [59]. These datasets were originally tested with a KF using an inverse method that combines Gauss with LU factorization implemented with Python NumPy [13]. NumPy is the *reference* implementation. When a pure Gauss is employed in all KF iterations, it represents the *baseline*. Hardware accelerators are designed in C/C++ with Vivado HLS 2019.2. We name the accelerators according to their embedded calculation/approximation methods or with a designated name. The accelerators use 32-bit floating-point data types unless specified otherwise.

TABLE II: Accuracy Ranges with Three Neural Datasets

	MSE	MAE	Max Diff.
Motor	2.1×10^{-13}–1.1×10^{-6}	2×10^{-7}–1.6×10^{-4}	4.3×10^{-5}–1.91
Soma.	2.2×10^{-13}–9.9×10^{-6}	2.3×10^{-7}–5.1×10^{-4}	3.5×10^{-5}–5.3
Hippo.	3.1×10^{-11}–7.1×10^{-11}	1.2×10^{-6}–2.2×10^{-6}	8.2×10^{-5}–2.1×10^{-3}
Baseline	4.8×10^{-13}, 3×10^{-13}, 3.5×10^{-11}	2.7×10^{-7}, 2.7×10^{-7}, 1.4×10^{-6}	1.1×10^{-4}, 8.5×10^{-5}, 3.8×10^{-4}

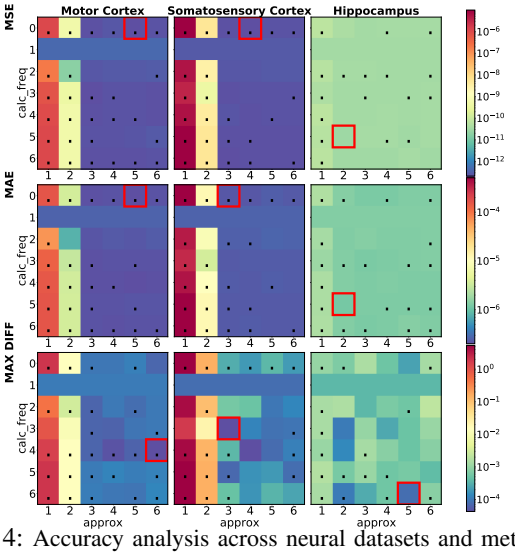


Fig. 4: Accuracy analysis across neural datasets and metrics.

SoC Integration: Hardware accelerators are synthesized with Vivado 2019.2 and leverage the open-source ESP platform [60]. ESP provides a tiled SoC architecture connected by a network-on-chip (NoC) with a library of heterogeneous components to facilitate "mix-and-match" SoC design. By following the accelerator integration flow [61], ESP is utilized for its implementations of DMA, memory-mapped registers, and interrupts that allow for seamless usage of an accelerator within a complete heterogeneous SoC architecture. ESP is also leveraged to generate FPGA prototypes of complete SoCs combining accelerators along with a 64-bit CVA6 RISC-V processor [33], an I/O tile, and a memory channel tile. The SoCs are tested on the Xilinx Virtex UltraScale XCVU440 FPGA board with a clock frequency of 78MHz, which is set according to the critical path of CVA6. We develop custom Linux software applications that run on the CVA6 and invoke the accelerators.

Accuracy Analysis. The predictions provided by the Gauss/Newton accelerator are analyzed with different configurations. The motor dataset dimensions are $\{x{=}6, z{=}164\}$, the somatosensory dataset dimensions are $\{x{=}6, z{=}52\}$, and the hippocampus dimensions are $\{x{=}6, z{=}46\}$. We run the accelerator in simulation for 100 iterations on each of the datasets. For `approx`, we use a range of 1-6, which means that Newton can have up to 6 internal iterations. For `calc_freq`, we use a range of 0-6 which means that Gauss can run in each iteration of the KF (`calc_freq=1`), every `calc_freq` iterations (`calc_freq=2`$-$`6`), or only at the first iteration (`calc_freq=0`). We compare the output at each KF iteration to the output from the reference and calculate three accuracy metrics: *(i)* Mean Square Error (MSE), *(ii)* Mean Absolute Error (MAE), and *(iii)* the normalized maximum difference between one output and its expected value (MAX DIFF).

Fig. 4 visualizes the accuracy results: purple denotes the highest accuracy while red denotes the lowest. For each pair of `calc_freq` and `approx`, we report the better result between the seed policies (`policy=1` is marked with a dot). For each metric and dataset, the highest accuracy is outlined in red.

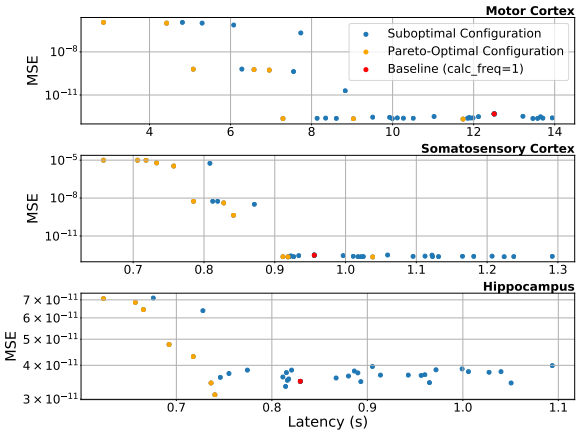


Fig. 5: Latency vs. accuracy with the Gauss/Newton accelerator.

The results show that, thanks to the flexibility of the accelerator, we can adjust the computation across a wide range of accuracies. TABLE II summarizes the accuracy ranges to which the accelerator can be configured for each dataset and metric. In all cases, we can even find a configuration that provides better accuracy than the baseline. This occurs because Newton can reduce the numerical errors introduced by Gauss by avoiding floating-point divisions. The best result is for MAX DIFF and the hippocampus, with a 78% improvement in accuracy.

It is important to note that, with the same set of configurations, each neural dataset achieves its best accuracy with a different configuration. In addition, the neural datasets from the NHP correspond to different accuracy ranges compared to the dataset from the rat. This highlights the importance of matching a neural dataset with a specific KF configuration to achieve sufficient accuracy and optimize performance. A configurable KF, such as the one we propose, offers this flexibility and fosters more effective and specialized BCI applications.

Accuracy vs. Latency. We combine the previous analysis with the latency of the Gauss/Newton accelerator when running on FPGA. Fig. 5 shows the results for each of the datasets with the MSE metric. The plots highlight Pareto-optimal points, each representing a specific tradeoff between accuracy and latency. The Pareto-optimal points result from different interleaving patterns of the two matrix inverse methods; for example, the point that provides the least latency in each plot has `approx=1` and `calc_freq=0`, while the point with the best accuracy in each plot has `approx`$\geq$`2`. In each plot, we can find configurations with comparable or better accuracy than the baseline with less latency and ultimately a better performance of the KF.

Overall, this analysis provides valuable insights for a KF hardware design tailored to specific application constraints.

Comparative Analysis of KF Implementations. We use KalmMind to design a set of hardware accelerators. TABLE III summarizes their FPGA resources, power, performance range, energy range, and accuracy range, measured while running 100 KF iteration on FPGA. It also includes the baseline implementation on an Intel i7 processor with a clock frequency of 3.7GHz, and on the CVA6 with a clock frequency of 78MHz (the same as the accelerators). We experiment with the largest

TABLE III: FPGA Resources and Performance across KF Implementations/Accelerators

Type	Method	LUT	FF	BRAM	DSP	Power [W]	Perf. [sec]	Energy [J]	Accuracy [MSE]
Software	Intel i7	N/A	N/A	N/A	N/A	78.6	0.065	5.1	3.8×10^{-12}
	CVA6	43996	29922	36	27	0.177	1927	341	1.3×10^{-12}
Hardware: Calc./Approx. Datapath	Gauss/Newton	22119	18725	228	252	0.185	2.8−8.9	0.52−1.64	$1.03\times10^{-12}-1.1\times10^{-6}$
	Cholesky/Newton	22429	20126	360	268	0.207	2.8−11.5	0.58−2.38	$1.05\times10^{-12}-1.1\times10^{-6}$
	QR/Newton	24842	21259	385	258	0.236	3.04−9.6	0.72−2.27	$1.02\times10^{-12}-1.1\times10^{-6}$
	Gauss/Newton FX32	19646	12131	195.5	217	0.146	4.25−4.25	0.354	$5.9\times10^{-2}-0.46$
	Gauss/Newton FX64	34831	26109	369	534	0.18	2.44−11.3	0.44−2.04	$1.9\times10^{-5}-0.24$
Hardware: One-way Datapath	LITE	15591	13405	146.5	193	0.114	2.688	0.306	1.14×10^{-6}
	LITE FX64	14782	8075	267	347	0.11	2.268	0.249	1.14×10^{-6}
	SSKF/Newton	18798	16961	204.5	240	0.158	0.53−11.6	0.08−1.82	$9.9\times10^{-13}-6.3\times10^{-5}$
	SSKF	8403	6752	19.5	102	0.051	0.03	0.0015	7.63×10^{-3}
	Taylor	15006	13437	118	230	0.155	1.203	0.186	2.3×10^{-3}
	Gauss-Only	12386	10290	102.5	153	0.098	12.507	1.225	1.3×10^{-12}

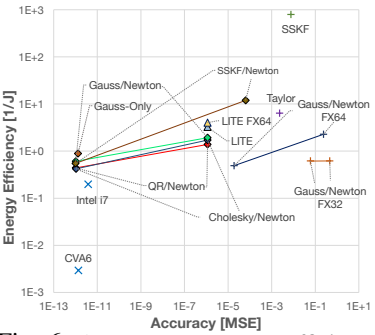


Fig. 6: Accuracy vs. energy efficiency.

dataset, the motor dataset, which requires a KF to run in less than $50ms$ per iteration for real-time BCI execution [28], [51].

One group of accelerators is implemented with unique calculation/approximation datapaths, using Gauss, QR, and Cholesky calculation methods alongside Newton approximation. The other group is implemented with one-way datapaths, where each accelerator uses either a calculation or approximation method: *(1)* LITE is designed to run Newton with one internal iteration and a seed from the previous KF iteration. In the first KF iteration, LITE loads a pre-computed seed from main memory. *(2)* SSKF embeds the method by Malik *et al.* [31]. The method is BCI-specific and approximates a constant K instead of the compute K module (Fig. 1). *(3)* In SSKF/Newton, we approximate a constant S^{-1}_{const}. We added the option to run Newton in order to improve the accuracy of S^{-1}_{const}. *(4)* Taylor integrates the method by Liu *et al.* [22], which approximates K at every KF iteration, avoiding the matrix inverse. *(5)* Gauss-Only always calculates the inverse with our optimized implementation of Gauss. *(6)* We provide accelerators with 64-bit (FX64) and 32-bit (FX32) fixed-point data types [32].

All accelerators, except Gauss-Only, demonstrate real-time execution by completing 100 KF iterations in under $5sec$. All accelerators meet the low-power constraint, consuming up to $\sim 200mW$. This proves that KalmMind successfully produces accelerators suitable for real-time embedded BCI systems.

Fig. 6 shows the tradeoff between accuracy and energy efficiency. The energy efficiency was calculated by inverting the energy result. Among the accelerators that use both calculation and approximation, Gauss/Newton has the best energy efficiency with a $10\times$ improvement compared to Intel i7 and $655\times$ compared to CVA6. SSKF provides the best energy efficiency with a $346\times$ improvement over Gauss/Newton. This is expected as SSKF is the only accelerator that does not compute K. However, the accuracy of SSKF is $10^9\times$ worse than Gauss/Newton and $10^3\times$ worse than LITE. Our SSKF/Newton offers the widest range of accuracy, achieving up to $15.3\times$ better energy efficiency compared to the standard Gauss (Gauss-Only).

Overall, KalmMind compensates for higher average power consumption, resulting from a more complex datapath, by offering substantially better energy efficiency and greater control over accuracy. KalmMind-based accelerators can be roughly divided into three tiers of accuracy. As accuracy decreases from the highest to the lowest tier, energy efficiency improves. Depending on the needs of the application, the neural data, and device constraints, the optimal KF accelerator can be chosen.

VI. Discussion and Related Work

The KF has been used frequently for motion decoding in BCI [13], [14], [16], [21], [31], [47], [62], [63]. The KF models we use to demonstrate KalmMind are trained according to the method of Wu *et al.* [16], and provided by Glaser *et al.* [13].

Usually, BCI applications utilize a Gauss-based KF [13], [14], [19]–[21]. They use the KF with additional ML models and continuously update the KF model. KalmMind can optimize the KF part of these applications for real-world embedded BCIs by managing accuracy, latency, and energy efficiency.

In Section II, we evaluate 4 different approximation techniques from literature [22], [23], [29], [31]. While using Newton [29] achieves the most accurate results for motion prediction from neural data, we use it in Section III to design a BCI-tailored technique that relies on propagating information between consecutive iterations of the KF. We are not aware of other implementations that use Newton within the KF. We suppose that this is because, prior to this work, no suitable seed policy has been identified. In addition, we design optimized accelerators for fixed-point datatypes [32], Taylor [22] and SSKF [31]; SSKF has been previously used for BCI [47], [62].

Other hardware implementations of the KF have been created to address computation in a mobile edge setting [8], [22], [24], [25], [32], [46], [64]–[67]. They are optimized for speed and low power consumption, but most are designed for domains other than BCI, have been tested with relatively small measurement vectors, and rely on costly calculations rather than approximations. However, SCALO [8] is an accelerator-rich distributed BCI system, where the inverse matrix for the KF is computed using Gauss. With KalmMind, they could leverage approximations with sufficient accuracy, reduce data dependencies, and enhance the scalability of the BCI system.

KalmMind is the first architecture to facilitate the design of configurable KF hardware accelerators for BCI, offering flexibility and uniquely supporting fine-grained control over latency and accuracy to address the diversity of brain data.

VII. Conclusion

We designed KalmMind to advance research on hardware architectures for embedded BCIs and to accelerate the development of practical, real-world BCI systems and specialized solutions for BCI applications. The contributions of this work have been released to the public domain[1].

[1]https://github.com/GuyEichler/KalmMind

REFERENCES

[1] R. P. Rao, "Brain Co-Processors: Using AI to Restore and Augment Brain Function," *Handbook of neuroengineering*, 2020.

[2] A. E. Urai *et al.*, "Large-Scale Neural Recordings Call for New Insights to Link Brain and Behavior," *Nature neuroscience*, 2022.

[3] I. H. Stevenson *et al.*, "How Advances in Neural Recording Affect Data Analysis," *Nature neuroscience*, 2011.

[4] Y. Wang *et al.*, "Implantable Intracortical Microelectrodes: Reviewing the Present with a Focus on the Future," *MNE*, 2023.

[5] N. Zeng *et al.*, "A Wireless, Mechanically Flexible, 25m-Thick, 65,536-Channel Subdural Surface Recording and Stimulating Microelectrode Array with Integrated Antennas," in *VLSI*, 2023.

[6] E. Musk *et al.*, "An Integrated Brain-Machine Interface Platform with Thousands of Channels," *JMIR*, 2019.

[7] G. Eichler *et al.*, "MasterMind: Many-Accelerator SoC Architecture for Real-Time Brain-Computer Interfaces," in *ICCD*, 2021.

[8] K. Sriram *et al.*, "SCALO: An Accelerator-Rich Distributed System for Scalable Brain-Computer Interfacing," in *ISCA*, 2023.

[9] W. Choi *et al.*, "A 1,024-Channel, 64-Interconnect, Capacitive Neural Interface Using a Cross-Coupled Microelectrode Array and 2-Dimensional Code-Division Multiplexing," in *VLSI*, 2023.

[10] J.-H. Cha *et al.*, "A Reconfigurable Sub-Array Multiplexing Microelectrode Array System with 24,320 Electrodes and 380 Readout Channels for Investigating Neural Communication," in *ISSCC*, 2022.

[11] C. Urrea *et al.*, "Kalman Filter: Historical Overview and Review of its Use in Robotics 60 Years after its Creation," *Journal of Sensors*, 2021.

[12] X. Gao *et al.*, "RL-AKF: An Adaptive Kalman Filter Navigation Algorithm based on Reinforcement Learning for Ground Vehicles," *Remote Sensing*, 2020.

[13] J. I. Glaser *et al.*, "Machine Learning for Neural Decoding," *Eneuro*, 2020.

[14] A. D. Degenhart *et al.*, "Stabilization of a Brain–Computer Interface via the Alignment of Low-dimensional Spaces of Neural Activity," *Nature biomedical engineering*, 2020.

[15] M. Śliwowski *et al.*, "Decoding ECoG Signal into 3D Hand Translation Using Deep Learning," *J. Neural Eng.*, 2022.

[16] W. Wu *et al.*, "Neural Decoding of Cursor Motion Using a Kalman filter," *Advances in neural information processing systems*, 2002.

[17] M. Vilela *et al.*, "Applications of Brain-Computer Interfaces to the Control of Robotic and Prosthetic Arms," *Handbook of clinical neurology*, 2020.

[18] A. Moly *et al.*, "An Adaptive Closed-Loop ECoG Decoder for Long-Term and Stable Bimanual Control of an Exoskeleton by a Tetraplegic," *J. Neural Eng.*, 2022.

[19] X. Zhang *et al.*, "Reinforcement Learning-Based Kalman Filter for Adaptive Brain Control in Brain-Machine Interface," in *EMBC*, 2021.

[20] EMBC, "A Stabilized Dual Kalman Filter for Adaptive Tracking of Brain-Computer Interface Decoding Parameters," 2013.

[21] V. Gilja *et al.*, "A High-Performance Neural Prosthesis Enabled by Control Algorithm Design," *Nature neuroscience*, 2012.

[22] Y. Liu *et al.*, "Efficient Mapping of a Kalman Filter into an FPGA Using Taylor Expansion," in *FPL*, 2007.

[23] K. S. Babu *et al.*, "Inverse Free Kalman Filter Using Approximate Inverse of Diagonally Dominant Matrices," *IEEE Control Systems Letters*, 2019.

[24] A. Valade *et al.*, "A Study about Kalman Filters Applied to Embedded Sensors," *Sensors*, 2017.

[25] M. Ordubağ *et al.*, "Model-Based Kalman Filter Design on an FPGA," in *ELECO*, 2021.

[26] Y. Gilhotra *et al.*, "A Wireless Subdural Optical Cortical Interface Device with 768 Co-Packaged Micro-LEDs for Fluorescence Imaging and Optogenetic Stimulation," in *CICC*, 2024.

[27] A. Y. Dogan *et al.*, "Multi-Core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *DATE*, 2012.

[28] J. P. Cunningham *et al.*, "A Closed-Loop Human Simulator for Investigating the Role of Feedback Control in Brain-Machine Interfaces," *Journal of neurophysiology*, 2011.

[29] A. Ben-Israel, "An Iterative Method for Computing the Generalized Inverse of an Arbitrary Matrix," *Mathematics of Computation*, 1965.

[30] V. Pan *et al.*, "Efficient Parallel Solution of Linear Systems," in *STOC*, 1985.

[31] W. Q. Malik *et al.*, "Efficient Decoding with Steady-State Kalman Filter in Neural Interface Systems," *IEEE TNSRE*, 2010.

[32] P. T. L. Pereira *et al.*, "Architectural Exploration for Energy-Efficient Fixed-Point Kalman Filter VLSI Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2021.

[33] F. Zaruba *et al.*, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *ITVL*, 2019.

[34] P. D. Wolf *et al.*, "Thermal Considerations for the Design of an Implanted Cortical Brain–Machine Interface (BMI)," *Indwelling Neural Implants: Strategies for Contending with the In Vivo Environment*, 2008.

[35] G. Udovičić *et al.*, "Wearable Technologies for Smart Environments: A Review with Emphasis on BCI," in *SoftCOM*, 2016.

[36] J. Lee *et al.*, "Hierarchical Optimal Transport for Multimodal Distribution Alignment," in *NeurIPS*, 2019.

[37] A. Lozano *et al.*, "Neurolight: A Deep Learning Neural Interface for Cortical Visual Prostheses," *Int J Neural Syst*, 2020.

[38] L. Feng *et al.*, "Brain Password: A Secure and Truly Cancelable Brain Biometrics for Smart Headwear," in *MobiSys*, 2018.

[39] D. Sussillo *et al.*, "Making Brain–Machine Interfaces Robust to Future Neural Variability," *Nature communications*, 2016.

[40] F. Liu *et al.*, "Deep Learning for Neural Decoding in Motor Cortex," *Journal of Neural Engineering*, 2022.

[41] L. Yao *et al.*, "Fast and Accurate Decoding of Finger Movements from ECoG through Riemannian Features and Modern Machine Learning Techniques," *Journal of Neural Engineering*, 2022.

[42] A. S. Benjamin *et al.*, "Modern Machine Learning as a Benchmark for Fitting Neural Responses," *Front. Comput. Neurosci.*, 2018.

[43] M. Asgharpour *et al.*, "Regularized Kalman Filter for Brain-Computer Interfaces Using Local Field Potential Signals," *J. Neurosci. Methods*, 2021.

[44] F. R. Willett *et al.*, "Principled BCI decoder design and parameter selection using a feedback control model," *Scientific reports*, 2019.

[45] G. Revach *et al.*, "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics," *IEEE Trans. Signal Process.*, 2022.

[46] P. Babu *et al.*, "FPGA Implementation of Multi-Dimensional Kalman Filter for Object Tracking and Motion Detection," *JESTECH*, 2022.

[47] Z. Irwin *et al.*, "Neural Control of Finger Movement via Intracortical Brain–machine Interface," *J. Neural Eng.*, 2017.

[48] A. K. Vaskov *et al.*, "Cortical Decoding of Individual Finger Group Motions Using ReFIT Kalman Filter," *Frontiers in neuroscience*, 2018.

[49] D. Shin *et al.*, "Prediction of Muscle Activities from Electrocorticograms in Primary Motor Cortex of Primates," *PloS one*, 2012.

[50] N. J. Higham, "Gaussian Elimination," *Wiley Interdisciplinary Reviews: Computational Statistics*, 2011.

[51] J. I. Glaser *et al.*, "Population Coding of Conditional Probability Distributions in Dorsal Premotor Cortex," *Nature communications*, 2018.

[52] J. Dongarra *et al.*, "High Performance Matrix Inversion Based on LU Factorization for Multicore Architectures," in *MTAGS*, 2011.

[53] A. Krishnamoorthy *et al.*, "Matrix Inversion Using Cholesky Decomposition," in *SPA*, 2013.

[54] A. Irturk *et al.*, "An Efficient FPGA Implementation of Scalable Matrix Inversion Core using QR Decomposition," 2009.

[55] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001.

[56] G. Schulz, "Iterative berechung der reziproken matrix," *ZAMM-Journal of Applied Mathematics and Mechanics*, 1933.

[57] F. Toutounian *et al.*, "An Iterative Method for Computing the Approximate Inverse of a Square Matrix and the Moore–Penrose Inverse of a Non-Square Matrix," *Appl. Math. Comput.*, 2013.

[58] C. Pilato *et al.*, "System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip," *TCAD*, 2017.

[59] K. Mizuseki *et al.*, "Multi-Unit Recordings from the Rat Hippocampus Made During Open Field Foraging," *CRCNS. org*, 2009.

[60] P. Mantovani *et al.*, "Agile SoC Development with Open ESP," in *ICCAD*, 2020.

[61] D. Giri *et al.*, "Accelerator Integration for Open-Source SoC Design," *IEEE Micro*, 2021.

[62] B. Jarosiewicz *et al.*, "Advantages of Closed-Loop Calibration in Intracortical Brain–Computer Interfaces for People with Tetraplegia," *J. Neural Eng.*, 2013.

[63] M. Asgharpour *et al.*, "Regularized Kalman filter for brain-computer interfaces using local field potential signals," 2021.

[64] D. Pritsker, "Hybrid Implementation of Extended Kalman Filter on an FPGA," in *RadarCon*, 2015.

[65] B. Xu *et al.*, "A Resource Saving FPGA Implementation Approach to Fractional Kalman filter," *IET Control Theory & Applications*, 2022.

[66] F. Sandhu *et al.*, "FPGA-Based Implementation of Kalman Filter for Real-Time Estimation of Tire Velocity and Acceleration," *IEEE Sens. J.*, 2017.

[67] J. Soh *et al.*, "An FPGA-Based Unscented Kalman Filter for System-on-Chip Applications," *IEEE Trans. Circuits Syst. II: Express Briefs*, 2016.

CIMFlow: An Integrated Framework for Systematic Design and Evaluation of Digital CIM Architectures

Yingjie Qi[†], Jianlei Yang[†§✉], Yiou Wang[†], Yikun Wang[†], Dayu Wang[†], Ling Tang[†],
Cenlin Duan[‡], Xiaolin He[†], Weisheng Zhao[‡]
[†] School of Computer Science and Engineering, Beihang University
[‡] School of Integrated Circuit Science and Engineering, Beihang University
[§] Qingdao Research Institute, Beihang University

***Abstract*—Digital Compute-in-Memory (CIM) architectures have shown great promise in Deep Neural Network (DNN) acceleration by effectively addressing the "memory wall" bottleneck. However, the development and optimization of digital CIM accelerators are hindered by the lack of comprehensive tools that encompass both software and hardware design spaces. Moreover, existing design and evaluation frameworks often lack support for the capacity constraints inherent in digital CIM architectures. In this paper, we present CIMFlow, an integrated framework that provides an out-of-the-box workflow for implementing and evaluating DNN workloads on digital CIM architectures. CIMFlow bridges the compilation and simulation infrastructures with a flexible instruction set architecture (ISA) design, and addresses the constraints of digital CIM through advanced partitioning and parallelism strategies in the compilation flow. Our evaluation demonstrates that CIMFlow enables systematic prototyping and optimization of digital CIM architectures across diverse configurations, providing researchers and designers with an accessible platform for extensive design space exploration.**

***Index Terms*—Digital Compute-in-Memory, Integrated Framework, Instruction Set Architecture, Compilation**

	NeuroSim [9]	MNSIM [10]	CIM-MLC [13]	CIMFlow
Design Level	System-level simulation	System-level simulation	Compilation framework	Integrated framework
SW/HW Co-design	Tightly coupled	Tightly coupled	-	Modularized
Datapath	Fixed	Configurable	High-level	Extensible
DNN Support	Limited	Limited	Flexible	Flexible
ISA Support	Not supported	Not supported	Partially supported	ISA-based framework

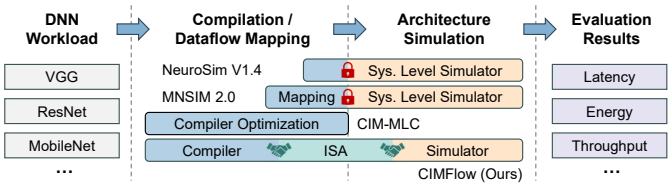


Fig. 1. Comparing CIMFlow with recent design and evaluation frameworks for CIM architectures.

I. Introduction

Recent advances in Deep Neural Networks (DNNs) have led to unprecedented achievements across various domains, significantly increasing the demand for efficient processing of deep learning workloads. However, traditional von Neumann architectures [1], [2] are hitting the "memory wall" due to frequent data transmission between separate computing and memory units [3]. As AI technologies continue to transform computing, Compute-in-Memory (CIM) architectures [4]–[6] have emerged as a promising solution for next-generation DNN accelerators, aiming to eliminate this bottleneck by integrating computation logic within memory arrays.

Mainstream CIM designs can be broadly categorized into analog [7]–[9] and digital [10]–[12] approaches. While analog CIM relies on current/voltage summation for computations, digital CIM embeds digital logic units directly into SRAM arrays. By avoiding analog-to-digital (ADC) and digital-to-analog (DAC) conversion overhead and non-ideality issues inherent in analog computations, digital CIM demonstrates both robust computation and enhanced parallelism, thus exhibiting great potential in DNN acceleration.

However, fully realizing the benefits of digital CIM across various DNN workloads is fraught with challenges. Current CIM architectures, like many domain-specific accelerators, are typically optimized for specific applications, limiting their adaptability to emerging workloads. As the field of AI continues to evolve, designers face a constantly expanding design space, comprising diverse DNN structures, dataflow mapping strategies, and various design choices in CIM accelerators. Navigating this complex landscape requires versatile tools capable of accommodating a wide range of software and hardware configurations.

While various software tools have been developed to facilitate CIM designs, existing approaches have notable limitations. On the one hand, many tools tend to focus predominantly on specific aspects of the design flow, such as hardware simulation [13]–[16] or dataflow compilation [17]–[19], lacking the holistic view necessary for effective design space exploration. On the other hand, most of these tools are primarily designed for analog CIM, and are only later adapted to support digital implementations, often overlooking crucial characteristics of digital CIM architectures.

In particular, SRAM-based digital CIM faces inherent capacity constraints due to its lower integration density compared to DRAM or emerging NVM solutions [20]. In practical

This work is supported in part by the National Natural Science Foundation of China (Grant No. 62072019), the Beijing Natural Science Foundation (Grant No. L243031), and the National Key R&D Program of China (Grant No. 2023YFB4503704 and 2024YFB4505601).
✉Corresponding author is Jianlei Yang. Email: jianlei@buaa.edu.cn.
The CIMFlow framework is available at https://www.cimflow.org/.

implementations, these density limitations often result in insufficient on-chip memory capacity for modern DNN models. This constraint could potentially negate the benefits of in-memory computing, as it necessitates frequent data movement and restricts opportunities for inter-layer pipeline optimization. Consequently, there is a pressing need for an integrated framework specifically tailored to the nuances of digital CIM, enabling comprehensive evaluation and rapid prototyping across diverse configurations.

To address these challenges, we introduce CIMFlow, an integrated framework for systematic design and evaluation of digital CIM architectures. As shown in Fig. 1, our framework integrates a highly extensible Instruction Set Architecture (ISA), a digital CIM-oriented compiler, and an efficient cycle-accurate simulator, enabling comprehensive and flexible exploration of diverse software and hardware configurations. The main contributions can be summarized as follows:

- We propose CIMFlow, an integrated framework that provides an out-of-the-box workflow for implementing and evaluating DNN workloads on digital CIM architectures. The framework enables systematic design space exploration through seamless integration of compilation and simulation infrastructures.
- We develop a CIM-specific ISA design that employs a hierarchical hardware abstraction, bridging compilation and simulation while providing flexible support for various architectural configurations.
- We implement an advanced compilation flow built on the MLIR infrastructure, addressing the capacity limitations in digital CIM architectures through innovative partitioning and parallelism strategies.

II. Background and Motivation

A. Digital CIM Preliminaries

CIM architectures represent a paradigm shift from traditional von Neumann computing by enabling computation directly within memory arrays. In essence, CIM architectures perform matrix-vector multiplication (MVM) through parallelized in-situ Multiply-Accumulate (MAC) operations within memory arrays, significantly reducing the data movement bottleneck that plagues conventional architectures. These MAC operations are typically carried out in a bit-serial fashion, where multiplications are decomposed into a series of bit-wise computations, followed by shift and accumulation operations.

A digital CIM macro comprises two key components: (1) a modified SRAM array that stores weight data and enables in-situ computations, and (2) peripheral circuits that orchestrate row-wise Boolean operations and accumulation [21]. Unlike analog CIM approaches, which face significant limitations due to area and power overhead from ADC and DAC converters, digital CIM enables simultaneous activation of the entire memory array. This enhanced parallelism, combined with the inherent robustness of digital computation, makes digital CIM particularly well-suited for accelerating deep learning workloads that exhibit high computational intensity and parallel structure. While researchers have proposed various digital CIM accelerators to harness these benefits [22], [23], scaling these solutions to meet the diverse computational demands of modern AI models remains a significant challenge.

B. Related Works and Our Motivation

The growing complexity of deep learning workloads and CIM architectures has created a critical need for tools and frameworks to bridge the gap between hardware and software. Existing frameworks broadly fall into two categories: modeling/simulation frameworks that evaluate architectural decisions through detailed performance analysis [13]–[16], and compilation frameworks that enable efficient mapping of DNN workloads to CIM hardware [17]–[19]. While modeling frameworks help hardware designers optimize latency, power, and area trade-offs, compilation frameworks provide support for managing the hardware-software interface and generating optimized dataflows for CIM architectures.

However, most existing frameworks have focused primarily on analog CIM architectures, given their prominence in current research. As illustrated in Fig. 1, recent simulation frameworks have begun extending support towards digital CIM implementations. NeuroSim V1.4 [15] extends its system-level simulation capabilities to support advanced digital CIM technology nodes, while MNSIM 2.0 [16] proposes a unified memory array model for both analog and digital CIM architectures. Although these tools provide valuable insights into architectural trade-offs, they often employ tightly-coupled co-design with fixed assumptions about DNN structures and datapath organizations, limiting their flexibility and applicability for comprehensive digital CIM exploration. On the compilation front, while traditional deep learning compilation infrastructures offer general-purpose solutions [24], frameworks such as CIM-MLC [19] introduce optimization strategies specifically designed for analog CIM. However, there remains a notable lack of dedicated compilation support for digital CIM architectures.

The above observations highlight three key challenges in digital CIM development. ❶ **Integration Gap:** The lack of comprehensive integration between different design stages necessitates an integrated framework that combines ISA definition, compilation, and simulation. ❷ **Limited Flexibility:** Fixed architectural assumptions in current tools might constrain design space exploration, calling for extensibility in CIM design and evaluation frameworks. ❸ **Resource Constraints:** The limited SRAM array capacity in digital CIM requires sophisticated dataflow management and parallelism strategies that existing works do not adequately address. To tackle these challenges, we propose CIMFlow, an integrated framework that enables flexible design space exploration through extensible ISA support and advanced compilation optimization techniques. By providing comprehensive support across the design stack, CIMFlow facilitates structured development and evaluation of digital CIM architectures while maintaining the flexibility to accommodate future innovations.

III. CIMFlow Framework

A. Framework Overview

The design of CIMFlow is guided by two core principles: **integration** for seamless exploration, and **extensibility** through

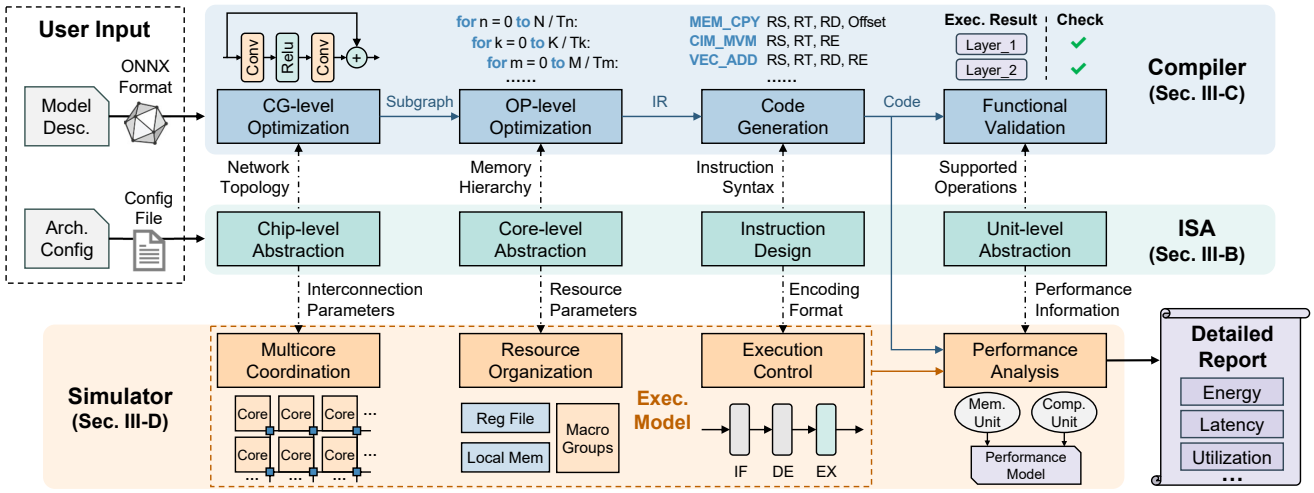


Fig. 2. Overview of the CIMFlow framework.

hierarchical abstractions. CIMFlow unifies the entire workflow from high-level DNN model description to detailed performance analysis, providing researchers with a comprehensive view of DNN workload execution on digital CIM architectures. Built with a modular design, CIMFlow readily adapts to emerging advances in CIM technology while abstracting implementation complexities through its streamlined interface, enabling designers and researchers to perform systematic evaluation and exploration.

As illustrated in Fig. 2, CIMFlow offers a cohesive workflow through three main components: a flexible ISA with hierarchical hardware abstraction (Sec. III-B), a compiler featuring multilevel optimization techniques (Sec. III-C), and a cycle-accurate simulator delivering detailed performance insights (Sec. III-D). The workflow begins with a DNN model description in ONNX format, complemented by an architecture configuration file that specifies the target CIM hardware parameters. The compiler first performs computational graph (CG) level optimizations for workload distribution and data management, followed by operator (OP) level optimizations to further maximize hardware efficiency. These optimizations are guided by the hardware specifications defined through the hierarchical hardware abstraction interface in ISA, spanning from chip-level interconnection to unit-level execution details. The generated code is validated by the compiler and then fed into the cycle-accurate simulator, which models the execution across multiple cores while tracking resource utilization and performance metrics. This process produces a detailed report covering energy consumption, latency, and hardware utilization, enabling comprehensive evaluation of different architecture designs.

B. ISA Design

As depicted in Fig. 3, the CIMFlow ISA implements a three-level hardware abstraction hierarchy complemented by a flexible instruction set design. Each abstraction level interfaces with the corresponding stages in the compilation and simulation infrastructure, providing architectural specifications that guide both compilation optimization and simulation execution.

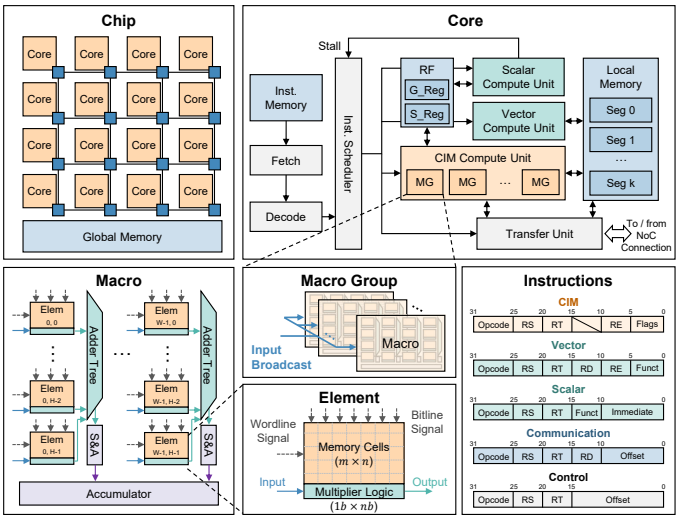


Fig. 3. Hardware abstraction and instruction design in the CIMFlow ISA.

Hardware Abstraction. At chip level, the architecture consists of multiple cores interconnected through a Network-on-Chip (NoC) structure, facilitating synchronous inter-core communication and global memory access. This organization enables scalable workload distribution and flexible inter-core pipelining, with each core functioning as a basic unit of program execution with its own instruction control flow.

The core-level abstraction defines the organization of hardware resources, encompassing instruction memory, various compute units, register files (RFs), and local memory. To facilitate efficient memory management and architectural extensibility, CIMFlow implements a unified address space across both global and local memories. In addition, the local memory is divided into segments to efficiently handle the input and output of DNN layers. The register file consists of general-purpose registers (G_Reg) for instruction-level access and special-purpose registers (S_Reg) for operation-specific functions.

At unit level, the CIM compute unit incorporates multiple macro groups (MGs) that support weight duplication and

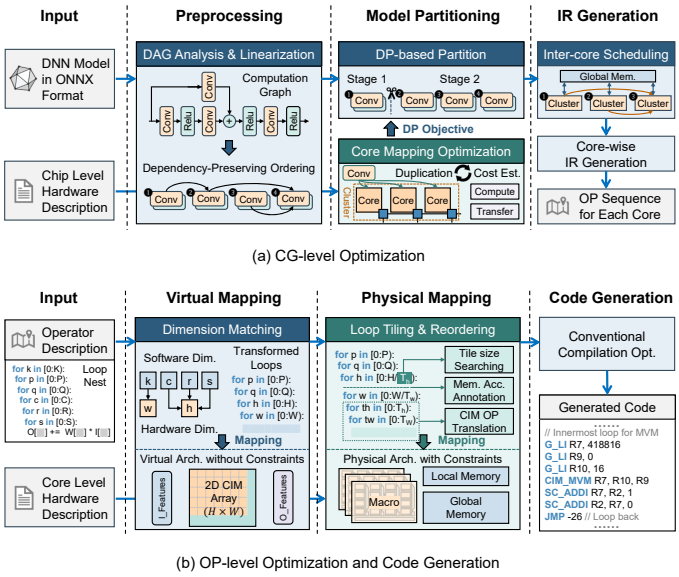


Fig. 4. Compilation flow and mapping optimization strategies in CIMFlow.

flexible spatial mapping strategies. Within each MG, weights are typically organized along the output channel, enabling efficient input data broadcast across macros for parallel in-memory MVM operations. The vector compute unit handles auxiliary DNN operations such as activation, pooling, and quantization. The scalar compute unit executes control flow operations through scalar arithmetic computations.

Instruction Design. To support efficient execution across the hardware hierarchy, CIMFlow implements a unified 32-bit instruction format with specialized variations for different operation types. Instructions are categorized into compute, communication, and control flow instructions, with compute instructions further specialized for CIM, vector, and scalar compute units. Each instruction contains a 6-bit operation specifier (opcode) and multiple 5-bit operand fields. Certain instruction types may also include supplementary fields, such as a 6-bit functionality specifier, execution flags, or immediate values of 10 or 16 bits. The instruction format supports up to four operands depending on the operation type, providing flexibility for complex operations while maintaining encoding efficiency. The instruction set is designed for extensibility through incorporating a customized instruction description template, which enables seamless integration of new operations into the framework when provided with their associated performance parameters.

C. Compilation Flow

The CIMFlow compiler bridges the semantic gap between high-level DNN models and low-level CIM operations through a two-level optimization strategy, as illustrated in Fig. 4. Starting with an ONNX model, the compiler first performs CG-level optimizations to partition and schedule workloads across multiple cores, effectively addressing the limited capacity issue. This is followed by OP-level optimizations built upon the MLIR infrastructure [24], which translates DNN operations into efficient CIM instruction sequences while taking into account the underlying hardware constraints.

Algorithm 1: DP-based partitioning and mapping

Input: Preprocessed computation graph $G = (V, E)$, Hardware resources R
Output: Optimal partitioning and mapping solution S

$D \leftarrow$ GetDependencyMasks(G) // Find all dependency closures in G and encode them as bitmasks
$dp \leftarrow [\infty]^{|D|}$, $prev \leftarrow [-1]^{|D|}$, $map \leftarrow [\emptyset]^{|D|}$
for $i \leftarrow 0$ **to** $|D| - 1$ **do**
  if $D[i] = \emptyset$ **then**
    $dp[i] \leftarrow 0$
    continue
  for $j \leftarrow 0$ **to** $i - 1$ **do**
    if $D[i] \& D[j] = D[j]$ **then**
      $stage \leftarrow D[i] - D[j]$ // Extract the set difference of dependencies as a partition
      $(cost, mp) \leftarrow$ OptimalMapping($stage, R$)
      if $dp[j] + cost < dp[i]$ **then**
        $dp[i] \leftarrow dp[j] + cost$
        $prev[i] \leftarrow j$
        $map[i] \leftarrow map[j] \cup mp$
$S \leftarrow$ ReconstructSolution($dp, prev, map$)
return S

CG-level Optimization. The optimization at this level begins with preprocessing the computation graph through analyzing the operator dependencies within the directed acyclic graph (DAG). During preprocessing, the compiler first identifies and extracts MVM-based operators, then groups adjacent operators with them to create a condensed CG. This analysis produces a dependency-preserving linear sequence of operators that forms the foundation for subsequent optimization stages.

To address the capacity limitation inherent in digital CIM architectures, CIMFlow implements a systematic partitioning strategy to divide the model into multiple execution stages. As detailed in Alg. 1, the model partitioning phase employs a dynamic programming (DP) based approach that optimizes workload distribution across available cores. The algorithm incorporates a state compression optimization that encodes all the dependency closures in the DAG as bitmasks, significantly reducing both space complexity and computational overhead. Each dependency closure represents a self-contained set of operators whose dependencies are fully enclosed within the set, serving as basic building blocks for candidate partitions.

The compiler derives candidate partitions through set operations on these dependency closures, and performs core mapping optimization for each partition. This process involves strategically duplicating operator weights across clusters of cores when deemed beneficial by the cost estimation model. To balance parallel execution benefits against communication costs, the estimation model accounts for both computation costs and data transfer overheads across inter- and intra-cluster communications. These cost assessments and their corresponding optimal mapping configurations are then used to guide the

DP-based partition selection.

The final phase of CG-level optimization focuses on inter-core scheduling and intermediate representation (IR) generation. The scheduler orchestrates data movement through the NoC interconnection, facilitating the inter-operator pipelines across different clusters. For each core, the compiler generates an optimized operation sequence incorporating both partitioning decisions and mapping destinations, establishing the foundation for OP-level optimizations.

OP-level Optimization. Following CG-level workload distribution, the compiler performs fine-grained operator transformations to maximize hardware efficiency. This process involves a structured approach that first establishes an ideal mapping in a constraint-free virtual space, and then adapts this mapping to actual hardware resource constraints.

The virtual mapping phase begins by analyzing the dimensional structure of each operator, transforming complex nested loops into a simplified version that aligns with the CIM array structure. This transformation process maps the software-level weight dimensions onto a two-dimensional array representation. By temporarily abstracting away physical constraints, the compiler explores the optimal weight data layout strategies, including the image-to-column (`im2col`) transformation commonly employed in DNN acceleration.

The physical mapping phase then adapts the idealized representation to actual hardware constraints through a series of optimization passes implemented within the MLIR infrastructure. The compiler first applies loop tiling based on resource capacity constraints, then systematically extracts MVM operations from the tiled loops for translation into CIM operations. Through automated analysis, it determines the optimal tile sizes and loop ordering to maximize computational efficiency while respecting resource limitations at each memory hierarchy. Memory access operations are then strategically annotated at appropriate loop levels to minimize data transfer overhead.

In the final code generation phase, the optimized IR undergoes conventional compilation techniques, including constant propagation, dead code elimination, and register allocation. The generated instructions adhere to the CIMFlow ISA specification while realizing the optimized resource mapping decisions, ensuring efficient utilization of the CIM hardware resources.

D. Simulator Design

The CIMFlow simulator provides cycle-accurate performance analysis through detailed modeling of the digital CIM architecture across multiple abstraction levels, from individual core execution to chip-level coordination. Implemented in SystemC [25], the simulator features a detailed pipeline model to track execution flow and resource utilization within each processing unit, while managing parallel execution across cores connected via NoC. The simulator supports diverse architectural configurations through a user-defined configuration file that adheres to the ISA specifications, while its modular design and standardized interfaces allow straightforward integration of custom architectural components.

TABLE I
ARCHITECTURE PARAMETERS OF THE DEFAULT ARCHITECTURE.

Chip Level		**Core Level**		**Unit Level**	
Core num.	64	CIM comp. unit	16 # MG	Macro	512×64
NoC flit size	8 Byte	Macro group	8 # macro	Element	32×8
Global mem.	16 MB	Local mem.	512 KB		

At core level, instruction execution follows a three-stage pipeline comprising instruction fetch (IF), decode (DE), and execute (EX). The EX stage implements detailed execution models for different compute units, each with fine-grained pipelining to enable instruction-level parallelism. Instruction conflicts and resource utilization are efficiently tracked through a bitmap-based scoring board within the instruction scheduler, ensuring accurate modeling of both computation and data movement patterns. Through this detailed modeling of both computation and data movement patterns, the simulator provides a comprehensive performance analysis across different architectural levels, tracking metrics such as energy consumption, execution latency, and hardware utilization for each unit. These detailed insights enable both quantitative evaluation of different CIM design choices and validation of compiler optimizations.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

To demonstrate how CIMFlow facilitates digital CIM architecture design, we conduct detailed analyses of compilation optimization strategies and present case studies exploring the impact of various architectural and software design choices. The default architecture parameters are carefully selected to efficiently support typical DNN workload characteristics while maintaining practical hardware constraints, as detailed in Tab. I. The performance statistics are acquired from multiple industry-standard tools to ensure accurate modeling of all components within the architecture. The CIM macro specifications are derived from post-layout analysis based on the design presented in [11], while other on-chip memory components are evaluated using memory compilers. The remaining digital logic modules are implemented in Verilog HDL and synthesized using Design Compiler, with power analysis conducted through PrimeTime PX. The NoC interconnection costs are modeled using Noxim [26].

We select representative DNN models that span different architectural characteristics and computational demands as our evaluation benchmark. The suite encompasses compute-intensive architectures including ResNet18 and VGG19, alongside compact models featuring depth-wise separable convolutions such as MobileNetV2 and EfficientNetB0. To align with digital CIM implementation constraints, the weights and activations of all models are quantized to INT8.

B. Compilation Optimization Evaluation

We evaluate our proposed compilation optimization strategies against two baseline approaches: (1) a generic mapping scheme that implements inter-layer pipeline without operator duplication, and (2) the CG-level partition and opportunistic

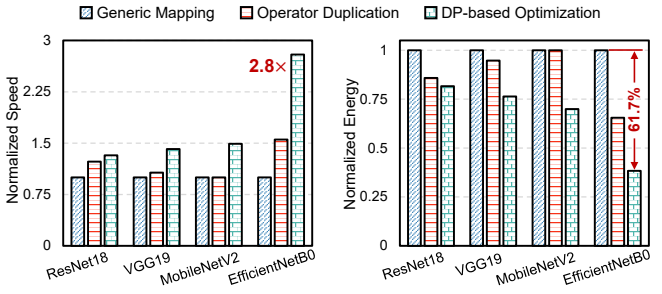


Fig. 5. Normalized speed and energy comparison of different compilation optimization strategies across DNN models.

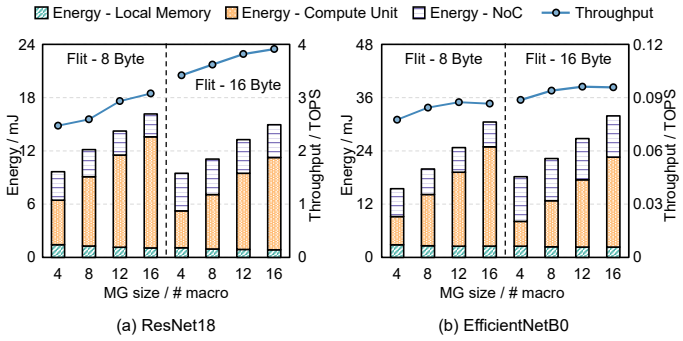


Fig. 6. Energy consumption breakdown and throughput across architectures with different MG sizes and NoC link bandwidth.

operator duplication technique from CIM-MLC [19], which first partitions CG to fit the limited capacity then attempts to utilize vacant resources through weight duplication. All evaluations use the default architecture configuration described in Tab. I to isolate the impact of compilation strategies.

As illustrated in Fig. 5, our DP-based partitioning and optimization method demonstrates significant performance improvements, achieving up to 2.8× speedup and 61.7% energy reduction compared to the baseline approaches. The benefits are particularly pronounced for compact models like MobileNetV2 and EfficientNetB0, where the conventional partition method proves less effective due to their smaller weight footprints, which leaves fewer unoccupied cores within such partitioned execution stages for duplication opportunities. This highlights the effectiveness of our DP-based approach in finding optimal partitioning and mapping schemes that maximize performance while respecting SRAM capacity constraints.

C. Architectural Configuration Exploration

The efficiency of digital CIM architectures hinges on the careful balance between computational and data movement capabilities. We explore this trade-off through two critical design parameters: MG size scaling and NoC link bandwidth (flit size per cycle) configuration. Fig. 6 presents the energy breakdown and throughput analysis across architectural configurations for both compute-intensive and compact models compiled with the generic mapping strategy.

For ResNet18, increasing MG size consistently improves throughput at the cost of moderately higher energy consumption, with compute unit energy remaining its dominant component. Doubling the communication bandwidth also boosts inter-layer pipeline throughput by up to 39.6%. In contrast, EfficientNetB0 shows different scaling characteristics. Its lower resource requirements mean that increasing the number of macros per group yields only modest throughput gains, while higher NoC bandwidth introduces substantial data transfer overhead without commensurate performance benefits, consuming up to 55.4% of total energy consumption when MG size is 4. These distinct scaling behaviors across models highlight the importance of early-stage architectural exploration, demonstrating the value of CIMFlow as a systematic design and evaluation framework.

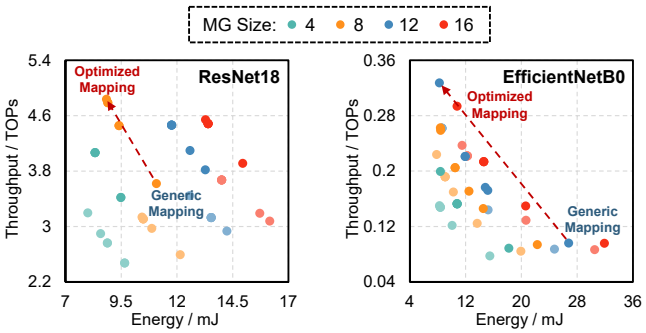


Fig. 7. Software/Hardware design space categorized by MG size. Light/dark shades indicate 8/16-byte NoC flit sizes.

To provide insights into the interaction between software and hardware design choices, we further compare different compilation strategies across these hardware configurations. As shown in Fig. 7, while hardware configurations significantly impact the achievable performance envelope, the performance differences between hardware configurations can be significantly reduced or even reversed through careful compilation optimizations. These observations highlight why an integrated hardware-software co-design approach is essential for digital CIM architectures, as isolated exploration of either space would overlook crucial optimization opportunities.

V. Conclusion and Future Work

This paper presents CIMFlow, an integrated framework that enables systematic design and evaluation of digital CIM architectures. Through a flexible multi-level ISA design and advanced compilation strategies, CIMFlow effectively addresses key challenges in digital CIM implementation, particularly the SRAM capacity limitations. Our experimental results demonstrate the capabilities of our proposed compilation optimizations, achieving up to 2.8× speedup and 61.7% energy reduction. The highly customizable and extensible nature of CIMFlow enables systematic design space exploration, providing crucial insights for the software and hardware design of digital CIM. In the future, we will keep on expanding the framework to support emerging DNN operators and developing automated design space exploration techniques. We believe CIMFlow represents a significant step toward making digital CIM a practical solution for next-generation AI accelerators.

REFERENCES

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits (JSSC)*, vol. 52, no. 1, pp. 127–138, 2016.

[2] J. Yang, W. Fu, X. Cheng, X. Ye, P. Dai, and W. Zhao, "S^2 Engine: A novel systolic architecture for sparse convolutional neural networks," *IEEE Transactions on Computers (TC)*, vol. 71, no. 6, pp. 1440–1452, 2021.

[3] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and memory wall," *IEEE Micro*, vol. 44, no. 3, pp. 33–39, 2024.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[5] Y. Zhao, J. Yang, B. Li, X. Cheng, X. Ye, X. Wang, X. Jia, Z. Wang, Y. Zhang, and W. Zhao, "NAND-SPIN-based processing-in-MRAM architecture for convolutional neural network acceleration," *Science China Information Sciences (SCIS)*, vol. 66, no. 4, p. 142401, 2023.

[6] C. Duan, J. Yang, X. He, Y. Qi, Y. Wang, Y. Wang, Z. He, B. Yan, X. Wang, X. Jia *et al.*, "DDC-PIM: Efficient algorithm/architecture co-design for doubling data capacity of SRAM-based processing-in-memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 3, pp. 906–918, 2023.

[7] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[8] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems (ASPLOS)*, 2019, pp. 715–731.

[9] J.-W. Su, Y.-C. Chou, R. Liu, T.-W. Liu, P.-J. Lu, P.-C. Wu, Y.-L. Chung, L.-Y. Hung, J.-S. Ren, T. Pan *et al.*, "16.3 a 28nm 384kb 6T-SRAM computation-in-memory macro with 8b precision for AI edge chips," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 250–252.

[10] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori *et al.*, "16.4 an 89TOPS/W and 16.3 TOPS/mm 2 all-digital SRAM-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 252–254.

[11] B. Yan, J.-L. Hsu, P.-C. Yu, C.-C. Lee, Y. Zhang, W. Yue, G. Mei, Y. Yang, Y. Yang, H. Li *et al.*, "A 1.041-Mb/MM 2 27.38-TOPS/W signed-int8 dynamic-logic-based ADC-less SRAM compute-in-memory macro in 28nm with reconfigurable bitwise operation for AI and embedded applications," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 188–190.

[12] C. Duan, J. Yang, Y. Wang, Y. Wang, Y. Qi, X. He, B. Yan, X. Wang, X. Jia, and W. Zhao, "Towards efficient sram-pim architecture design by exploiting unstructured bit-level sparsity," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[13] Q. Zheng, X. Li, Y. Guan, Z. Wang, Y. Cai, Y. Chen, G. Sun, and R. Huang, "PIMulator-NN: An event-driven, cross-level simulation framework for processing-in-memory-based neural network accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 12, pp. 5464–5475, 2022.

[14] H. Liu, J. Xu, X. Liao, H. Jin, Y. Zhang, and F. Mao, "A simulation framework for memristor-based heterogeneous computing architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 12, pp. 5476–5488, 2022.

[15] J. Lee, A. Lu, W. Li, and S. Yu, "Neurosim v1. 4: Extending technology support for digital compute-in-memory toward 1nm node," *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 71, no. 4, pp. 1733–1744, 2024.

[16] Z. Zhu, H. Sun, T. Xie, Y. Zhu, G. Dai, L. Xia, D. Niu, X. Chen, X. S. Hu, Y. Cao *et al.*, "MNSIM 2.0: A behavior-level modeling tool for processing-in-memory architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 11, pp. 4112–4125, 2023.

[17] A. Siemieniuk, L. Chelini, A. A. Khan, J. Castrillon, A. Drebes, H. Corporaal, T. Grosser, and M. Kong, "OCC: An automated end-to-end machine learning optimizing compiler for computing-in-memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 6, pp. 1674–1686, 2021.

[18] X. Sun, X. Wang, W. Li, L. Wang, Y. Han, and X. Chen, "PIMCOMP: A universal compilation framework for crossbar-based PIM DNN accelerators," in *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[19] S. Qu, S. Zhao, B. Li, Y. He, X. Cai, L. Zhang, and Y. Wang, "CIM-MLC: A multi-level compilation stack for computing-in-memory accelerators," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024, pp. 185–200.

[20] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of SRAM-based computing-in-memory for AI edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 68, no. 5, pp. 1773–1786, 2021.

[21] J. Chen, F. Tu, K. Shao, F. Tian, X. Huo, C.-Y. Tsui, and K.-T. Cheng, "AutoDCIM: An automated digital CIM compiler," in *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[22] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie, and S. Yin, "ReDCIM: Reconfigurable digital computing-in-memory processor with unified FP/INT pipeline for cloud AI acceleration," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 58, no. 1, pp. 243–255, 2022.

[23] G. Desoli, N. Chawla, T. Boesch, M. Avodhyawasi, H. Rawat, H. Chawla, V. Abhijith, P. Zambotti, A. Sharma, C. Cappetta *et al.*, "16.7 A 40-310TOPS/W SRAM-based all-digital up to 4b in-memory computing multi-tiled NN accelerator in FD-SOI 18nm for deep-learning edge applications," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 260–262.

[24] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 2–14.

[25] *IEEE Standard for Standard SystemC® Language Reference Manual*, IEEE Std., 2023, IEEE Std 1666-2023 (Revision of IEEE Std 1666-2011).

[26] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proceedings of the 26th IEEE international conference on application-specific systems, architectures and processors (ASAP)*. IEEE, 2015, pp. 162–163.

GPS: GNN-Based Two-Stage Pre-Scheduling Loop Mapping Method on CGRAs

Mingyang Kou[1], Weiqing Ji[1], Shouyi Yin[2] and Hailong Yao[13*]
1. University of Science and Technology Beijing
2. Tsinghua University
3. Key Laboratory of Advanced Materials and Devices for Post-Moore Chips, Ministry of Education
Beijing, China

ABSTRACT

Coarse-grained reconfigurable architecture (CGRA) has emerged as a promising solution for accelerating computationally intensive applications, particularly in the field of artificial intelligence. One of the primary challenges for CGRA compilers is generating effective mapping results for complex applications within a limited timeframe. This paper presents an enhanced pre-scheduling method that integrates Integer Linear Programming (ILP) and Graph Neural Networks (GNN), along with a corresponding two-stage mapping approach. This combination significantly reduces the search space and accelerates the solution process for mapping problems. Experimental results demonstrate performance improvements ranging from 29.4% to 406.7%, along with compilation time reductions of up to 1106.8× compared to existing compilation techniques, as well as excellent scalability.

KEYWORDS

CGRA, Loop mapping, Graph Neural Network, Scheduling

1 INTRODUCTION

The rapid advancement of technologies, such as large language models and cloud computing, has led to an increasing demand for computational power. As Moore's Law approaches its limits and the need for substantial computational resources continues to grow, the design of novel hardware architectures and their corresponding programming software systems has become essential for enhancing computer performance [1]. Reconfigurable architectures, particularly Coarse-Grained Reconfigurable Architectures (CGRAs) [2–9], combine the flexibility of general-purpose processors, such as CPUs and GPUs, with the high performance of Application-Specific Integrated Circuits (ASICs). This combination positions CGRAs as a promising approach to overcoming the computational power bottlenecks of traditional chips and enhancing computing efficiency.

As illustrated in Figure 1, a typical CGRA consists of a host controller, memory, multiple input/output buffers, and several spatially interconnected Processing Element Arrays (PEAs). Each PEA is an array of processing elements (PEs) that are spatially interconnected. Each PE serves as the fundamental unit for computation and data processing within the CGRA, responsible for executing basic coarse-grained computational tasks. A PE is composed of several input multiplexers (MUX), output functional units (FUs), a Local Register File (LRF), and an Arithmetic Logic Unit (ALU).

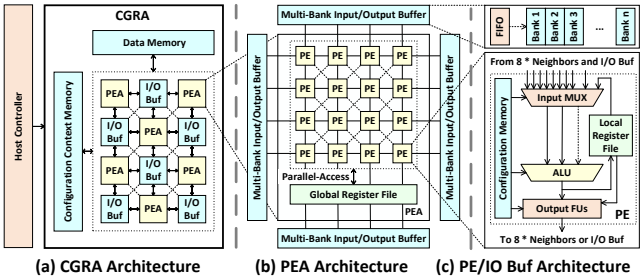


Figure 1: Architecture of a CGRA with 4×4 PEAs. (a) CGRA, (b) PEA, (c) PE and I/O Buffer.

A high-performance compiler is essential for fully leveraging the potential of CGRAs. Code generation is a critical stage within the compilation process, as it significantly impacts the performance of fine-grained operators. The loop mapping problem represents a core challenge in backend code generation. Solving this problem often incurs substantial computational and storage overhead, making it the most time-consuming aspect of the CGRA compilation process. CGRA compilers have explored various methods to effectively tackle the loop mapping problem [2–9], which is classified as an NP-hard problem [11]. However, as software applications scale, the complexity of mapping onto CGRAs grows exponentially, resulting in prohibitively long compilation times.

To accelerate the solution of the loop mapping problem, particularly for large and complex applications, this paper proposes GPS, a two-stage loop mapping method that integrates graph neural networks (GNN) and pre-scheduling techniques. The primary contributions of this paper are as follows.

1. **Two-Stage Mapping Method:** We propose a two-stage mapping method for loop mapping, which leverages Graph Isomorphism Networks [12] and pre-scheduling techniques to narrow the search space in solving mapping problems.
2. **Graph Isomorphism Model and Optimization:** A graph isomorphism model is introduced to formulate the loop mapping problem. Additionally, we present a mapping algorithm based on pattern graph matching to tackle this challenge. Pre-mapping optimizations and polyhedral-based loop transformations are employed to enhance the execution efficiency of loops on CGRAs.
3. **GPS Algorithm:** We introduce a fast and effective loop mapping algorithm, named GPS, designed for heterogeneous CGRAs. This algorithm optimizes both solution quality and computational efficiency. Experimental results demonstrate that GPS achieves significant performance improvements,

*Corresponding authors: Hailong Yao (hailongyao@ustb.edu.cn). This work was supported by the Key Program of National Natural Science Foundation of China (No. 62034005) and the China Postdoctoral Science Foundation under Grant Number 2024M750183.

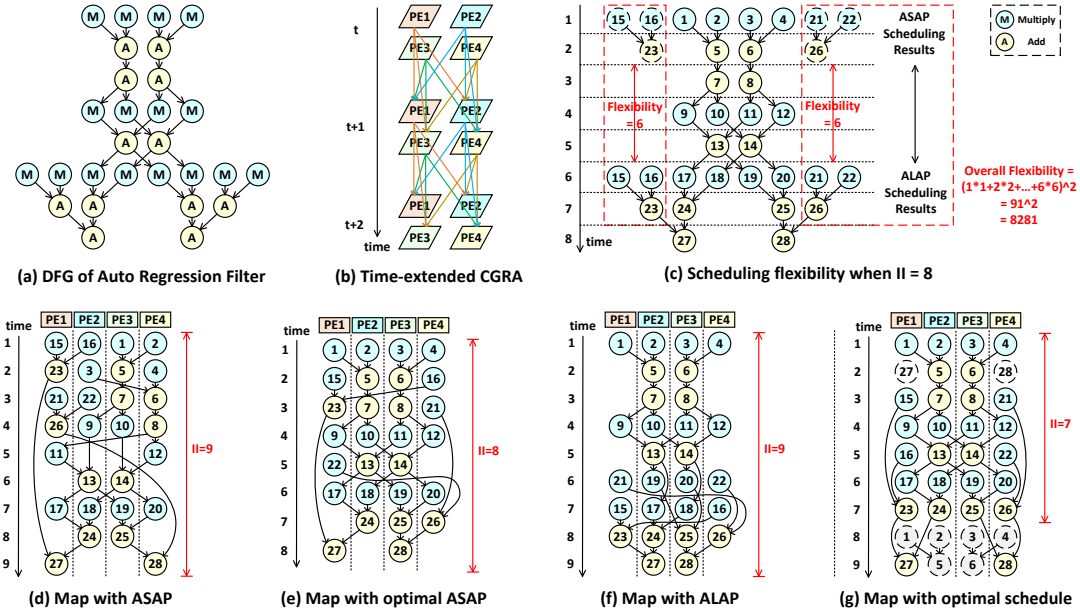


Figure 2: Motivation of this work. (a) DFG of testcase Auto Regression Filter in EXPRESS benchmark [10]. (b) Time-extended CGRA of a 2×2 2-D mesh-like PEA. (c) Scheduling flexibility when II = 8. (d) Mapping results with ASAP schedule. (e) Mapping results with optimal ASAP schedule. (f) Mapping results with ALAP schedule. (g) Mapping results with optimal schedule.

ranging from 29.4% to 406.7%, along with up to a 1106.8× reduction in compilation time compared to existing compilation techniques.

2 BACKGROUND AND MOTIVATION

CGRAs are highly flexible architectures that can be dynamically reconfigured to support various computational tasks, including multimedia and AI applications [6–9]. CGRA compilers play a crucial role in harnessing the full potential of these architectures, enabling them to deliver high performance and flexibility across applications. Due to their array-based, highly parallel architecture, CGRAs are particularly well-suited for executing repetitive tasks, such as loops. Notably, loops often account for a significant portion of the overhead during program execution, and loop mapping typically represents the most time-consuming aspect of the CGRA compilation process. Consequently, CGRA compilers employ various methods to efficiently address the loop mapping problem, including heuristic approaches [4–7, 9] and optimization-based methods [8].

During the loop mapping stage, the input consists of the loop kernel of an application represented as a Data Flow Graph (DFG). In the DFG, nodes represent element-wise operations such as ADD or MUL, while edges depict dependencies between these operations. The goal of loop mapping is to optimize the performance of the loop kernel on CGRAs by minimizing the initiation interval (II) of loop iterations. The II represents the minimum number of clock cycles required to execute each iteration of the loop, thereby indicating the program's performance when executed on CGRAs.

As the size of the DFG increases, the complexity of the loop mapping problem grows exponentially, presenting significant challenges for effective problem-solving. For instance, the Auto Regression Filter (*arf*) case in the EXPRESS benchmark [10], illustrated in Figure 2(a), consists of 28 nodes and 30 edges. Solving the mapping problem for this example with an ILP-based approach [8] on a 2×2 Processing Element Array (PEA) as depicted in Figure 2(b) necessitates constructing a Modulo Routing Resource Graph (MRRG) containing 812 nodes and 1120 edges to achieve an optimal II of 7. For more complex DFGs, such as the *jpeg_fdct* case with 134 nodes and 169 edges, the complexity of the MRRG increases exponentially, posing a significant challenge for direct solutions using ILP-based methods. In the case of heuristic approaches, solving the mapping for the *arf* example in Figure 2(a) requires addressing the maximum common subgraph problem for two graphs, each with 28 nodes. This translates to solving an NP-hard maximum clique problem with 784 nodes [6], or employing heuristic methods like simulated annealing [7], which may struggle to find optimal solutions.

For complex loop mapping problems, we have found that employing a pre-scheduling approach to decompose the mapping problem into two subproblems can effectively narrow the search space. This pre-scheduling method involves initially allocating cycles for each operation in the DFG, followed by precise graph matching. In contrast, traditional methods require solving the mapping problem while simultaneously determining the execution start cycle for each operation. Figure 2(c) illustrates the search space for scheduling problems when using traditional ILP methods or heuristic approaches for mapping at an II of 8, highlighting scheduling flexibility. Notably, nodes 15, 16, 23, and 21, 22, 26 exhibit significant flexibility. For instance, nodes 15 and 16 can be executed in the first cycle, followed by 23, or they can be executed before 27 at the end. The flexibility of nodes 15 and 16 is also adjusted based on the scheduling of node 23. Consequently, the total number of valid scheduling options for all nodes is given by $(\sum_{i=1}^{6}(7-i)^2)^2 = 91^2 = 8,281$. If

scheduling can be completed in advance, the size of the search space would reduce to approximately $\frac{1}{8281}$ of its original size, significantly optimizing the mapping process.

Previous research has explored two-stage solving processes that involve scheduling prior to mapping, such as the As-Soon-As-Possible (ASAP) scheduling method [6], diagonal-based scheduling [13], and random scheduling [14]. However, these approaches often struggle to produce optimal scheduling results, leading to larger II values and negatively affecting the execution performance of loop tasks on CGRAs. Figures 2(d) and 2(e) illustrate the scheduling results obtained using the ASAP scheduling method for mapping the DFG shown in Figure 2(a) onto the 2×2 PEA depicted in Figure 2(b). ASAP scheduling refers to scheduling nodes as early as possible in feasible positions, while ALAP scheduling, shown in Figure 2(f), involves scheduling nodes to the latest feasible cycles. Figure 2(g) presents the mapping results obtained using the optimal scheduling scheme, achieving an optimal II of 7. The results from Figures 2(d) and 2(e) indicate that, despite both using the ASAP scheduling method, the scheduling and mapping outcomes differ due to resource constraints, ultimately resulting in various II values. Furthermore, the results presented in Figures 2(d-g) demonstrate that regardless of the scheduling methods employed—whether ASAP, ALAP, or random scheduling—it remains challenging to achieve optimal scheduling or mapping results corresponding to the optimal II. Therefore, in this paper, we propose a scheduling method based on ILP and GNN to obtain the best possible scheduling results within a short timeframe. This scheduling approach can also be integrated with existing mapping methods to narrow the search space of the loop mapping problem and accelerate the problem-solving process.

3 PROBLEM FORMULATION

1) DFG: A DFG, denoted as $G = (V, E)$, typically represents an iteration of a loop. Here, V is the set of nodes, and for every $v \in V$, v represents an operation on the Processing Element (PE). E is the set of edges, where for every edge $e = (v_i, v_j) \in E$ (with $v_i, v_j \in V$), it indicates a data dependency from operation i to operation j.

2) TEC: A Time-Extended CGRA (TEC) refers to a CGRA extended by II times on time axis, represented as $G' = (V', E')$. Here, V' and E' are the sets of nodes and edges, respectively. PE_i^t denotes the i-th PE at modulo cycle t, while $e = (PE_i^t, PE_j^{t+1}) \in E$ represents a datapath from the i-th PE at cycle t to the j-th PE at cycle $t+1$.

3) Modulo Schedule and II: Modulo scheduling is a software pipelining technique that parallelizes different iterations of a loop, creating a repeatedly executed block consisting of a complete set of operations from the inner loop body, with each operation potentially coming from different iterations. The duration of this block represents the II between sequential iterations, reflecting the performance of the scheduled loop on the CGRA. After modulo scheduling of the DFG, a Modulo Data Flow Graph (MDFG) is generated.

4) Resource-Constrained Scheduling Problem: Given the DFG $G = (V, E, F)$ and the CGRA $G' = (V', E', F')$ (where F and F' represent the types of operations in the DFG and the functions of the PEs), along with the II, the problem of finding an optimal schedule can be formulated as a combinatorial optimization problem.

- *Decision Variables:* $t_i = t \in \mathbb{N}^*, i \in \mathbb{N}^*, 1 \le i \le |G|$, indicating that the i-th DFG node is scheduled to start at cycle t.

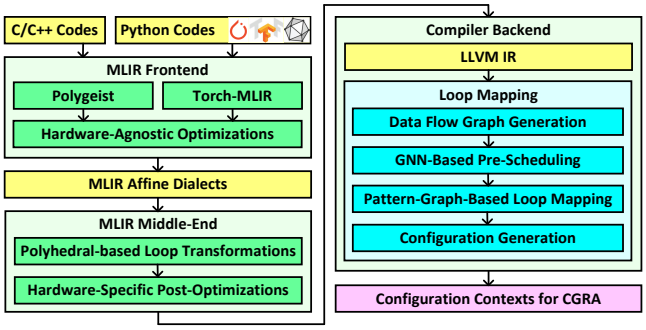


Figure 3: Overview of the compilation process of GPS.

The II denotes the initiation interval, and a *schedule* can be defined as $S = (t_1, t_2, \ldots, t_n, II)$. $t_{mi} = t_i \mod II$ represents the modulo start cycle at which operation i is scheduled.

- *Constraints:*
  - i) *Freedom Constraint:* For each operation, its scheduled cycle should be confined between ASAP and ALAP times. For all $v_i \in V, t_{si} \le t_i \le t_{li}$. Here, $t_{si} = \text{ASAP}(v_i)$ denotes the ASAP time of the i-th DFG node, while $t_{li} = \text{ALAP}(v_i)$ denotes the ALAP time of the i-th DFG node.
  - ii) *Resource Constraint:* In every modulo cycle, the number of operations must be less than or equal to the number of PEs with the corresponding functions. For all $t \in [0, II), t \in \mathbb{N}, \{f_i | f_i \in F, t_{mi} = t\} \subset F'$.
- *Optimization Objective:* Minimize the II.

This resource-constrained scheduling problem can be solved for optimal solutions using Integer Linear Programming methods based on the linear equations outlined above.

5) Mapping Problem: Given a scheduled MDFG $G = (V, E)$ and a TEC $T = (V', E')$, the objective is to find a mapping function $f : V \to V'$ that satisfies the following conditions:

- *Graph Isomorphism:* G is isomorphic to a subgraph of T, which indicates all nodes and edges in the DFG are mapped, i.e., $\forall v_i, v_j \in V, f(v_i), f(v_j) \in V', f(v_i) \ne f(v_j)$, and $\forall e = (u, v) \in E, u, v \in V, \exists e_t = (f(u), f(v)) \in E'$.
- *Optimization Objective:* Minimize the II while ensuring that G is isomorphic to a subgraph of T.

6) GNN Model: GNNs [12] possess significant representational power for complex graphs, making them suitable for extracting intricate relationships between operations in DFGs. In this paper, we design a GNN model to obtain the optimal schedule for the DFG. Denoting the input data as $\mathcal{D}$, our model can be formulated as:

$$\mathcal{M}(i) = MLP(GIN_EMBED(\mathcal{D}_i)) \tag{1}$$

Details of our loss function will be introduced in Section 4. We adopt an ILP-based scheduling algorithm to generate the training labels for our GNN model. The output of the GNN model, denoted as $\mathcal{M}$, is utilized by the list scheduling algorithm as the priority function, which results in enhanced scheduling solutions.

4 THE GPS METHOD

To facilitate efficient mapping on CGRAs, we have developed GPS, a GNN-based pre-scheduling loop mapping method. Figure 3

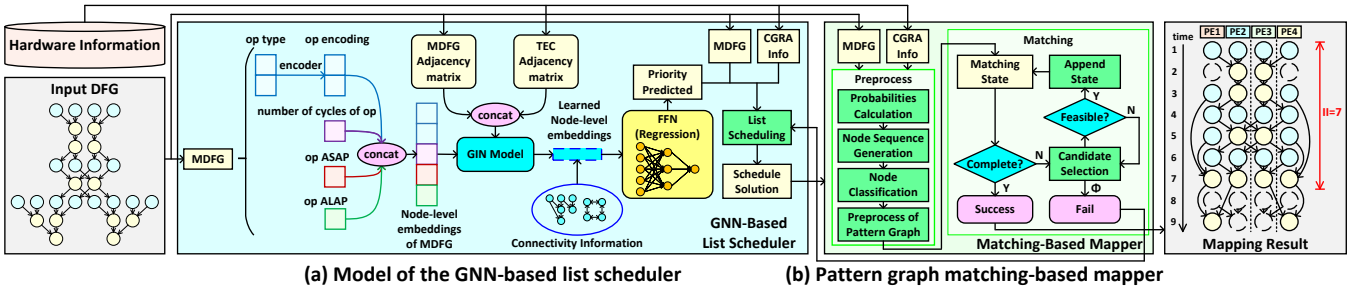


Figure 4: Overview of the two-stage mapper. (a) Model of the GNN-based scheduler. (b) Pattern graph matching-based mapper.

Algorithm 1 GNN-based pre-scheduling algorithm.

Input: Trained network model $\mathcal{M}$, DFG $G = (V, E, F)$, CGRA $G' = (V', E', F')$, delay array d, current initiation interval II
Output: Generated schedule $S_{\mathcal{M}}$

1: $\mathcal{D} \leftarrow encodeData(G), \mathcal{P} \leftarrow runModel(\mathcal{M}, \mathcal{D})$
2: $V_F \leftarrow \phi, V_R \leftarrow \phi, V_U \leftarrow V, S_{\mathcal{M}} \leftarrow \phi, Time \leftarrow 0$
3: **for** $v_i \in V$ **do**
4: **if** $v_i.inDegree() = 0$ **then**
5: $V_R \leftarrow V_R \cup v_i, V_U \leftarrow V_U \setminus v_i$
6: **end if**
7: **end for**
8: **while** $|V_F| < |V|$ **and** $Time < II$ **do**
9: **for** $u_j \in V'$ **do**
10: $M \leftarrow MAX_FLOAT, v_h \leftarrow \phi$
11: **for** v_i in V_R **do**
12: **if** $\mathcal{P}(v_i) < M$ **and** $v_i.canUse(u_j)$ **then**
13: $M \leftarrow \mathcal{P}(v_i), v_h \leftarrow v_i$
14: **end if**
15: **end for**
16: **if** $v_h = \phi$ **or** $u_j.unavailable(Time)$ **then**
17: **continue**
18: **end if**
19: $S_{\mathcal{M}}.insert(v_h, u_j, Time), V_R \leftarrow V_R \setminus v_h, V_F \leftarrow V_F \cup v_h$
20: $u_j.setUnavailable(Time, Time + d_{u_j} - 1)$
21: **for** v_i in V_R **do**
22: **if** $(v_h, v_i) \in E$ **then**
23: $E \leftarrow E \setminus (v_h, v_i)$
24: **if** $v_i \in V_U$ **and** $v_i.inDegree() = 0$ **then**
25: $V_R \leftarrow V_R \cup v_i, V_U \leftarrow V_U \setminus v_i$
26: **end if**
27: **end if**
28: **end for**
29: **end for**
30: $Time \leftarrow Time + 1$
31: **end while**
32: **if** $Time < II$ **and** $|V_F| = |V|$ **then**
33: **return** $S_{\mathcal{M}}$
34: **end if**
35: **return failed**

illustrates the overall compilation process of GPS, which consists of the compiler frontend, middle-end, and backend. Our compiler framework is is implemented based on ML-CGRA [3], MLIR [15], Polygeist [16], and Torch-MLIR [17]. The compiler frontend reads source codes written in Python, C, or C++, performs hardware-agnostic optimizations—including loop boundary and conditional statement optimizations—and generates MLIR Affine Dialects. The middle-end comprises two key steps: 1) polyhedral-based loop transformations and 2) hardware-specific post-optimizations. Once optimizations are complete in the middle-end, the backend takes the input LLVM Intermediate Representation (IR) and generates the DFG. Finally, the backend performs loop mapping to produce executable CGRA configuration contexts. The complete mapping process consists of two stages: pre-scheduling and loop mapping. During the pre-scheduling stage, time steps for each operation in the DFG are allocated to reduce the search space for loop mapping. A GNN-based deep learning framework is proposed to generate high-quality scheduling solutions.

Algorithm 2 Mapper Based on Pattern Graph Matching

Input: DFG D and CGRA C
Output: Mapping result M or Failed

1: $II \leftarrow$ CalculateMII(D, C);
2: **while** $II \leq$ Threshold **do**
3: $T \leftarrow$ GenerateTEC(C, II);
4: $D' \leftarrow$ PreSchedule(D, C, II);
5: $M \leftarrow \emptyset$
6: $P_f \leftarrow$ ComputeProbilities(T, D')
7: $N_{D'} \leftarrow$ GenerateNodeSequence(D', P_f)
8: ClassifyNodes(D', T)
9: (s_0, Parent) $\leftarrow$ PreprocessPatternGraph(D', $N_{D'}$)
10: Match(s_0, T, D', $N_{D'}$, Parent, M) ▷ $M \leftarrow$ Maximum subgraph of D' and T
11: **if** $|V_M| = |V_{D'}|$ **then**
12: **return** M;
13: **end if**
14: $II \leftarrow II + 1$;
15: **end while**
16: **return** Failed;

4.1 GNN-based Pre-scheduling

Algorithm 1 provides an overview of the proposed GNN-based pre-scheduling algorithm, which builds upon the list scheduling algorithm [14] and our neural network models. This algorithm divides the node set V into three subsets: finished nodes V_F, ready nodes V_R, and unscheduled nodes V_U. Given the node priority function $\mathcal{P}$, where $\mathcal{P}(v_i) < \mathcal{P}(v_j)$ for $v_i, v_j \in V_R$ indicates that operation v_i has a higher priority than operation v_j, we use the trained network model $\mathcal{M}$ as the priority function $\mathcal{P}$ (line 1) and continue the process until every node in the DFG is successfully scheduled (line 8). At each step, we select the node $v_h \in V_R$ to schedule, such that $\mathcal{P}(v_h) = \max_{v_r \in V_R} \mathcal{P}(v_r)$ (line 13). In each cycle, we attempt to schedule every available PE using the node v_h with the highest priority (line 16). After selecting node v_h from the ready set V_R, we update the schedule $S_{\mathcal{M}}$ and the sets V_R and V_F (line 19). We then remove all outgoing edges (v_i, v_h) from v_h (line 23). If the in-degree of v_i equals 0, we add v_i to the ready set V_R (line 25). Ultimately, the scheduling solution at II is obtained in $S_{\mathcal{M}}$.

A better priority function in the list scheduling algorithm leads to improved scheduling solutions. An ILP formulation has been proposed as an optimal solution for the scheduling problem [18]. However, the exponential runtime of ILP renders it impractical and inefficient for larger instances. Therefore, employing a GNN model to approximate the optimal solution generated by ILP presents a feasible and efficient alternative. The output of the GNN-based deep learning model can serve as the heuristic function for list scheduling. Figure 4(a) illustrates the architecture of the proposed model, which consists of a Graph Isomorphism Network (GIN) [12] and a feed-forward network (FFN). The GIN model is responsible for learning the embeddings of each operation based on its features

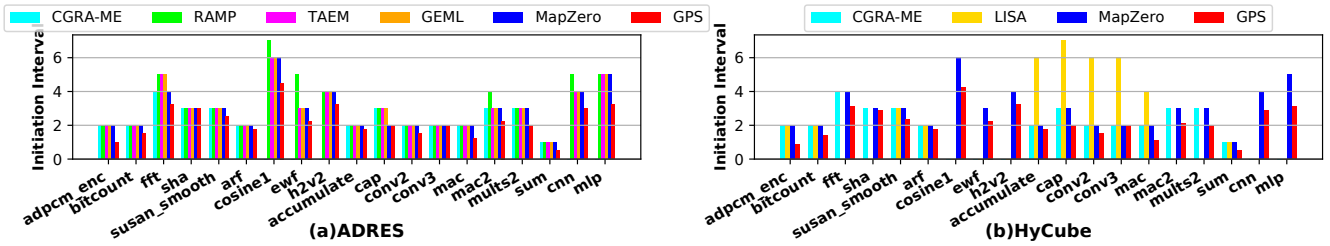


Figure 5: II of loops when mapped to a 4×4 CGRA of different architectures. (a) ADRES, (b) HyCube.

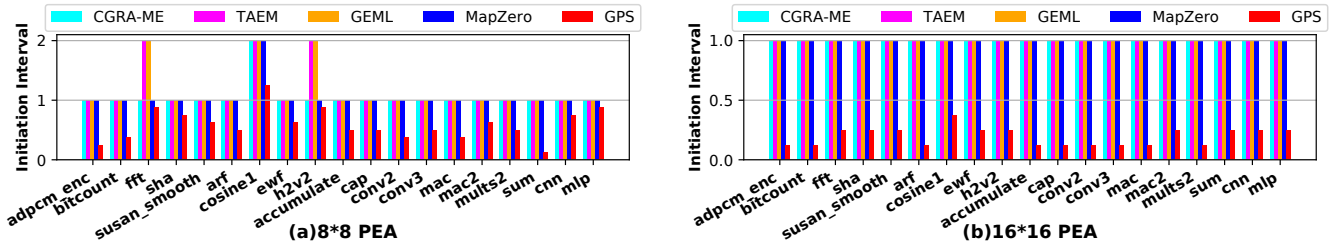


Figure 6: II of loops when mapped to a 8×8 or 16×16 CGRA of ADRES architecture.

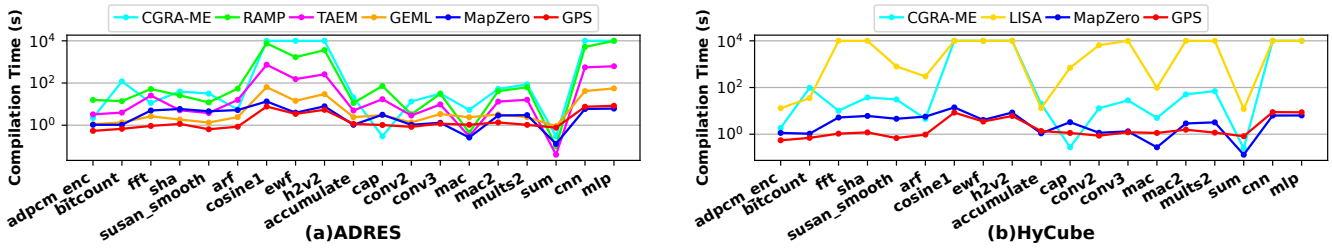


Figure 7: Compilation time when loops are mapped to a 4×4 CGRA of different architectures. (a) ADRES, (b) HyCube.

and the graph structure. The FFN model is utilized for regression tasks, specifically to predict the priorities of operations. The input to the model is $\mathcal{D} = \{g_i, p_i, l_i\}$, where g_i denotes the graph structures of the MDFG and TEC calculated according to the II, p_i represents operation properties (such as ASAP and ALAP), and l_i denotes the training label generated by ILP. We utilized the ILP method to generate a CGRA-related scheduling dataset to train our model. Our loss function is defined as the following rank loss function, where $\mathcal{M}(i) = \mathcal{M}(g_i, p_i)$ and $\mathcal{M}(j) = \mathcal{M}(g_j, p_j)$ represent the outputs of the GIN model for two test cases, respectively.

$$\mathcal{L} = \sum_{i,j} \log\left(1 + e^{-\operatorname{sign}(l_i - l_j)(\mathcal{M}(i) - \mathcal{M}(j))}\right). \tag{2}$$

4.2 Pattern Graph Matching-Based Mapping

After scheduling the DFG, the next step is to map the loop DFG onto the CGRA. To accelerate the mapping process, we formulate the mapping problem as a subgraph isomorphism problem and employ the pattern graph matching algorithm based on VF3 algorithm [19] for efficient resolution. If the MDFG is isomorphic to a subgraph of the TEC, it indicates that the current mapping scheme is feasible. The complete mapping procedure is depicted in Algorithm 2. We initialize the II to the minimum possible value under recursive and resource constraints. Subsequently, we use the II to generate the TEC and perform a pre-scheduling of the DFG. The mapping of the DFG to the TEC is generated using our pattern graph matching algorithm. If the mapping fails at the current II, we increment the II by 1 until a successful mapping is achieved.

Our mapping algorithm is based on pattern graph matching [19], and Figure 4(b) illustrates its workflow. The algorithm progressively narrows the search space by executing the following five steps, incorporating constraint propagation and pruning:

- *Probability Calculation:* For each node in the pattern graph, its matching probability is assessed by calculating the ratio of nodes in the target graph that share the same label and have greater in-degrees and out-degrees. This serves as a critical criterion for node sorting.
- *Node Sequence Generation:* Nodes are sorted based on their probability and degree to prioritize the search for strongly constrained nodes. This helps reduce the width of the search tree in the early stages of the search.
- *Node Classification:* Candidate nodes are filtered using set-level connectivity relations (e.g., predecessor and successor sets) and node labels (operation type and functional unit type for CGRA mapping) to determine the candidate set for the next matching step.
- *Preprocessing of Pattern Graph:* The neighborhood of candidate nodes are explored to facilitate subsequent matching.

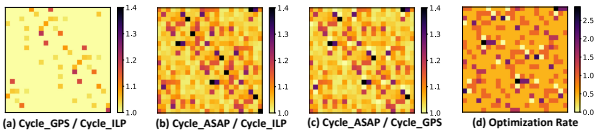


Figure 8: Ablation study on our pre-scheduling method. (a) T_{GPS}/T_{ILP}, (b) T_{ASAP}/T_{ILP}, (c) T_{ASAP}/T_{GPS}, (d) Optimization rate of searching space $log_{10}\frac{Naive}{GPS}$.

- *Matching:* Starting from an empty mapping state, new matching pairs are gradually searched until a complete mapping solution is found. In this step, through a comprehensive analysis of the relationships and connectivity between sets, the algorithm proactively identifies mapping pairs that can be matched in the current step but may not generate new mapping pairs in subsequent iterations.

4.3 Polyhedral-Based Loop Transformations

Based on the OpenCGRA [2] and Polygeist [16] frameworks, we performed polyhedral-based loop transformations prior to mapping to enhance the execution performance of programs on the CGRA. The loop transformations we applied include loop tiling, loop fusion, loop interchange, loop fission, loop reversal, and loop unrolling. By applying these transformations, we optimized the loops in terms of storage efficiency, operation parallelism, and tensor shapes. Additionally, we executed hardware-agnostic optimizations such as constant folding and conditional statement optimization, as well as hardware-specific optimizations based on Single Static Assignment (SSA), to further improve loop performance. When the related work is published, GPS will be open-sourced at Github.

5 EXPERIMENTS

In this section, we evaluate the solution quality, compilation time overhead, and scalability of our CGRA mapper, GPS, in comparison with state-of-the-art CGRA mappers: CGRA-ME [20] (optimized version based on ILP [8]), RAMP [6], TAEM [4], GEML [5], LISA [7], and MapZero [9]. We conducted experiments on two typical architectures: ADRES [21] and HyCube [22]. The sizes of the PEA are configurable, including 4×4, 8×8, and 16×16. To evaluate GPS with a diverse set of loop benchmarks, we selected loops with varying sizes, dependencies, and operator types from MiBench [23], EXPRESS benchmarks [10], and the benchmarks used in CGRA-ME [20]. The experiments were conducted on a server equipped with a 128-Core Intel(R) Xeon(R) Gold 5218 CPU running at 2.30 GHz and 256 GB of memory, along with an Nvidia Tesla A100 GPU.

5.1 Mapping Results and Scalability

Figure 5 compares the performance of various compilers across their supported architectures, including CGRA-ME [20], RAMP [6], TAEM [4], GEML [5], LISA [7], MapZero [9], and the proposed GPS. Since GPS incorporates loop transformations, the II obtained by GPS is normalized based on the average II of 16 loop iterations. From the figure, on a 4×4 ADRES architecture, GPS demonstrates an average performance improvement of 29.5% over CGRA-ME, 45.9% over RAMP, 34.1% over both TAEM and GEML, and 29.4% over MapZero. On a 4×4 HyCube architecture, GPS shows a performance improvement of 34.0% over CGRA-ME, 155.8% over LISA, and 36.2% over MapZero.

GPS also exhibits better scalability for CGRAs of varying sizes. As shown in Figure 6, GPS achieves improved performance with larger PEA. On an 8×8 ADRES architecture, GPS demonstrates an average performance improvement of 77.8% over CGRA-ME and MapZero, and 95.6% over TAEM and GEML. On a 16×16 ADRES architecture, GPS achieves an average performance improvement of 406.7% over CGRA-ME, TAEM, GEML, and MapZero. These experimental results indicate that GPS exhibits strong scalability across different CGRA architectures and varying scales of CGRAs.

5.2 Compilation Time

Figure 7 compares the compilation times of various compilers, including CGRA-ME [20], RAMP [6], TAEM [4], GEML [5], LISA [7], MapZero [9], and GPS. The experimental results demonstrate that GPS achieves an average compilation speed improvement of 1106.8× over CGRA-ME, 623.9× over RAMP, 52.7× over TAEM, 4.17× over GEML, and 59.8% over MapZero. For instance, the compilation time of a loop in *arf* of MapZero is about 5.2s, while it takes only 0.8s for GPS to obtain a better solution. However, for smaller DFGs, GPS may exhibit worse performance due to the time required to load the network models. In summary, GPS effectively reduces compilation time, particularly in cases involving larger loops.

5.3 Ablation Study on Pre-scheduling Method

Ablation studies were conducted to verify the contribution of our pre-scheduling method, as depicted in Figure 8. We evaluated the performance of the GNN-based pre-scheduling method against the naive ASAP scheduling algorithm used in CRIMSON [14]. Optimal results were obtained using an ILP-based algorithm [18]. As shown in Figure 8(a), GPS achieves optimal scheduling solutions for 89.9% of the tested benchmarks. In contrast, the ASAP algorithm fails to obtain optimal solutions for most benchmarks, as illustrated in Figure 8(b). Figure 8(c) demonstrates that GPS provides better solutions for 95.2% of the benchmarks compared to the ASAP algorithm. Furthermore, Figure 8(d) indicates that the pre-scheduling method in GPS reduces the search space by an average of 92.4%. These results highlight that the use of pre-scheduling methods effectively reduces the search space and enhances search efficiency.

6 CONCLUSION

CGRA is an energy-efficient and processing-flexible parallel computing architecture. A major challenge for CGRA compilers is generating optimal loop mapping results within the shortest possible time. In this paper, we propose a fast loop mapping method called GPS, which leverages GNNs and pre-scheduling to effectively reduce compilation time while intelligently generating high-quality loop mapping results. Experimental results demonstrate that GPS achieves performance improvements ranging from 29.4% to 406.7% and significantly accelerates compilation by up to 1106.8× compared to existing compilation techniques. Future work will focus on further enhancing the GNN-based scheduling method and exploring additional polyhedral-based optimization tools.

REFERENCES

[1] John L Hennessy et al. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, 2019.

[2] Cheng Tan et al. Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 381–388. IEEE, 2020.

[3] Yixuan Luo et al. Ml-cgra: an integrated compilation framework to enable efficient machine learning acceleration on cgras. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[4] Mingyang Kou et al. Taem: Fast transfer-aware effective loop mapping for heterogeneous resources on cgra. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[5] Mingyang Kou et al. Geml: Gnn-based efficient mapping method for large loop applications on cgra. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, page 337–342, New York, NY, USA, 2022.

[6] Shail Dave et al. Ramp: Resource-aware mapping for cgras. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[7] Zhaoying Li et al. Lisa: Graph neural network based portable mapping on spatial accelerators. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 444–459. IEEE, 2022.

[8] M. J. P. Walker et al. Generic connectivity-based cgra mapping via integer linear programming. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 65–73, 2019.

[9] Xiangyu Kong et al. Mapzero: Mapping for coarse-grained reconfigurable architectures with reinforcement learning and monte-carlo tree search. In *the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[10] Ryan Kastner. benchmark, 2002.

[11] Hamzeh Mahdi et al. Epimap: using epimorphism to map applications on cgras. In *Design Automation Conference*, pages 1284–1291. ACM, 2012.

[12] Keyulu Xu et al. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA*, 2019.

[13] Jinho Lee et al. Ultra-fast cgra scheduling to enable run time, programmable cgras. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1207–1212. IEEE, 2021.

[14] Mahesh Balasubramanian et al. Crimson: Compute-intensive loop acceleration by randomized iterative modulo scheduling and optimized mapping on cgras. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3300–3310, 2020.

[15] Chris Lattner et al. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2021.

[16] William S Moses et al. Polygeist: Raising c to polyhedral mlir. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 45–59. IEEE, 2021.

[17] Nicolas Vasilache et al. Composable and modular code generation in mlir: A structured and retargetable approach to tensor compiler construction. *arXiv preprint arXiv:2202.03293*, 2022.

[18] Cheng-Tsung Hwang et al. A formal approach to the scheduling problem in high level synthesis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 10(4):464–475, 1991.

[19] Vincenzo Carletti et al. Introducing vf3: A new algorithm for subgraph isomorphism. In *Graph-Based Representations in Pattern Recognition: 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16–18, 2017, Proceedings 11*, pages 128–139. Springer, 2017.

[20] S Alexander Chin et al. Cgra-me: A unified framework for cgra modelling and exploration. In *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*, pages 184–189. IEEE, 2017.

[21] Bingfeng Mei et al. Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. In *Field Programmable Logic and Application*, pages 61–70, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[22] Manupa Karunaratne et al. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[23] Matthew R. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, 2002.

New Time-Domain Preconditioners for HB Jacobian of RF Circuits

Chenyi Tan[1], Yangfeng Su[1,*], Fan Yang[2], and Xuan Zeng[2]
[1]School of Mathematical Sciences, Fudan University, Shanghai, China
[2]State Key Laboratory of Integrated Chips and Systems, Microelectronics Department, Fudan University, Shanghai, China

***Abstract*—Harmonic balance (HB) method is a powerful frequency-domain method used in RF circuit simulations. The key point of HB method is efficiently solving the Jacobian system in Newton's method. In this paper, we first introduce a new time-domain preconditioner for HB Jacobian. Unlike existing time-domain preconditioners, which cannot balance the efficiency of solving the linear system corresponding to the preconditioner with the reduction in iteration step for strongly nonlinear circuit, the proposed preconditioner successfully addresses both aspects. We also present a new preconditioning method that extends time-domain preconditioners to circuit with distributed devices, which was previously unattainable. Finally, a matrix norm-based metric is proposed to measure the strength of circuit nonlinearity, which can help us a priori choose the appropriate preconditioner.**

***Index Terms*—RF Circuit Simulation, Harmonic Balance, HB Jacobian, Preconditioning Techniques.**

I. Introduction

HARMONIC balance (HB) method has been widely used in RF simulation and microwave applications [1]. HB method requires solving nonlinear equations in the frequency-domain, for which Newton's method is often used. Therefore, efficiently solving Jacobian matrix (HB Jacobian) at each Newton step is crucial in HB method. When solving HB Jacobian, Krylov subspace iterative methods such as GMRES method are widely employed. These iterative methods do not require the matrix to be formed explicitly and only require the matrix-vector product in every iteration [2], which has greatly improved the efficiency.

Despite its advantages, the effectiveness of the iterative solver for HB Jacobian depend critically on a good preconditioner which aims to reduce the iteration step. Existing preconditioning techniques for HB method can be divided into two types, the frequency and time-domain preconditioners. These two types of preconditioners each have their own applicable scenarios.

Frequency-domain preconditioner is first proposed by [3], in which the most widely used preconditioner in HB analysis, the frequency-domain average preconditioner is introduced. Several researches have also been carried out to improve the efficiency. A frequency-domain section-average preconditioner is introduced by [4], a better approximation for HB Jacobian is also established in [5] based on a one step correction strategy, and adaptive dynamical preconditioners are presented in [6] by discovering the special structure of HB Jacobian. Frequency-domain preconditioner works pretty well for circuit that is mildly nonlinear. However, when the circuit becomes strongly nonlinear, the number of iteration step using frequency-domain preconditioner increases significantly [5] [7].

Time-domain preconditioning technique is established in [7]. Further extensions and improvements are also provided by [8] and [9]. The greatest advantage of time-domain preconditioner is its effectiveness for circuit with strong nonlinearity. Time-domain preconditioner needs to use a matrix corresponding to a low-step finite difference scheme as approximation of part of the original Jacobian matrix. Two strategies are provided by [7], which suggests using a finite difference scheme in preconditioner fully or partially. The first strategy has advantage in reducing iteration step, but solving the preconditioned system is inefficient; the second strategy's preconditioner is easier to solve, but it requires more iteration steps. Thus, existing time-domain preconditioners cannot balance the efficiency of solving the preconditioned linear system with the reduction in the number of iterations for the iterative solver. Another limitation of existing time-domain preconditioner is the incapability of handling circuit with distributed devices, while they can be directly simulated in frequency-domain [5] [7] [10].

The main contributions of our work are summarized as follows:

- We provide a new type of time-domain hybrid finite-difference preconditioner which successfully addresses both aspects mentioned above. It has significant advantages in handling strongly nonlinear circuit.
- We provide a time-domain average preconditioner, which is suitable for circuit with distributed devices.
- We introduce a matrix norm-based metric to measure the strength of circuit nonlinearity that, to some extend, can help us a priori choose the appropriate preconditioner.

The rest of the paper is organized as follows. Section II illustrates some preliminaries of the HB method and existing preconditioners. Section III introduces our newly proposed time-domain hybrid finite-difference and average preconditioner. Section IV proposes a matrix norm-based metric to measure the strength of circuit nonlinearity. Numerical results are shown in Section V, and section VI draws a conclusion.

*Corresponding author: Yangfeng Su (email: yfsu@fudan.edu.cn)

II. Preliminaries

A. The derivation of HB Jacobian

Based on modified nodal analysis, we consider an RF circuit whose equations are given by

$$\int_{-\infty}^{t} y(t-s)\, v(s)\, \mathrm{d}s + \frac{\mathrm{d}q(v(t))}{\mathrm{d}t} + i(v(t)) + u(t) = 0,$$

where

- $v(t) \in \mathbb{R}^n$: the voltage variables in the circuit;
- $\int_{-\infty}^{t} y(t-s)\, v(s)\, \mathrm{d}s$: the distributed devices with the matrix-valued function $y(t) = 0$ for all $t < 0$;
- $q : \mathbb{R}^n \to \mathbb{R}^n$: the charge and flux storage in the circuit;
- $i : \mathbb{R}^n \to \mathbb{R}^n$: the current variables in the circuit;
- $u(t) \in \mathbb{R}^n$: the input source current waveform with period T.

Notice that the input source is T-periodic and the system is nonautonomous, the steady-state response will also be T-periodic and can be approximated as

$$v(t) \approx v_K(t) = \sum_{k=-K}^{K} V_k \exp(\jmath 2\pi k f_0 t), \quad f_0 = \frac{1}{T}.$$

In the following, by abuse of notation, we identify v_K as v and let $\{t_i\}_{i=0}^{2K+1}$ be the equidistant discretization of $[0, T]$. Take the approximated form of solution and apply numerical integration, we can collocate the equations in frequency-domain and write it in matrix form

$$F_{\text{HB}}(\mathcal{V}) = \mathcal{Y}\mathcal{V} + \Omega\Gamma q\left(\Gamma^{-1}\mathcal{V}\right) + \Gamma i\left(\Gamma^{-1}\mathcal{V}\right) + \mathcal{U} = 0, \quad (1)$$

where

- $\mathcal{Y} = \text{diag}\{Y_{-K}, ..., Y_0, ..., Y_K\}$ with Y_i's the Fourier coefficients of $y(t)$;
- $\Omega = \jmath 2\pi \cdot \text{diag}\{0, f_0 I, \ldots, K f_0 I, -K f_0 I, \ldots, -f_0 I\}$;
- $\mathcal{V} = [V_0^T, V_1^T, \ldots, V_K^T, V_{-K}^T, \ldots, V_{-1}^T]^T \in \mathbb{C}^{n(2K+1)}$;
- Γ and Γ^{-1} are N-point FFT and inverse FFT matrices respectively.

To solve equation (1), Newton's method is used to compute $\mathcal{V}$. The k-th iteration step of Newton's method is

$$\mathcal{V}^{(k+1)} = \mathcal{V}^{(k)} - J_{\text{HB}}\left(\mathcal{V}^{(k)}\right)^{-1} F_{\text{HB}}\left(\mathcal{V}^{(k)}\right).$$

For $J_{\text{HB}}\left(\mathcal{V}^{(k)}\right)$, it can be computed by the chain rule and has the following form:

$$J_{\text{HB}} = \mathcal{Y} + \Omega\Gamma\mathcal{C}\Gamma^{-1} + \Gamma\mathcal{G}\Gamma^{-1}, \quad (2)$$

where

$$\begin{aligned}\mathcal{C} &= \text{diag}\left\{\frac{\partial q}{\partial v}\Big|_{v=v(t_0)}, \ldots, \frac{\partial q}{\partial v}\Big|_{v=v(t_{2K})}\right\} \\ &= \text{diag}\{C(t_0), \ldots, C(t_{2K})\}.\end{aligned}$$

$\mathcal{G}$ can be also defined similarly.

Thus, the key point of HB method is solving the Jacobian system shown in equation (2) efficiently.

B. Frequency-Domain Preconditioner

Frequency-domain preconditioners seek sparse approximation of J_{HB} directly in the frequency-domain. Note that HB Jacobian is of the form shown in equation (2). Since $\mathcal{Y}$ and Ω are both block diagonal, the key point is to approximate $\Gamma\mathcal{C}\Gamma^{-1}$ and $\Gamma\mathcal{G}\Gamma^{-1}$.

We take frequency-domain average preconditioner, which is the preconditioner of most use in both academic and industry field, as an example. The components of frequency-domain average preconditioner are constructed as

$$\Gamma\mathcal{C}\Gamma^{-1} \approx \bar{\mathcal{C}} = I_{2K+1} \otimes \left(\frac{1}{2K+1}\sum_{i=0}^{2K} C(t_i)\right)$$

$$\Gamma\mathcal{G}\Gamma^{-1} \approx \bar{\mathcal{G}} = I_{2K+1} \otimes \left(\frac{1}{2K+1}\sum_{i=0}^{2K} G(t_i)\right)$$

And the preconditioner $\bar{J}_{\text{HB}}$ is constructed as

$$\bar{J}_{\text{HB}} = \mathcal{Y} + \Omega\bar{\mathcal{C}} + \bar{\mathcal{G}}.$$

It is obvious that $\bar{J}_{\text{HB}}$ is a block-diagonal matrix with $2K+1$ diagonal blocks of size $n \times n$, which means the preconditioned system $\bar{J}_{\text{HB}} z = r$ can be easily solved.

C. Time-Domain Preconditioner

1) Circuit without Distributed Devices: We first consider the case when no distributed devices in the circuit. Left multiply equation (2) by Γ^{-1} and right multiply by Γ, one may have

$$\Gamma^{-1}J_{\text{HB}}\Gamma = \Gamma^{-1}\Omega\Gamma\mathcal{C} + \mathcal{G}.$$

We denote $J_t = \Gamma^{-1}J_{\text{HB}}\Gamma$ and $\mathcal{D} = \Gamma^{-1}\Omega\Gamma$, the above equation can be rewritten as

$$J_t = \mathcal{D}\mathcal{C} + \mathcal{G}. \quad (3)$$

J_t is often called time-domain Jacobian for the diagonal blocks of $\mathcal{C}$ and $\mathcal{G}$ are actually variables in time-domain [7]. Time-domain preconditioning techniques seek a sparse approximation of J_t: $J_t \approx \tilde{J}_t$ and then use $\Gamma\tilde{J}_t\Gamma^{-1}$ as a preconditioner.

The special structure of $\mathcal{D}$ is the key of finding an approximation of J_t [7], it can be shown that

$$\mathcal{D} = \Gamma^{-1}\Omega\Gamma = \begin{bmatrix} 0 & \alpha_1 I_n & \alpha_2 I_n & \ldots & \alpha_{-1} I_n \\ \alpha_{-1} I_n & 0 & \alpha_1 I_n & \ldots & \alpha_{-2} I_n \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \alpha_1 I_n \\ \alpha_1 I_n & \alpha_2 I_n & \ldots & \alpha_{-1} I_n & 0 \end{bmatrix}.$$

Assuming $\alpha_0 = 0$, it can be also proved that

$$\sum_{i=-K}^{K} \alpha_i = 0.$$

Thus, $\mathcal{D}\mathcal{C}$ can be viewed as a kind of $(2K+1)$-step finite difference scheme on a equidistant grid of $[0, T]$.

The above result suggests that $\mathcal{D}\mathcal{C}$ can be approximated by a lower-step finite difference scheme. We refer to this kind of preconditioner more precisely as *time-domain finite difference preconditioner*:

$$J_t \approx \mathcal{D}_{\text{FD}}\mathcal{C} + \mathcal{G},$$

here $\mathcal{D}_{\text{FD}}\mathcal{C}$ means a lower-step finite difference scheme.

2) Circuit with Distributed Devices: When the circuit has distributed devices, we have time-domain Jacobian of the following form

$$J_t = \Gamma^{-1}\mathcal{Y}\Gamma + \mathcal{D}\mathcal{C} + \mathcal{G}.$$

Unlike equation (3), there is one more term $\Gamma^{-1}\mathcal{Y}\Gamma$ in the above equation. For distributed test problems, existing techniques do not have approach to find a sparse approximation of $\Gamma^{-1}\mathcal{Y}\Gamma$, so existing time-domain preconditioners can not be applied to this kind of circuit [5] [7] [10].

3) Two Strategies of Forming the Preconditioner: As we have mentioned, time-domain finite difference preconditioners seek a sparse approximation of $\mathcal{D}\mathcal{C}$ by replacing it with a finite difference scheme of lower-step. In practical, two types of strategies have been used in this process [7], we take Back-Euler scheme as an example. The first strategy retains the entire finite difference scheme:

$$\mathcal{D}\mathcal{C} \approx \mathcal{D}_{\text{FFD}}\mathcal{C} = \frac{1}{h}\begin{bmatrix} C(t_0) & & & & -C(t_{2K}) \\ -C(t_0) & C(t_1) & & & \\ & -C(t_1) & C(t_2) & & \\ & & \ddots & \ddots & \\ & & & -C(t_{2K-1}) & C(t_{2K}) \end{bmatrix}$$

and $J_t \approx \tilde{J}_{\text{FFD}} = \mathcal{D}_{\text{FFD}}\mathcal{C} + \mathcal{G}$, we will refer to it as *time-domain full finite difference preconditioner* in subsequent sections of this paper. The second strategy eliminates the upper right corner of the finite difference scheme:

$$\mathcal{D}\mathcal{C} \approx \mathcal{D}_{\text{PFD}}\mathcal{C} = \frac{1}{h}\begin{bmatrix} C(t_0) & & & & \\ -C(t_0) & C(t_1) & & & \\ & -C(t_1) & C(t_2) & & \\ & & \ddots & \ddots & \\ & & & -C(t_{2K-1}) & C(t_{2K}) \end{bmatrix}$$

and $J_t \approx \tilde{J}_{\text{PFD}} = \mathcal{D}_{\text{PFD}}\mathcal{C} + \mathcal{G}$. Similarly, we refer to it as *time-domain partial finite difference preconditioner*.

III. New Time-Domain Preconditioners

In this section we first introduce our newly proposed time-domain hybrid finite difference preconditioner. Then we present a new time-domain average preconditioning technique that can make time-domain preconditioner applicable to circuit with distributed devices.

A. Time-Domain Hybrid Finite Difference Preconditioner

We first discuss the strengths and weaknesses of the two existing time-domain preconditioning strategies mentioned in the last section. The full finite difference strategy performs very well for strongly nonlinear circuit but suffers from low efficiency in solving the preconditioned system as we have to compute the factorization of the entire matrix [7]. The partial finite difference strategy is highly efficient in solving the preconditioned system because the system that need to be solved is block lower triangular. However, this preconditioner tends to require more iterations for circuit with strong nonlinearity. This is because the finite difference scheme used by this preconditioner corresponds to an initial value problem for an ordinary differential equation (ODE), rather than a boundary value problem for the original system.

Our newly proposed preconditioner improves upon existing strategies by incorporating the following enhancement: *using backward differencing in the reverse time-domain at the first time step*. We still use Back-Euler scheme as an example to help readers understand it:

$$\mathcal{D}\mathcal{C} \approx \mathcal{D}_{\text{HFD}}\mathcal{C} = \frac{1}{h}\begin{bmatrix} C(t_0) & -C(t_1) & & & \\ -C(t_0) & C(t_1) & & & \\ & -C(t_1) & C(t_2) & & \\ & & \ddots & \ddots & \\ & & & -C(t_{2K-1}) & C(t_{2K}) \end{bmatrix}$$

and $J_t \approx \tilde{J}_{\text{HFD}} = \mathcal{D}_{\text{HFD}}\mathcal{C} + \mathcal{G}$. When solving the system corresponding to this preconditioner, the upper left 2×2 matrix block is solved first, followed by block forward substitution. Similar generalizations can be easily derived when using higher-step finite difference schemes such as Gear-2 scheme and we refer to this kind of preconditioner as *time-domain hybrid finite difference preconditioner*.

Combining the previous discussion on existing preconditioners, the newly proposed preconditioner has the following characteristics. First, the system that need to be solved by this preconditioner is almost block lower triangular, resulting in significantly lower computational cost compared to the time-domain full finite difference preconditioner. Second, the finite difference scheme used by this preconditioner still corresponds to a boundary value problem, providing a better approximation to the original system compared to the time-domain partial finite difference preconditioner.

B. Time-Domain Average Preconditioner

1) Frequency Behavior of Y-parameter: We denote the diagonal blocks of $\mathcal{Y}$ as Y_i's, which are the Fourier coefficients of $y(t)$. In electronic engineering, the elements of matrix Y_i are called Y-parameters. Y-parameters are generally frequency-dependent, meaning they can vary with the frequency of the applied signal. In general, for linear time-invariant (LTI) circuits, Y-parameters may exhibit similar trends at integer multiples of the fundamental frequency [11].

It is known that distributed devices in RF analysis, such as transmission lines, packages and substrates are usually linear passive components. Thus, Y-parameters at integer multiples of the fundamental frequency are generally similar or related, although there might be slight variations due to the effects of impedance mismatch, losses, and other factors. This indicates that the diagonal blocks of matrix $\mathcal{Y}$ is nearly invariant.

2) Preconditioning Procedure: In section II we have mentioned that for system with distributed devices, the time-domain Jacobian has one more term $\Gamma^{-1}\mathcal{Y}\Gamma$, it is a dense matrix and existing preconditioning techniques have not provided a sparse approximation of it. This results in existing time-domain preconditioners being unsuitable for circuit with distributed devices.

Motivated by the behavior of Y-parameter which is discussed above, we construct a new approach which is similar to the average preconditioner in frequency-domain. The new preconditioning technique can construct a time-domain preconditioner for distributed test problems. We approximate $\Gamma^{-1}\mathcal{Y}\Gamma$ as follows:

$$\begin{aligned}\Gamma^{-1}\mathcal{Y}\Gamma &\approx \Gamma^{-1}\left(I_{2K+1}\otimes\left(\frac{1}{2K+1}\sum_{i=-K}^{K}Y_i\right)\right)\Gamma\\ &= \left(F_{2K+1}^{-1}\otimes I_N\right)\left(I_{2K+1}\otimes\bar{Y}\right)\left(F_{2K+1}\otimes I_N\right)\\ &= I_{2K+1}\otimes\bar{Y},\end{aligned}$$

where F_{2K+1} is a FFT matrix of order $2K+1$.

Denote $\bar{\mathcal{Y}} = I_{2K+1}\otimes\bar{Y}$ and the time-domain Jacobian J_t with distributed devices can be approximated as

$$J_t \approx \tilde{J}_t = \bar{\mathcal{Y}} + \mathcal{D}_{\text{FD}}\mathcal{C} + \mathcal{G}, \tag{4}$$

where $\mathcal{D}_{\text{FD}}\mathcal{C}$ denotes a finite difference scheme on a equidistant grid of $[0,T]$. We refer to this kind of preconditioner as *Time-domain average preconditioner*.

IV. The Metric of Nonlinearity

Previous works have mentioned that frequency-domain preconditioners are more effective for circuit with mild nonlinearity, while time-domain preconditioners are better for circuit with strong nonlinearity [5] [7]. It is natural to ask how to a priori assess the strength of circuit nonlinearity in order to select an appropriate preconditioner. In this section we introduce a matrix norm-based metric to help answer this question.

A. Block Circulant Structure in HB Jacobian

First, we have to review the special structure of HB Jacobian. The term $\Gamma\mathcal{C}\Gamma^{-1}$ and $\Gamma\mathcal{G}\Gamma^{-1}$ are both block circulant and this has been mentioned in many previous works [3] [10] [12]. Take $\Gamma\mathcal{G}\Gamma^{-1}$ as an example:

$$\Gamma\mathcal{G}\Gamma^{-1} = \begin{bmatrix} \mathcal{G}_0 & \mathcal{G}_1 & \cdots & \mathcal{G}_{2K-1} & \mathcal{G}_{2K} \\ \mathcal{G}_{2K} & \mathcal{G}_0 & \mathcal{G}_1 & & \mathcal{G}_{2K-1} \\ \vdots & \mathcal{G}_{2K} & \mathcal{G}_0 & \ddots & \vdots \\ \mathcal{G}_2 & & \ddots & \ddots & \mathcal{G}_1 \\ \mathcal{G}_1 & \mathcal{G}_2 & \cdots & \mathcal{G}_{2K} & \mathcal{G}_0 \end{bmatrix}, \tag{5}$$

where $\mathcal{G}_i$'s are all $n\times n$ block matrix and can be obtain easily by block iFFTs. $\mathcal{C}_i$'s are of the same form.

Actually, if the circuit is pure linear, which means $C(t) = \frac{\partial q}{\partial v}(t)$ and $G(t) = \frac{\partial i}{\partial v}(t)$ are both time-invariant. In this case we have

$$\mathcal{C}_i = \mathcal{G}_i = \mathbf{0}_{n\times n}\,(i = 1,2,...,2K).$$

Thus, $\Gamma\mathcal{C}\Gamma^{-1}$ and $\Gamma\mathcal{G}\Gamma^{-1}$ are both block diagonal which means the HB Jacobian J_{HB} is block diagonal. Previous works have also mentioned that for circuit with mild nonlinearity, J_{HB} is block diagonal dominant while the dominance will deteriorate as the nonlinearity increases [5] [10].

B. Proposed Metric of Nonlinearity

The discussion above inspires us that we can use the degree of diagonal block dominance to measure the strength of nonlinearity. Since $\mathcal{C}$ and $\mathcal{G}$ represent different circuit components, we present the following matrix norm-base metric which can measure the strength of nonlinearity of circuit components corresponding to matrix $\mathcal{C}$ and $\mathcal{G}$ respectively as following:

$$\begin{aligned}\text{LinRate}\,(\mathcal{C}) &= \frac{\|\mathcal{C}_0\|_\infty}{\sum_{i=1}^{2K}\|\mathcal{C}_i\|_\infty},\\ \text{LinRate}\,(\mathcal{G}) &= \frac{\|\mathcal{G}_0\|_\infty}{\sum_{i=1}^{2K}\|\mathcal{G}_i\|_\infty}.\end{aligned} \tag{6}$$

The smaller the two values, the lower the dominance of the main diagonal blocks in the block circulant matrix, indicating stronger nonlinearity for the corresponding circuit components; conversely, the larger the two values, the milder the nonlinearity.

We first suppose that there is no distributed devices in the circuit. Review the form of HB Jacobian shown in equation (2), matrix $\Gamma\mathcal{C}\Gamma^{-1}$ will be left-multiplied by a block diagonal matrix Ω of coefficients resulting from differentiation. For each harmonic, the corresponding row of the block matrix will be multiplied by the coefficient $\jmath 2k\pi f_0$, which corresponds to the harmonic order. In RF simulation, the fundamental frequency f_0 is generally of a large order, typically in the range of MHz, GHz, or even higher frequencies. Thus $|\jmath 2k\pi f_0|\|\mathcal{C}_i\|$'s are often significantly larger than $\|\mathcal{G}_i\|$'s when $k \neq 0$. So for rows corresponding to non-zero order harmonics, the diagonal block dominance is almost entirely determined by LinRate $(\mathcal{C})$, while for the first row which corresponds to the zero-order harmonic it is determined by LinRate $(\mathcal{G})$. Thus, when using this metric to help test the nonlinearity of the whole circuit in order to a priori choose the appropriate preconditioner, both LinRate $(\mathcal{C})$ and LinRate $(\mathcal{G})$ have to be computed.

Besides, for circuit with distributed devices, an additional block diagonal matrix $\mathcal{Y}$ needs to be added. The elements of $\mathcal{Y}$ have different dimension compared to those of $\mathcal{C}$ and $\mathcal{G}$, and their magnitude is also generally larger. In this case, the circuit properties become more complex, making it difficult to have a universal metric to measure their nonlinear characteristics. However, numerical experiments in the next section have shown that the proposed metric can still serve to assess the degree of nonlinearity for such circuit.

TABLE I
Details of The Test Cases

Case	Circuit Size	Number of Harmonics	Distributed Devices
1	85	21	×
2	400	41	×
3	44	65	×
4	615	81	×
5	18	321	×
6	52	101	×
7	15	11	✓
8	15	101	✓
9	15	41	✓
10	25	21	✓

TABLE II
GMRES ITERATION STEPS OF THE PROPOSED PRECONDITIONER AND EXISTING ONES

Case	Frequency-domain Preconditioner		Time-domain Preconditioner				
	Avg		TF		TP		TH (Ours)
	Iter_{Avg}	$\text{Iter}_{\text{Avg}}/\text{Iter}_{\text{TH}}$	Iter_{TF}	$\text{Iter}_{\text{TF}}/\text{Iter}_{\text{TH}}$	Iter_{TP}	$\text{Iter}_{\text{TP}}/\text{Iter}_{\text{TH}}$	Iter_{TH}
1(+)	233	**2.10**	98	0.88	170	**1.53**	**111**
2(+)	156	**1.97**	58	0.73	300(*)	*	**79**
3(+)	300(*)	*	46	**1.02**	71	**1.58**	**45**
4(+)	300(*)	*	164	**1.14**	300(*)	*	**144**
5	21	0.45	42	0.91	56	**1.22**	**46**
6	67	0.69	77	0.79	300(*)	*	**97**

* Fails to converge after the maximum step of iterations
+ means strongly nonlinear case

TABLE III
COMPUTING TIME (TIME UNIT: SECOND)

Case	Frequency-domain Preconditioner		Time-domain Preconditioner				
	Avg		TF		TP		TH (Ours)
	T_{Avg}	$\text{T}_{\text{Avg}}/\text{T}_{\text{TH}}$	T_{TF}	$\text{T}_{\text{TF}}/\text{T}_{\text{TH}}$	T_{TP}	$\text{T}_{\text{TP}}/\text{T}_{\text{TH}}$	T_{TH}
1(+)	2.7821	**2.14**	2.4403	**1.87**	2.0963	**1.61**	**1.3043**
2(+)	18.6134	**2.19**	19.3762	**2.28**	23.6477(*)	*	**8.4924**
3(+)	3.6597(*)	*	2.0175	**2.35**	1.7597	**2.05**	**0.8601**
4(+)	47.3242(*)	*	70.9859	**2.30**	55.2421(*)	*	**30.8505**
5	0.5627	0.53	1.4251	**1.37**	1.1873	**1.12**	**1.0659**
6	1.6785	0.30	7.4829	**1.33**	16.3049(*)	*	**5.6181**

* Fails to converge after the maximum step of iterations
+ means strongly nonlinear case

V. NUMERICAL EXPERIMENTS

In this section we would like to highlight the advantage of the proposed preconditioners when dealing with strongly nonlinear circuit (circuits with or without distributed devices are both included) and the feasibility of using the proposed matrix norm-based metric to a priori assess the strength of circuit nonlinearity by numerical experiments. Also, we present a test to verify that diagonal blocks of $\mathcal{Y}$ are nearly invariant using a similar metric of the metric shown in equation (6) and this can ensure that the idea of time-domain average preconditioner is reasonable. The numerical experiments were performed in MATLAB Version 9.13.0.2320565 (R2022b) on a standard desktop with the following specifications: 12th Gen Intel(R) Core(TM) i5-12600KF 3.70 GHz 16GB RAM.

The case information is shown in Table I, including the circuit size, number of harmonics used in simulation and whether the circuits have distributed devices.

A. Time-domain Hybrid Finite Difference Preconditioner vs. Existing Preconditioners

Firstly, we compare our proposed preconditioner with existing preconditioners that all have been mentioned in previous sections on circuits without distributed devices. For frequency-domain preconditioner, we use average preconditioner. The TF, TP and TH in the table stands for time-domain full, partial and hybrid finite difference preconditioner respectively. The finite difference scheme used for approximating $\mathcal{DC}$ is Back-Euler scheme. Table II and III show the GMRES iteration step and computing time for different preconditioners.

We first compare our proposed preconditioner to existing time-domain preconditioners. In all the cases, the number of GMRES iteration step of our proposed preconditioner is fewer than that of the time-domain partial finite difference preconditioner with $1.4\times$ iteration reduction on average, while the iteration step is similar to or even fewer compared to time-domain full finite difference preconditioner. For those cases in which full finite difference gain less iterations, the number is not excessively fewer. The advantage is also reflected in terms of computing time, the time-domain hybrid finite difference preconditioner has advantages over the other two time-domain preconditioners and achieves $1.9\times$ and $1.6\times$ speed up on average.

At the same time, compared to frequency-domain average preconditioner, the newly proposed preconditioner also achieves $2.0\times$ iteration reduction and $2.2\times$ computing time speed up on average in all strongly nonlinear cases (case 1-4). Even in cases where frequency-domain average preconditioner does not converge, convergence can be achieved in a few steps with our proposed preconditioner. In circuits with relatively mild nonlinearity (case 5-6), frequency-domain average preconditioner is notably superior to all time-domain preconditioners. This aligns with the experiment observation of prior related works for HB method [5] [7].

Overall, the newly proposed preconditioner shows significant advantages when dealing with circuit that exhibits strong nonlinearity.

B. Time-Domain Average Preconditioner Applied to Strongly Nonlinear Distributed Test Cases

In this part, we primarily demonstrate the value of the time-domain average preconditioner through experiments. We apply time-domain average preconditioner based on equation (4) to the four strongly nonlinear cases with distributed devices. The hybrid finite difference scheme is used to approximate $\mathcal{DC}$ in experiments. Table IV shows GMRES iteration step when different preconditioners are applied to these cases.

TABLE IV
TEST FOR STRONGLY NONLINEAR CIRCUITS WITH DISTRIBUTED DEVICES

Case	Frequency-domain Average Preconditioner		Time-domain Average Preconditioner
	$\text{Iter}_{\text{FAvg}}$	$\text{Iter}_{\text{FAvg}}/\text{Iter}_{\text{TAvg}}$	$\text{Iter}_{\text{TAvg}}$
7	37	**2.18**	17
8	38	**1.36**	28
9	155	**3.04**	51
10	80	**3.08**	26

It can be seen that in all cases, the GMRES iteration step of time-domain average preconditioner is significantly reduced compared to that of frequency-domain average preconditioner, with 2.4× iteration reduction on average. The experiment confirms that the newly proposed preconditioning technique makes time-domain preconditioners applicable to circuit with distributed devices and exhibits clear advantage over frequency-domain average preconditioner when the circuit is also strongly nonlinear.

C. Preliminary Nonlinearity Test

In this subsection, we conduct a preliminary nonlinearity test on all the test cases using the aforementioned matrix norm-based metric, categorizing them into strongly nonlinear cases and mildly nonlinear cases, which is shown in Table V.

It is evident that for circuits with strong nonlinearity, at least one of the values of $\text{LinRate}\,(\mathcal{C})$ and $\text{LinRate}\,(\mathcal{G})$ will be noticeably small compared to that of circuits with mild nonlinearity (highlighted in bold in Table V). Based on our experimental observations, when $\text{LinRate}\,(\mathcal{C}) \lesssim$ 1e+13 or $\text{LinRate}\,(\mathcal{G}) \lesssim$ 1e+02, the circuit can be considered to have strong a priori nonlinearity; otherwise, the a priori nonlinearity is relatively weak.

From the experiments in the previous two subsections, it is also obvious that cases with mild a priori nonlinearity require fewer iterations using frequency-domain average preconditioner, while those with strong nonlinearity require fewer iterations using time-domain preconditioners. This shows that the strength of circuit nonlinearity of these cases is consistent with the a priori judgments, even when the circuit has distributed devices (case 7-10). The experiment indicates that the metric we proposed can effectively provide a priori judgments on the strength of circuit nonlinearity to a certain extent, and also help us choose the appropriate preconditioner.

TABLE V
TEST OF NONLINEARITY

Case	$\text{LinRate}\,(\mathcal{C})$	$\text{LinRate}\,(\mathcal{G})$	Consistent with the prior
Strongly Nonlinear Case			
1	**4.0363e+10**	**4.7570**	✓
2	**2.2216e+13**	990.4579	✓
3	**1.2828e+13**	**26.9139**	✓
4	**8.8784e+09**	**2.7155**	✓
7	4.9654e+15	**1.0531**	✓
8	9.8265e+14	**0.9881**	✓
9	1.8495e+15	**0.2581**	✓
10	3.2005e+15	**0.9476**	✓
Mildly Nonlinear Case			
5	2.2636e+14	4.1668e+03	✓
6	3.2005e+15	3.1730e+05	✓

TABLE VI
TEST OF VARIANCE OF THE DIAGONAL BLOCKS OF $\mathcal{Y}$

Case	$\text{InvarRate}\,(\mathcal{Y})$
7	3.2358e+07
8	2.9700e+07
9	1.7716e+09
10	1.0048e+11

D. Test of Variance of the Diagonal Blocks of $\mathcal{Y}$

In this part, we aim to show that the diagonal blocks of $\mathcal{Y}$ are nearly invariant using the same strategy of the metric shown in equation (6). It is easy to check that $\Gamma^{-1}\mathcal{Y}\Gamma$ is also block circulant. Denote the circulant blocks as $\mathcal{Y}_i$'s, and $\mathcal{Y}_i$'s can be computed similarly as $\mathcal{G}_i$'s and $\mathcal{C}_i$'s by performing block FFTs, which have been mentioned in equation (5) in section IV. Similarly, we can use the dominance of the main diagonal blocks of $\Gamma^{-1}\mathcal{Y}\Gamma$ to determine the variance of diagonal blocks of $\mathcal{Y}$. We introduce the following metric:

$$\text{InvarRate}\,(\mathcal{Y}) = \frac{\|\mathcal{Y}_0\|_\infty}{\sum_{i=1}^{2K}\|\mathcal{Y}_i\|_\infty}.$$

Table VI shows the values of $\text{InvarRate}\,(\mathcal{Y})$ for all cases with distributed devices. The values corresponding to this metric are very large, indicating that the diagonal blocks of matrix $\mathcal{Y}$ are nearly invariant. Thus, the idea of time-domain average preconditioner is reasonable.

VI. CONCLUSION

In this paper, two new time-domain preconditioners are proposed. Compared to existing time-domain preconditioners, the proposed preconditioning methods not only show significant advantages for circuit with strong nonlinearity but also enable the use of time-domain preconditioners for circuit with distributed devices. In addition, we present a matrix norm-based metric which can assist us in the preliminary selection of which preconditioner to use. In the future, we aim to conduct more in-depth mathematical research on HB-related subproblems to enhance the efficiency of the HB method for RF circuit simulation.

ACKNOWLEDGMENT

This research is partly supported by National Key R&D Program of China 2020YFA0711900, 2020YFA0711902, partly by National Natural Science Foundation of China (NSFC) research projects 62141407, 61929102 and 62090025. The authors also sincerely acknowledge Primarius for their pivotal technical collaboration in this research.

REFERENCES

[1] K. S. Kundert, "Introduction to RF simulation and its application," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 9, pp. 1298–1319, 1999.

[2] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[3] P. Feldmann, B. Melville, and D. Long, "Efficient frequency domain analysis of large nonlinear analog circuits," in *Proceedings of Custom Integrated Circuits Conference*. IEEE, 1996, pp. 461–464.

[4] D. Long, R. Melville, K. Ashby, and B. Horton, "Full-chip harmonic balance," in *Proceedings of CICC 97-Custom Integrated Circuits Conference*. IEEE, 1997, pp. 379–382.

[5] W. Dong and P. Li, "Hierarchical harmonic-balance methods for frequency-domain analog-circuit analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 12, pp. 2089–2101, 2007.

[6] M. M. Gourary, S. G. Rusakov, S. L. Ulyanov, M. M. Zharov, K. K. Gullapalli, and B. J. Mulvaney, "The enchancing of efficiency of the harmonic balance analysis by adaptation of preconditioner to circuit nonlinearity," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*, 2000, pp. 537–540.

[7] O. J. Nastov, "Spectral methods for circuit analysis," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[8] O. J. Nastov and J. K. White, "Time-mapped harmonic balance," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 641–646.

[9] F. Veersé, "Efficient iterative time preconditioners for harmonic balance RF circuit simulation," in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No. 03CH37486)*. IEEE, 2003, pp. 251–254.

[10] A. Mehrotra and A. Somani, "A robust and efficient harmonic balance (hb) using direct solution of hb jacobian," in *Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 370–375.

[11] C. Bowick, *RF circuit design*. Elsevier, 2011.

[12] L. Han, X. Zhao, and Z. Feng, "An adaptive graph sparsification approach to scalable harmonic balance analysis of strongly nonlinear post-layout RF circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 173–185, 2014.